CS6501 - Internet programming

	Unit- I **Part - A**
1	**Define Java.** Java is a programming language expressly designed for use in the distributed environment of the Internet. It was designed to have the "look and feel" of the C++ language, but it is simpler to use than C++ and enforces an object-oriented programming model.
2.	**What is a Class?** Class is a template for a set of objects that share a common structure and a common behaviour.
3.	**What is an Object?** Object is an instance of a class. It has state, behaviour and identity. It is also called as an instance of a class.
4.	**What is an Instance?** An instance has state, behaviour and identity. The structure and behaviour of similar classes are defined in their common class. An instance is also called as an object.
5.	**What are different types of access modifiers (Access specifiers)?** Access specifiers are keywords that determine the type of access to the member of a class. These keywords are for allowing privileges to parts of a program such as functions and variables. These are: *public*: Anything declared as public can be accessed from anywhere. *private*: Anything declared as private can't be seen outside of its class. *protected*: Anything declared as protected can be accessed by classes in the same package and subclasses in the there packages. *default modifier* : Can be accessed only to classes in the same package.
6.	**What is method overloading and method overriding?** Method overloading: When a method in a class having the same method name with different arguments is said to be method overloading. Method overriding: When a method in a class having the same method name with same arguments is said to be method overriding.
7.	**List the access specifier used in JAVA?** Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors. The four access levels are: • Visible to the package. the default. No modifiers are needed. • Visible to the class only (**private**). • Visible to the world (**public**). • Visible to the package and all subclasses (**protected**).
8.	**What is the difference between Array and vector?** Array is a set of related data type and static whereas vector is a growable array of objects and dynamic
9.	**What is a package?** A package is a collection of classes and interfaces that provides a high-level layer of access protection and name space management.
10.	**What is meant by Inheritance?** Inheritance is a relationship among classes, wherein one class shares the structure or behaviour defined in another class. This is called Single Inheritance. If a class shares the structure or behaviour from multiple classes, then it is called Multiple Inheritance. Inheritance defines "is-a" hierarchy among classes in which one subclass inherits from one or more generalised superclasses.
11.	**What is an Abstract Class?** Abstract class is a class that has no instances. An abstract class is written with the expectation that its concrete subclasses will add to its structure and behaviour, typically by implementing its abstract operations.
12.	**What are inner class and anonymous class?** **Inner class:** classes defined in other classes, including those defined in methods are called inner classes. An inner

	class can have any accessibility including private. **Anonymous class:** Anonymous class is a class defined inside a method without a name and is instantiated and declared in the same place and cannot have explicit constructors.
13.	**Define interface and write the syntax of the Interface.** Interface is an outside view of a class or object which emphaizes its abstraction while hiding its structure and secrets of its behaviour. **Syntax**: *[visibility] interface InterfaceName [extends other interfaces] {* *constant declarations* *abstract method declarations* *}*
14.	**What is the difference between abstract class and interface?** **a)** All the methods declared inside an interface are abstract whereas abstract class must have at least one abstract method and others may be concrete or abstract. **b)** In abstract class, key word abstract must be used for the methods whereas interface we need not use that keyword for the methods. **c)** Abstract class must have subclasses whereas interface can't have subclasses.
15.	**What is an exception?** An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.
16.	**What is meant by JAVA package?(Nov/Dec 2014)** Package represents a collection of classes, methods and interface. The name of the package must be written as the first statement in the java source program. The syntax of specifying the package in the java program is: package name_of_package
17.	**What are the types of Exceptions in Java?** There are two types of exceptions in Java, unchecked exceptions and checked exceptions. ***Checked exceptions:*** A checked exception is some subclass of Exception (or Exception itself), excluding class RuntimeException and its subclasses. Each method must either handle all checked exceptions by supplying a catch clause or list each unhandled checked exception as a thrown exception. ***Unchecked exceptions:*** All Exceptions that extend the RuntimeException class are unchecked exceptions. Class Error and its subclasses also are unchecked.
18.	**What are the different ways to handle exceptions?** There are two ways to handle exceptions: • Wrapping the desired code in a try block followed by a catch block to catch the exceptions. • List the desired exceptions in the throws clause of the method and let the caller of the method handle those exceptions.

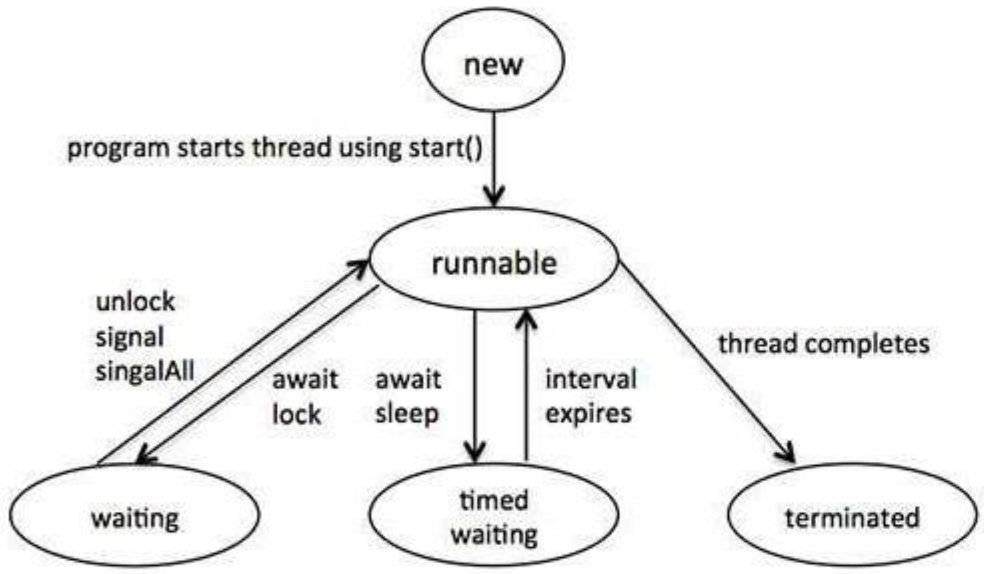
19.	**How to create custom exceptions?** By Extending the Exception class or one of its subclasses.
	class MyException extends Exception {
	public MyException() { super(); }
	public MyException(String s) { super(s);
	} }

20.	**Write the properties of Threads.(Nov/Dec 2014).** • Thread Priority • Deamon Thread • Thread group
21.	**What is multi-threaded programming?(Nov/Dec 2014)**

Multithreading is the ability of a program or an operating system process to manage its use by more than one user at a time and to even manage multiple requests by the same user without having to have multiple copies of the programming running in the computer.

22. Write the life cycle of thread.

A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. Following diagram shows complete life cycle of a thread.



23. What is daemon thread and which method is used to create the daemon thread?

A daemon thread is a thread, that does not prevent the **JVM** from exiting when the program finishes but the thread is still running. An example for a daemon thread is the garbage collection. You can use the setDaemon() method to change the Thread daemon properties

24. What is the purpose of toString() method in java ?

The toString() method returns the string representation of any object. If you print any object, java compiler internally invokes the toString() method on the object. So overriding the toString() method, returns the desired output, it can be the state of an object etc. depends on your implementation.

25. What is immutable string in java?

In java, **string objects are immutable**. Immutable simply means unmodifiable or unchangeable. Once string object is created its data or state can't be changed but a new string object is created.

Eg:

```
class Testimmutablestring{
public static void main(String args[]){
 String s="Sachin";
s.concat(" Tendulkar");//concat() method appends the string at the end
System.out.println(s);//will print Sachin because strings are immutable objects
}
```

26. Define assert .

Java assertion feature allows developer to put "assert" statements in Java source code to help unit testing and debugging.

An "assert" statement has the following format:

assert boolean_expression : string_expression;

When this statement is executed:

If boolean_expression evaluates to true, the statement will pass normally.

If boolean_expression evaluates to false, the statement will fail with an "AssertionError" exception.

27. Define Applet.

An applet is a small Internet-based program written in Java, a programming language for the Web, which can be downloaded by any computer. The applet is also able to run in HTML. The applet is usually embedded in an HTML page on a Web site and can be executed from within a browser.

28. Define transient and volatile Modifiers.

Java defines two interesting type modifiers: transient and volatile. These modifiers are usedto handle somewhat specialized situations. When an instance variable is declared as transient, then its value need not persist when an object is stored. For example:

```
class T {
transient int a; // will not persist
int b; // will persist
}
```

Here, if an object of type T is written to a persistent storage area, the contents of a would not be saved, but the contents of b would.

29. What is use of the run-time operator *instanceof*.

The instanceof operator has this general form:

objref instanceof type

Here, objref is a reference to an instance of a class, and type is a class type. If objref is of the specified type or can be cast into the specified type, then the instanceof operator evaluates to true. Otherwise, its result is false. Thus, instanceof is the means by which your program canobtain run-time type information about an object

30. How to Enabling and Disabling Assertion Options?

When executing code, you can disable assertions by using the **-da** option. You can enable or disable a specific package by specifying its name after the **-ea** or **-da** option. For example, to enable assertions in a package called MyPack, use

-ea:MyPack

To disable assertions in MyPack, use

-da:MyPack

31. Define String Constructors.

The String class supports several constructors. To create an empty String, you call the default constructor. For example,

String s = new String();

will create an instance of String with no characters in it. Frequently, you will want to create strings that have initial values. The String class provides a variety of constructors to handle this. To create a String initialized by an array of characters, use the constructor shown here:

String(char chars[ ])

Here is an example:

```
char chars[] = { 'a', 'b', 'c' };
String s = new String(chars);
```

This constructor initializes s with the string “abc”.

32. What are the String Comparison?

The String class includes several methods that compare strings or substrings within strings.

equals( ) and equalsIgnoreCase( )	To compare two strings for equality, use equals( ). To perform a comparison that ignores case differences, call equalsIgnoreCase( ).
regionMatches( )	The regionMatches( ) method compares a specific region inside a string with another specific region in another string.
startsWith( ) and endsWith( )	The startsWith( ) method determines whether a given String begins with a specified string. Conversely, endsWith( ) determines whether the String in question ends with a specified string.
equals( ) Versus = =	The equals( ) method compares the characters inside a String object. The =

	= operator compares two object references to see whether they refer to the same instance.
compareTo( )	to simply know whether two strings are identical

33. List the name of methods for modifying string.

- substring( )
- concat( )
- replace( )
- trim( )

PART – B

1. (i) Describe the concepts of OOP.(5)

Object Oriented Programming is a paradigm that provides many concepts such as **inheritance**, **data binding**, **polymorphism** etc.

Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Object

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

Class

Collection of objects is called class. It is a logical entity.

Inheritance

When one object acquires all the properties and behaviours of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism

When one task is performed by different ways i.e. known as polymorphism. For example: to convense the customer differently, to draw something e.g. shape or rectangle etc. In java, we use method overloading and method overriding to achieve polymorphism. Another example can be to speak something e.g. cat speaks meaw, dog barks woof etc.

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing. In java, we use abstract class and interface to achieve abstraction.

Encapsulation

Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

(ii) What is meant by overriding method? Give example.(5)

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in java**.

In other words, If subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as method overriding.

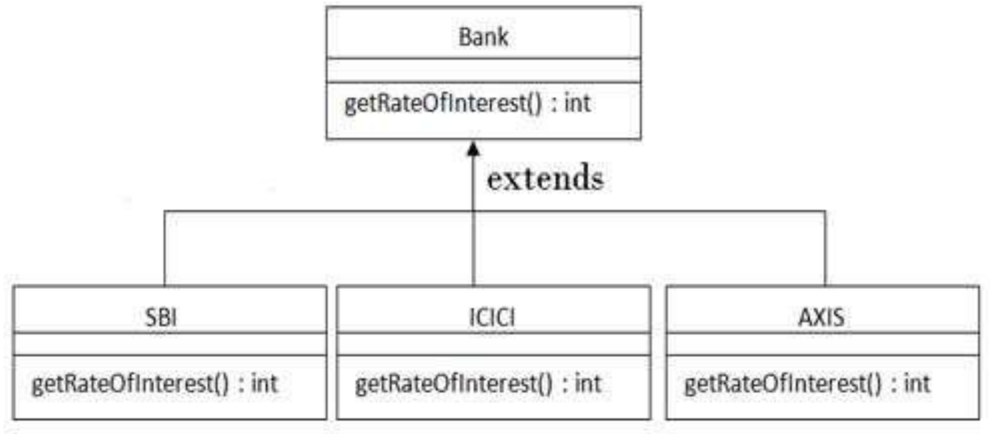
Usage of Java Method Overriding

- Method overriding is used to provide specific implementation of a method that is already provided by its super class.
- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

1. method must have same name as in the parent class
2. method must have same parameter as in the parent class.
3. must be IS-A relationship (inheritance).

Consider a scenario, Bank is a class that provides functionality to get rate of interest. But, rate of interest varies according to banks. For example, SBI, ICICI and AXIS banks could provide 8%, 7% and 9% rate of interest.


Bank
getRateOfInterest() : int
extends
SBI
getRateOfInterest() : int
ICICI
getRateOfInterest() : int
AXIS
getRateOfInterest() : int

```
class Bank{
int getRateOfInterest(){return 0;}
}

class SBI extends Bank{
int getRateOfInterest(){return 8;}
}

class ICICI extends Bank{
int getRateOfInterest(){return 7;}
}
class AXIS extends Bank{
int getRateOfInterest(){return 9;}
}

class Test2{
public static void main(String args[]){
SBI s=new SBI();
ICICI i=new ICICI();
AXIS a=new AXIS();
```

```
    System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());
    System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());
    System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());
    }
    }
Output:
SBI Rate of Interest: 8
ICICI Rate of Interest: 7
      AXIS Rate of Interest: 9
```

(iii) Write a JAVA program to reverse the given number.(6)

```
import java.util.*;
public class RevNumString
{
   public static void main(String[] args)
   {
      Scanner scanner = new Scanner(System.in);
      System.out.println("Please enter a number: ");
      int num = scanner.nextInt();
      System.out.println("Please enter a string: ");
      String str = scanner.next();
      RevNumString rns = new RevNumString();
      int revNum = rns.reverse(num);
      String revStr = rns.reverse(str);
      System.out.printf("\n The reverse of number %d is %d ", num, revNum);
      System.out.printf("\n The reverse of string '%s' is '%s' ", str, revStr);
   }
      // Method to return the reverse of a number
      public int reverse(int num) {
      int revNum = 0;
      while (num > 0) {
         int rem = num % 10;
         revNum = (revNum * 10) + rem;
         num = num / 10;
      }
      return revNum;
   }
      // Method to return the reverse of a string
      public String reverse(String str) {
      StringBuilder revStr = new StringBuilder();
      for (int i = str.length()-1; i >= 0; i--) {
         revStr.append(str.charAt(i));
      }
      return revStr.toString();
   }
}
```

Program Output:

```
Please enter a number:
1234
Please enter a string:
Java

The reverse of number 1234 is 4321
The reverse of string 'Java' is 'avaJ'
```

2. **(i) What is meant by package? How it is created and implemented in JAVA.(8)**

Packages are used in Java in order to prevent naming conflicts, to control access, to make searching/locating and usage of classes, interfaces, enumerations and annotations easier, etc.

A Package can be defined as a grouping of related types(classes, interfaces, numerations and annotations) providing access protection and name space management.
Some of the existing packages in Java are::

- **java.lang** - bundles the fundamental classes
- **java.io** - classes for input , output functions are bundled in this package

Programmers can define their own packages to bundle group of classes/interfaces, etc. It is a good practice to group related classes implemented by you so that a programmer can easily determine that the classes, interfaces, enumerations, annotations are related.
Since the package creates a new namespace there won't be any name conflicts with names in other packages. Using packages, it is easier to provide access control and it is also easier to locate the related classes.

Creating a package:
When creating a package, you should choose a name for the package and put a package statement with that name at the top of every source file that contains the classes, interfaces, enumerations, and annotation types that you want to include in the package.

The package statement should be the first line in the source file. There can be only one package statement in each source file, and it applies to all types in the file.

If a package statement is not used then the class, interfaces, enumerations, and annotation types will be put into an unnamed package.

Example:
Let us look at an example that creates a package called animals. It is common practice to use lowercased names of packages to avoid any conflicts with the names of classes, interfaces.
Put an interface in the package *animals*:

```
/* File name : Animal.java */
package animals;

interface Animal {
   public void eat();
   public void travel();
}
```

Now, put an implementation in the same package *animals*:

```
package animals;

/* File name : MammalInt.java */
public class MammalInt implements Animal{

  public void eat(){
    System.out.println("Mammal eats");
  }

  public void travel(){
    System.out.println("Mammal travels");
  }

  public int noOfLegs(){
    return 0;
  }

  public static void main(String args[]){
    MammalInt m = new MammalInt();
    m.eat();
    m.travel();
  }
}
```

Now, you compile these two files and put them in a sub-directory called animals and try to run as follows:

```
$ mkdir animals
$ cp Animal.class MammalInt.class animals
$ java animals/MammalInt
Mammal eats
Mammal travels
```

The import Keyword:
If a class wants to use another class in the same package, the package name does not need to be used. Classes in the same package find each other without any special syntax.
Example:
Here, a class named Boss is added to the payroll package that already contains Employee. The Boss can then refer to the Employee class without using the payroll prefix, as demonstrated by the following Boss class.

```
 package payroll;
public class Boss
{
     public void payEmployee(Employee e)
    {
            e.mailCheck();
        }
    }
```

What happens if Boss is not in the payroll package? The Boss class must then use one of the following techniques for referring to a class in a different package.

- The fully qualified name of the class can be used. For example:
  payroll.Employee
- The package can be imported using the import keyword and the wild card (*). For example:
  import payroll.;*
- The class itself can be imported using the import keyword. For example:
  import payroll.Employee;

Note: A class file can contain any number of import statements. The import statements must appear after the package statement and before the class declaration.

The Directory Structure of Packages:

Two major results occur when a class is placed in a package:

- The name of the package becomes a part of the name of the class, as we just discussed in the previous section.
- The name of the package must match the directory structure where the corresponding bytecode resides.

Here is simple way of managing your files in Java:
Put the source code for a class, interface, enumeration, or annotation type in a text file whose name is the simple name of the type and whose extension is .java. For example:

```
// File Name : Car.java
package vehicle;
public class Car {
  // Class implementation.
}
```

Now, put the source file in a directory whose name reflects the name of the package to which the class belongs:

....\vehicle\Car.java

Now, the qualified class name and pathname would be as below:

- Class name -> vehicle.Car
- Path name -> vehicle\Car.java (in windows)

In general, a company uses its reversed Internet domain name for its package names. Example: A company's Internet domain name is apple.com, then all its package names would start with com.apple. Each component of the package name corresponds to a subdirectory.

Example: The company had a com.apple.computers package that contained a Dell.java source file, it would be contained in a series of subdirectories like this:

....\com\apple\computers\Dell.java

At the time of compilation, the compiler creates a different output file for each class, interface and enumeration defined in it. The base name of the output file is the name of the type, and its extension is .class

For example:

```
// File Name: Dell.java
package com.apple.computers;
public class Dell{
 }
class Ups{

}
```

Now, compile this file as follows using -d option:

```
$javac -d . Dell.java
```

(ii) Write a JAVA program to find the smallest number in the given list. (8)

```
import java.util.Scanner;
class group{
public static void main(String arng[]){
int value[]= new int[5];
int temp,i;
Scanner data = new Scanner(System.in);
System.out.println("Enter 5 element of array" );
// Enhanced for loop
for(i=0; i < 5; i++ )
value[i] = data.nextInt();
// finding smallest number
temp = value[0];
for(i=0; i < 5; i++ )
{
if(temp < value[i])
continue;
else
temp = value[i];
}
System.out.println("Smallest number in array is "+temp);
}
}
```

Output:-

```
Enter 5 element of array
56
84
95
12
32
Smallest number in array is 12
```

Output:-

```
Enter 5 element of array
48
124
```

```
20
54
14
Smallest number in array is 14
```

3. **(i) What is meant by interface? How it is declared and implemented in JAVA.Give example.(12)**

An interface is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

An interface is not a class. Writing an interface is similar to writing a class, but they are two different concepts. A class describes the attributes and behaviors of an object. An interface contains behaviors that a class implements.

Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

An interface is similar to a class in the following ways:

- **(ii)** An interface can contain any number of methods.
- **(iii)** An interface is written in a file with a **.java** extension, with the name of the interface matching the name of the file.
- **(iv)** The bytecode of an interface appears in a **.class** file.
- **(v)** Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

However, an interface is different from a class in several ways, including:

- **(vi)** You cannot instantiate an interface.
- **(vii)** An interface does not contain any constructors.
- **(viii)** All of the methods in an interface are abstract.
- **(ix)** An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- **(x)** An interface is not extended by a class; it is implemented by a class.
- **(xi)** An interface can extend multiple interfaces.

Declaring Interfaces:

The **interface** keyword is used to declare an interface. Here is a simple example to declare an interface:

Example:

Let us look at an example that depicts encapsulation:

```
/* File name : NameOfInterface.java */
import java.lang.*;
//Any number of import statements

public interface NameOfInterface
```

```
{
  //Any number of final, static fields
  //Any number of abstract method declarations\
}
```

Interfaces have the following properties:

(xii) An interface is implicitly abstract. You do not need to use the **abstract** keyword when declaring an interface.

(xiii) Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.

Methods in an interface are implicitly public.

Example:

```
/* File name : Animal.java */
interface Animal {

  public void eat();
  public void travel();
}
```

Implementing Interfaces:

When a class implements an interface, you can think of the class as signing a contract, agreeing to perform the specific behaviors of the interface. If a class does not perform all the behaviors of the interface, the class must declare itself as abstract.

A class uses the **implements** keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

```
/* File name : MammalInt.java */
public class MammalInt implements Animal{

  public void eat(){
    System.out.println("Mammal eats");
  }

  public void travel(){
    System.out.println("Mammal travels");
  }

  public int noOfLegs(){
    return 0;
  }
  public static void main(String args[]){
    MammalInt m = new MammalInt();
    m.eat();
    m.travel();
  }
}
```

This would produce the following result:

```
Mammal eats
Mammal travels
```

When overriding methods defined in interfaces there are several rules to be followed:

Checked exceptions should not be declared on implementation methods other than the ones declared by the interface method or subclasses of those declared by the interface method.

The signature of the interface method and the same return type or subtype should be maintained when overriding the methods.

An implementation class itself can be abstract and if so interface methods need not be implemented.

When implementation interfaces there are several rules:

(xiv) A class can implement more than one interface at a time.

(xv) A class can extend only one class, but implement many interfaces.

(xvi) An interface can extend another interface, similarly to the way that a class can extend another class.

Extending Interfaces:

An interface can extend another interface, similarly to the way that a class can extend another class. The extends keyword is used to extend an interface, and the child interface inherits the methods of the parent interface.

The following Sports interface is extended by Hockey and Football interfaces.

```
//Filename: Sports.java
public interface Sports
{
  public void setHomeTeam(String name);
  public void setVisitingTeam(String name);
  }
```

```
//Filename: Football.java
public interface Football extends Sports
{
   public void homeTeamScored(int points);
   public void visitingTeamScored(int points);
   public void endOfQuarter(int quarter);
}

//Filename: Hockey.java
public interface Hockey extends Sports
{
   public void homeGoalScored();
   public void visitingGoalScored();
   public void endOfPeriod(int period);
   public void overtimePeriod(int ot);
}
```

The Hockey interface has four methods, but it inherits two from Sports; thus, a class that implements Hockey needs to implement all six methods. Similarly, a class that implements Football needs to define the three methods from Football and the two methods from Sports.

Extending Multiple Interfaces:

A Java class can only extend one parent class. Multiple inheritance is not allowed. Interfaces are not classes, however, and an interface can extend more than one parent interface.

The extends keyword is used once, and the parent interfaces are declared in a comma-separated list.

For example, if the Hockey interface extended both Sports and Event, it would be declared as:

public interface Hockey extends Sports, Event

---

iii) Write note on final keyword.(4)

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

i) variable
ii) method
iii) class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these

Example of final variable

```
class Bike9{
final int speedlimit=90;//final variable
void run(){
```

```
speedlimit=400;
}
public static void main(String args[]){
Bike9 obj=new  Bike9();
obj.run();
}
}//end of class
```

Example of final method

```
class Bike{
final void run(){System.out.println("running");}
}

class Honda extends Bike{
void run(){System.out.println("running safely with 100kmph");}

public static void main(String args[]){
Honda honda= new Honda();
honda.run();
}
}
```

Example of final class

```
    final class Bike{}

    class Honda1 extends Bike{
      void run(){System.out.println("running safely with 100kmph");}

      public static void main(String args[]){
      Honda1 honda= new Honda();
      honda.run();
      }
}
```

4. **(i) Explain in details the concepts of inner classes.**

Java inner class or nested class is a class i.e. declared inside the class or interface.
We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.
Additionally, it can access all the members of outer class including private data members and methods.

Syntax of Inner class

```
class Java_Outer_class{
 //code
 class Java_Inner_class{
  //code
 }
}
```

Advantage of java inner classes

There are basically three advantages of inner classes in java. They are as follows:

1) Nested classes represent a special type of relationship that is **it can access all the members (data members and methods) of outer class** including private.
2) Nested classes are used **to develop more readable and maintainable code** because it logically group classes and interfaces in one place only.
3) **Code Optimization**: It requires less code to write.

Difference between nested class and inner class in Java

Inner class is a part of nested class. Non-static nested classes are known as inner classes.

Types of Nested classes

There are two types of nested classes non-static and static nested classes.The non-static nested classes are also known as inner classes.

1. Non-static nested class(inner class)
   - a)Member inner class
   - b)Annomynous inner class
   - c)Local inner class
2. Static nested class

Type	Description
Member Inner Class	A class created within class and outside method.
Anonymous Inner Class	A class created for implementing interface or extending class. Its name is decided by the java compiler.
Local Inner Class	A class created within method.
Static Nested Class	A static class created within class.
Nested Interface	An interface created within class or interface.

(ii) Explain in details the concepts of applets.

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

Advantage of Applet

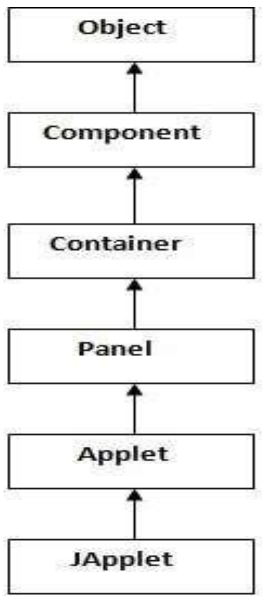
There are many advantages of applet. They are as follows:

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

Drawback of Applet

- Plugin is required at client browser to execute applet.

Hierarchy of Applet



As displayed in the above diagram, Applet class extends Panel. Panel class extends Container which is the subclass of Component.

Lifecycle of Java Applet

1. Applet is initialized.
2. Applet is started.
3. Applet is painted.
4. Applet is stopped.
5. Applet is destroyed.
6.

Lifecycle methods for Applet:

The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides 1 life cycle methods for an applet.

java.applet.Applet class

For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.

1. **public void init():** is used to initialized the Applet. It is invoked only once.
2. **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.
3. **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
4. **public void destroy():** is used to destroy the Applet. It is invoked only once.

java.awt.Component class

The Component class provides 1 life cycle method of applet.

1. **public void paint(Graphics g):** is used to paint the Applet. It provides Graphics class object that

can be used for drawing oval, rectangle, arc etc.

How to run an Applet?

There are two ways to run an applet

1. By html file.
2. By appletViewer tool (for testing purpose).

Simple example of Applet by html file:

To execute the applet by html file, create an applet and compile it. After that create an html file and place the applet code in html file. Now click the html file.

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet{
public void paint(Graphics g){
g.drawString("welcome",150,150);
}
}
```

myapplet.html

```
<html>
<body>
<applet code="First.class" width="300" height="300">
</applet>
</body>
</html>
```

Simple example of Applet by appletviewer tool:

To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. After that run it by: appletviewer First.java. Now Html file is not required but it is for testing purpose only.

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet{

public void paint(Graphics g){
g.drawString("welcome to applet",150,150);
}

}
/*
<applet code="First.class" width="300" height="300">
</applet>
*/
```

To execute the applet by appletviewer tool, write in command prompt:

```
c:\>javac First.java
```

```
c:\>appletviewer First.java
```

5. **What is Exception handling in java? Why it is used? Write a java code to simulate the way a stack mechanisms works with exception handling, throwing and dealing with exceptions such as stack is full(if you want to add more elements into the stack)or Stack is empty(you want to pop elements from the stack).**

An exception is a problem that arises during the execution of a program. An exception can occur for many different reasons, including the following:

- A user has entered invalid data.
- A file that needs to be opened cannot be found.
- A network connection has been lost in the middle of communications or the JVM has run out of memory.

Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

To understand how exception handling works in Java, you need to understand the three categories of exceptions:

- **Checked exceptions:** A checked exception is an exception that is typically a user error or a problem that cannot be foreseen by the programmer. For example, if a file is to be opened, but the file cannot be found, an exception occurs. These exceptions cannot simply be ignored at the time of compilation.
- **Runtime exceptions:** A runtime exception is an exception that occurs that probably could have been avoided by the programmer. As opposed to checked exceptions, runtime exceptions are ignored at the time of compilation.
- **Errors:** These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

```
class SimpleStackDemo {
public static void main(String arg[])
{
FixedLengthStack stack=new FixedLengthStack(5);
char ch;
int i;
  try{
// overrun the stack
for(i=0;i<6;i++)
System.out.println(“Attempting to push:”+(char) (‘A’+i));
stack.push((char) (‘A’+i));
System.out.println(“=ok”);
}
catch(StackFullException exc) {
```

	*System.out.println(exc);
	}
	System.out.println();
	try{
	//over-empty the stack
	for(i=0;i<6;i++) {
	
	System.out.println("Popping next char:");
	ch=stack.pop();
	System.out.println(ch);
	}
	}
	Catch(stackEmptyException exc) {
	System.out.println(exc);
	}
	}
	}*

6.	**Discuss the concept of synchronization in thread and develop a JAVA code for reader/writer problem.**
	
	When we start two or more threads within a program, there may be a situation when multiple threads try to access the same resource and finally they can produce unforeseen result due to concurrency issue. For example if multiple threads try to write within a same file then they may corrupt the data because one of the threads can overrite data or while one thread is opening the same file at the same time another thread might be closing the same file.
	
	So there is a need to synchronize the action of multiple threads and make sure that only one thread can access the resource at a given point in time. This is implemented using a concept called **monitors**. Each object in Java is associated with a monitor, which a thread can lock or unlock. Only one thread at a time may hold a lock on a monitor.
	
	Java programming language provides a very handy way of creating threads and synchronizing their task by using **synchronized** blocks. You keep shared resources within this block. Following is the general form of the synchronized statement:
	
	synchronized(objectidentifier) {
	  // Access shared variables and other shared resources
	}
	
	Here, the **objectidentifier** is a reference to an object whose lock associates with the monitor that the synchronized statement represents. Now we are going to see two examples where we will print a counter using two different threads. When threads are not synchronized, they print counter value which is not in sequence, but when we print counter by putting inside synchronized() block, then it prints counter very much in sequence for both the threads.

Multithreading example with Synchronization:

Here is the same example which prints counter value in sequence and every time we run it, it produces same result.

```
class PrintDemo {
   public void printCount(){
    try {
         for(int i = 5; i > 0; i--) {
            System.out.println("Counter   ---   "  + i );
         }
     } catch (Exception e) {
         System.out.println("Thread  interrupted.");
     }
   }

}

class ThreadDemo extends Thread {
   private Thread t;
   private String threadName;
   PrintDemo  PD;

   ThreadDemo( String name, PrintDemo pd){
       threadName = name;
       PD = pd;
   }
   public void run() {
     synchronized(PD) {
        PD.printCount();
     }
     System.out.println("Thread " + threadName + " exiting.");
   }

   public void start ()
   {
      System.out.println("Starting " + threadName );
      if (t == null)
      {
         t = new Thread (this, threadName);
         t.start ();
      }
   }

}
```

```
public class TestThread {
  public static void main(String args[]) {

    PrintDemo PD = new PrintDemo();

    ThreadDemo T1 = new ThreadDemo( "Thread - 1 ", PD );
    ThreadDemo T2 = new ThreadDemo( "Thread - 2 ", PD );

    T1.start();
    T2.start();

    // wait for threads to end
    try {
      T1.join();
      T2.join();
    } catch( Exception e) {
      System.out.println("Interrupted");
    }
  }
}
```

This produces same result every time you run this program:

```
Starting Thread - 1
Starting Thread - 2
Counter ---  5
Counter   ---   4
Counter   ---   3
Counter   ---   2
Counter   ---   1
Thread Thread - 1 exiting.
Counter ---  5
Counter   ---   4
Counter   ---   3
Counter   ---   2
Counter   ---   1
Thread Thread - 2 exiting.
```

```
/**
 * Reader.java
 *
 * A reader to the database.
 *
 */
```

```
   class Reader implements Runnable
  {

     private RWLock database;
     private int readerNum;

      public Reader(int readerNum, RWLock database) {
        this.readerNum = readerNum;
        this.database = database;
     }

      public void run() {
        while (true) {
           SleepUtilities.nap();

           System.out.println("reader " + readerNum + " wants to read.");
           database.acquireReadLock(readerNum);

        // you have access to read from the database
        // let's read for awhile .....
           SleepUtilities.nap();

           database.releaseReadLock(readerNum);
        }
     }
  ;
  }

//****************************************************************

/**
 * Writer.java
 *
 * A writer to the database.
 *
 */
   class Writer implements Runnable
  {
     private RWLock database;
     private int writerNum;

      public Writer(int w, RWLock d) {
        writerNum = w;
        database = d;
     }
```

```
    public void run() {
      while (true){
        SleepUtilities.nap();

        System.out.println("writer " + writerNum + " wants to write.");
        database.acquireWriteLock(writerNum);

      // you have access to write to the database
      // write for awhile ...
        SleepUtilities.nap();

        database.releaseWriteLock(writerNum);
      }
    }
  }
```

7. **(i) Describe the concept of I/O with example.(8)**

Java I/O (Input and Output) is used to process the input and produce the output based on the input.
Java uses the concept of stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.
We can perform **file handling in java** by java IO API.

Stream

A stream is a sequence of data.In Java a stream is composed of bytes. It's called a stream because it's like a stream of water that continues to flow.
In java, 3 streams are created for us automatically. All these streams are attached with console.
1) System.out: standard output stream
2) System.in: standard input stream
3) System.err: standard error stream

Let's see the code to print **output and error** message to the console.

1. System.out.println("simple message");
2. System.err.println("error message");

Let's see the code to get **input** from console.

1. int i=System.in.read();//returns ASCII code of 1st character
2. System.out.println((char)i);//will print the character
3.

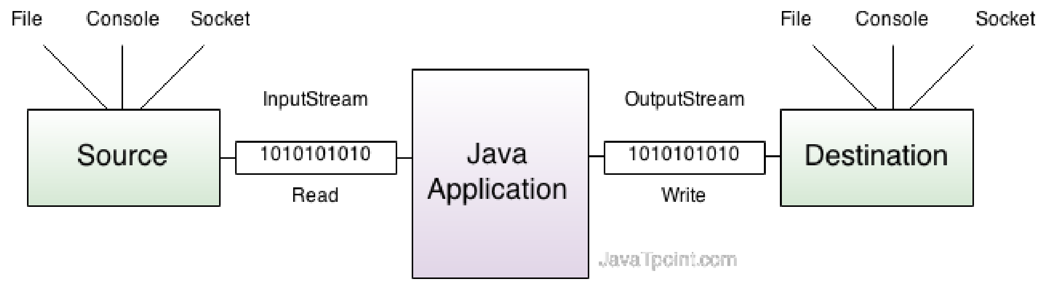
OutputStream

Java application uses an output stream to write data to a destination, it may be a file,an array,peripheral device or socket.

InputStream

Java application uses an input stream to read data from a source, it may be a file,an array,peripheral device or socket.

Let's understand working of Java OutputStream and InputStream by the figure given below.

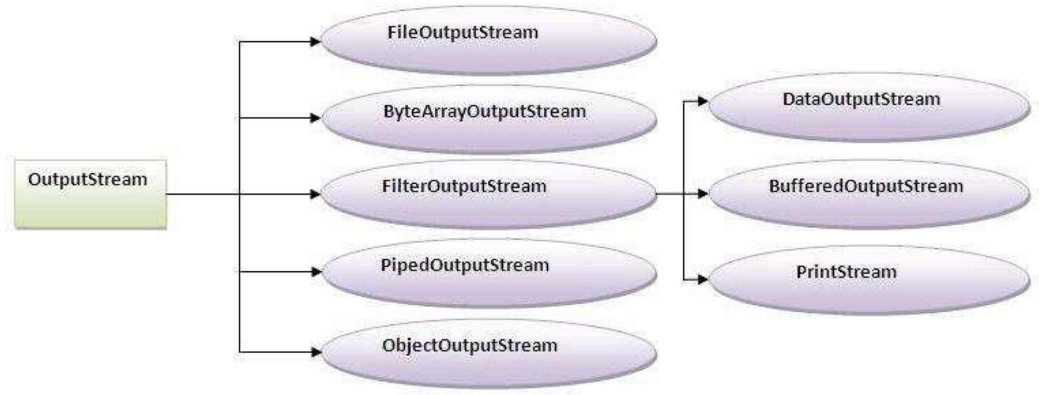


OutputStream class

OutputStream class is an abstract class.It is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Commonly used methods of OutputStream class

Method	Description
1) public void write(int)throws IOException:	is used to write a byte to the current output stream.
2) public void write(byte[])throws IOException:	is used to write an array of byte to the current output stream.
3) public void flush()throws IOException:	flushes the current output stream.
4) public void close()throws IOException:	is used to close the current output stream.

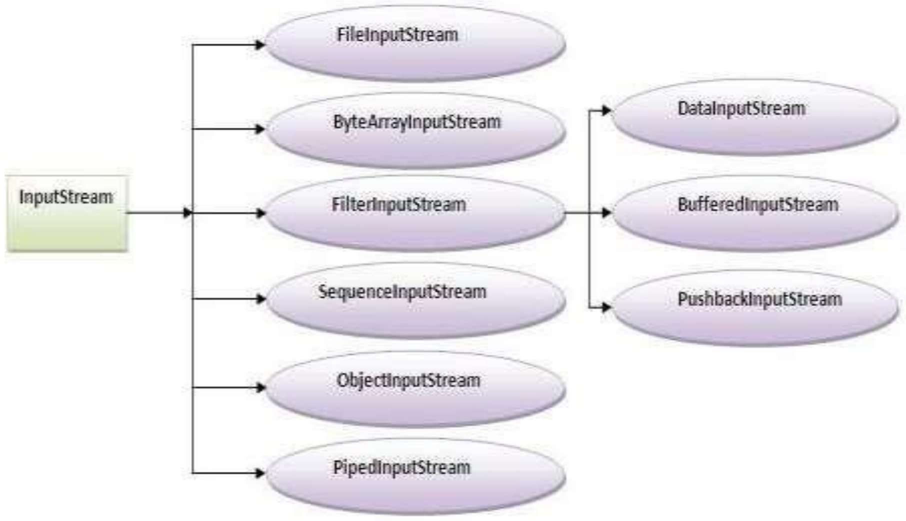


InputStream class

InputStream class is an abstract class.It is the superclass of all classes representing an input stream of bytes.

Commonly used methods of InputStream class

Method	Description
1) public abstract int read()throws IOException:	reads the next byte of data from the input stream.It returns -1 at the end of file.
2) public int available()throws IOException:	returns an estimate of the number of bytes that can be read from the current input stream.
3) public void close()throws IOException:	is used to close the current input stream.



(ii) Explain the detail about string handling.(8).

Java String provides a lot of concepts that can be performed on a string such as compare, concat, equals, split, length, replace, compareTo, intern, substring etc.

In java, string is basically an object that represents sequence of char values.

An array of characters works same as java string. For example:

```
1. char[] ch={'j','a','v','a','t','p','o','i','n','t'};
2. String s=new String(ch);
```

is same as:

```
1. String s="javatpoint";
```

The java.lang.String class implements *Serializable*, *Comparable* and *CharSequence* interfaces.

The java String is immutable i.e. it cannot be changed but a new instance is created. For mutable class, you can use StringBuffer and StringBuilder class.

Generally, string is a sequence of characters. But in java, string is an object that represents a sequence of characters. String class is used to create string object.
There are two ways to create String object:

1. By string literal
2. By new keyword

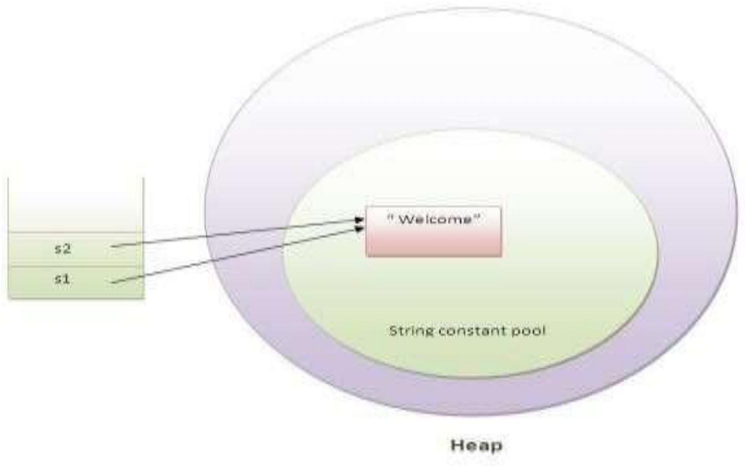
1) String Literal

Java String literal is created by using double quotes. For Example:

1. String s="welcome";

Each time you create a string literal, the JVM checks the string constant pool first. If the string already exists in the pool, a reference to the pooled instance is returned. If string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

1. String s1="Welcome";
2. String s2="Welcome";//will not create new instance



2) By new keyword

1. String s=new String("Welcome");//creates two objects and one reference variable

Java String Example

```
public class StringExample{
public static void main(String args[]){
String s1="java";//creating string by java string literal

char ch[]={'s','t','r','i','n','g','s'};
String s2=new String(ch);//converting char array to string

String s3=new String("example");//creating java string by new keyword

System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
}}
```

Output:

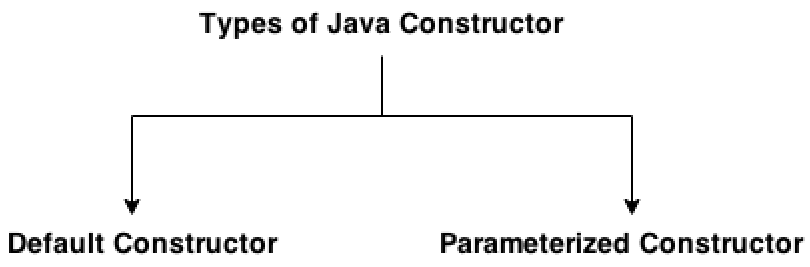
```
java
strings
example
```

8. **What is meant by constructors? Describe the type of constructors supported by java with example.**

Constructor in java is a *special type of method* that is used to initialize the object.
Java constructor is *invoked at the time of object creation*. It constructs the values i.e. provides data for the object that is why it is known as constructor.

Rules for creating java constructor
There are basically two rules defined for the constructor.
1. Constructor name must be same as its class name
2. Constructor must have no explicit return type

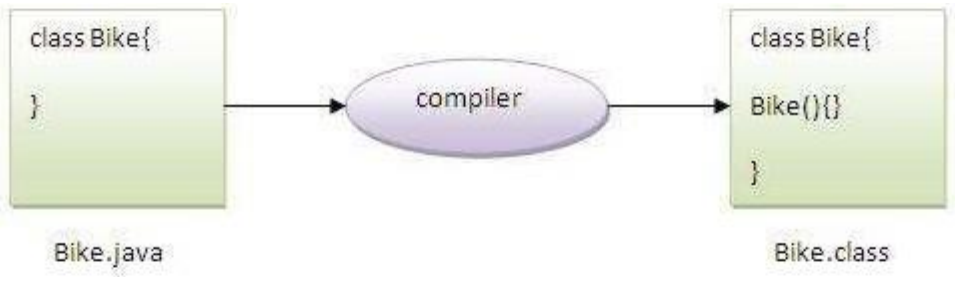
Types of java constructors
There are two types of constructors:
1. Default constructor (no-arg constructor)
2. Parameterized constructor



Java Default Constructor
A constructor that have no parameter is known as default constructor.
Syntax of default constructor:
<class_name>(){}
Example of default constructor
In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
class Bike1{
Bike1(){System.out.println("Bike is created");}
public static void main(String args[]){
Bike1 b=new Bike1();
}
}
```

Output:
Bike is created



Java parameterized constructor

A constructor that have parameters is known as parameterized constructor.
Parameterized constructor is used to provide different values to the distinct objects.
In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
class Student4{
   int id;
   String name;
   Student4(int i,String n){
   id = i;
   name = n;
   }
   void display(){System.out.println(id+" "+name);}
   public static void main(String args[]){
   Student4 s1 = new Student4(111,"Karan");
   Student4 s2 = new Student4(222,"Aryan");
   s1.display();
   s2.display();
  }
}
```

Output:

```
111 Karan
222 Aryan
```

Constructor Overloading in Java

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists.The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

```
class Student5{
int id;
String name;
int age;
Student5(int i,String n){
id = i;
name = n;
}
Student5(int i,String n,int a){
id = i;
name = n;
age=a;
}
void display(){System.out.println(id+" "+name+" "+age);}

public static void main(String args[]){
Student5 s1 = new Student5(111,"Karan");
Student5 s2 = new Student5(222,"Aryan",25);
s1.display();
s2.display();
}
}
```

Output:

```
111 Karan 0
222 Aryan 25
```

Java Copy Constructor

There is no copy constructor in java. But, we can copy the values of one object to another like copy constructor in C++.

There are many ways to copy the values of one object into another in java. They are:

By constructor
By assigning the values of one object into another
By clone() method of Object class

In this example, we are going to copy the values of one object into another using java constructor.

```
class Student6{
    int id;
    String name;
    Student6(int i,String n){
    id = i;
    name = n;
    }

    Student6(Student6 s){
    id = s.id;
    name =s.name;
    }
    void display(){System.out.println(id+" "+name);}
    public static void main(String args[]){
    Student6 s1 = new Student6(111,"Karan");
    Student6 s2 = new Student6(s1);
    s1.display();
    s2.display();
   }
 }
```

Test it Now

Output:

```
111 Karan
111 Karan
```

9. Define inheritances. Explain in details types of inheritances supported by JAVA with example program.

Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object.

The idea behind inheritance in java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of parent class, and you can add new methods and fields also.

Inheritance represents the **IS-A relationship**, also known as *parent-child* relationship.

use inheritance in java

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

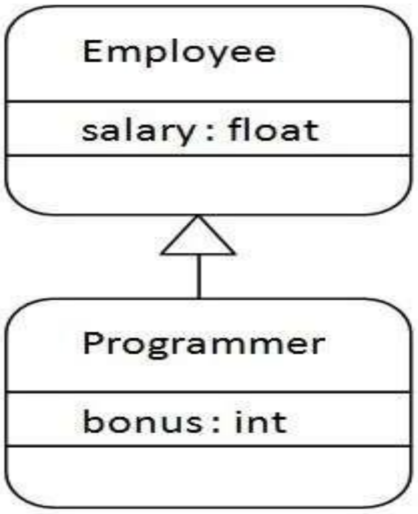
Syntax of Java Inheritance

```
class Subclass-name extends Superclass-name
{
//methods and fields
}
```

The **extends keyword** indicates that you are making a new class that derives from an existing class.

In the terminology of Java, a class that is inherited is called a super class. The new class is called a subclass.

Understanding the simple example of inheritance



As displayed in the above figure, Programmer is the subclass and Employee is the superclass. Relationship between two classes is **Programmer IS-A Employee**.It means that Programmer is a type of Employee.

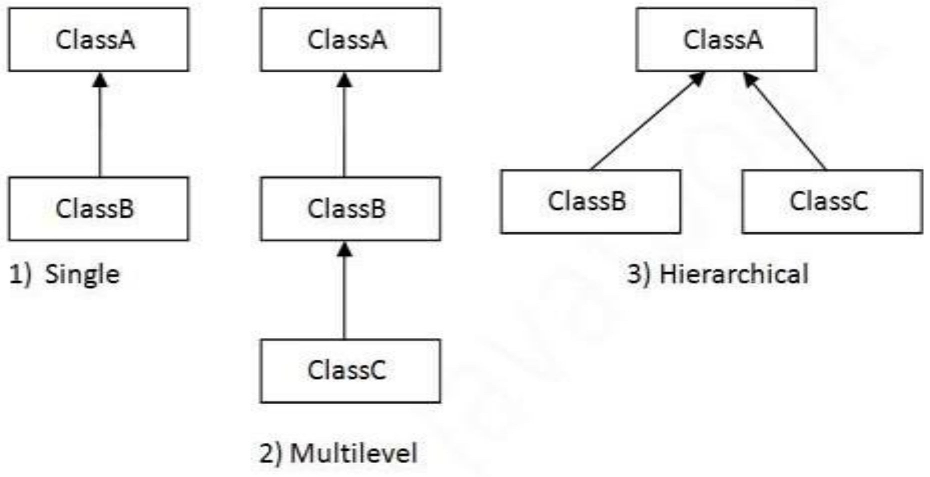
```
class Employee{
float salary=40000;
}
class Programmer extends Employee{
int bonus=10000;
public static void main(String args[]){
Programmer p=new Programmer();
System.out.println("Programmer salary is:"+p.salary);
System.out.println("Bonus of Programmer is:"+p.bonus);
}
}
```

```
Programmer salary is:40000.0
 Bonus of programmer is:10000
```

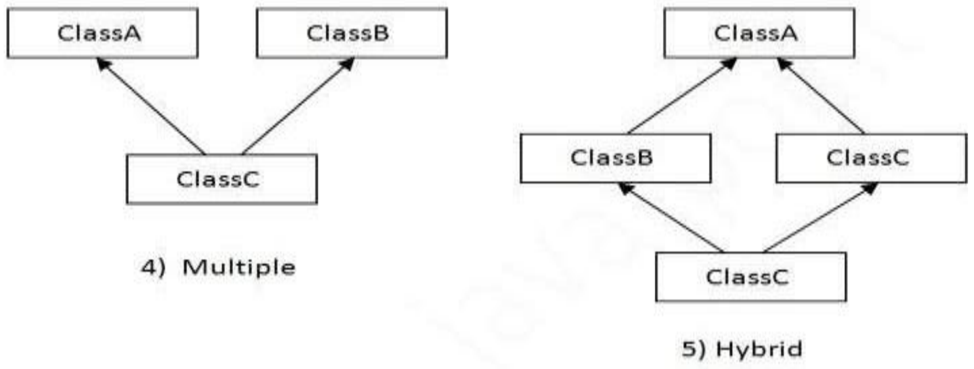
Types of inheritance in java

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.



When a class extends multiple classes i.e. known as multiple inheritance. For Example:



.

Why multiple inheritance is not supported in java?

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B and C are three classes. The C class inherits A and B classes. If A and B classes have same method and you call it from child class object, there will be ambiguity to call method of A or B class.

Since compile time errors are better than runtime errors, java renders compile time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error now.

```
class A{
void msg(){System.out.println("Hello");}
}
class B{
void msg(){System.out.println("Welcome");}
}
```

```
    class C extends A,B{//suppose if it were

     Public Static void main(String args[]){
       C obj=new C();
       obj.msg();//Now which msg() method would be invoked?
     }
     }
Compile Time Error
```

10. **Explain in details stream classes with suitable example.**

The java.io package contains nearly every class you might ever need to perform input and output (I/O) in Java. All these streams represent an input source and an output destination. The stream in the java.io package supports many data such as primitives, Object, localized characters, etc.

A stream can be defined as a sequence of data. The InputStream is used to read data from a source and the OutputStream is used for writing data to a destination.

Java provides strong but flexible support for I/O related to Files and networks but this tutorial covers very basic functionality related to streams and I/O. We would see most commonly used example one by one:

Byte Streams

Java byte streams are used to perform input and output of 8-bit bytes. Though there are many classes related to byte streams but the most frequently used classes are , **FileInputStream** and **FileOutputStream**. Following is an example which makes use of these two classes to copy an input file into an output file:

```
import  java.io.*;

public class CopyFile {
   public static void main(String args[]) throws IOException
   {
      FileInputStream in = null;
      FileOutputStream out = null;

      try {
         in = new FileInputStream("input.txt");
         out = new FileOutputStream("output.txt");

         int c;
         while ((c = in.read()) != -1) {
            out.write(c);
         }
      }finally {
         if (in != null) {
            in.close();
```

```
        }
        if (out != null) {
           out.close();
        }
      }
   }
}
```

Now let's have a file **input.txt** with the following content:

This is test for copy file.

As a next step, compile above program and execute it, which will result in creating output.txt file with the same content as we have in input.txt. So let's put above code in CopyFile.java file and do the following:

```
$javac CopyFile.java
$java CopyFile
```

Character Streams

Java **Byte** streams are used to perform input and output of 8-bit bytes, where as Java **Character** streams are used to perform input and output for 16-bit unicode. Though there are many classes related to character streams but the most frequently used classes are , **FileReader** and **FileWriter.**. Though internally FileReader uses FileInputStream and FileWriter uses FileOutputStream but here major difference is that FileReader reads two bytes at a time and FileWriter writes two bytes at a time.

We can re-write above example which makes use of these two classes to copy an input file (having unicode characters) into an output file:

```
import  java.io.*;

public class CopyFile {
   public static void main(String args[]) throws IOException
   {
      FileReader in = null;
      FileWriter out = null;

      try {
         in = new FileReader("input.txt");
         out = new FileWriter("output.txt");

         int c;
         while ((c = in.read()) != -1) {
            out.write(c);
         }
      }finally {
        if (in != null) {
          in.close();
        }
        if (out != null) {
          out.close();
        }
      }
    }
}
```

Now let's have a file **input.txt** with the following content:

This is test for copy file.

As a next step, compile above program and execute it, which will result in creating output.txt file with the same content as we have in input.txt. So let's put above code in CopyFile.java file and do the following:

```
$javac CopyFile.java
$java CopyFile
```

Standard Streams

All the programming languages provide support for standard I/O where user's program can take input from a keyboard and then produce output on the computer screen. If you are aware if C or C++ programming languages, then you must be aware of three standard devices STDIN, STDOUT and STDERR. Similar way Java provides following three standard streams

- **Standard Input:** This is used to feed the data to user's program and usually a keyboard is used as standard input stream and represented as **System.in**.
- **Standard Output:** This is used to output the data produced by the user's program and usually a computer screen is used to standard output stream and represented as **System.out**.
- **Standard Error:** This is used to output the error data produced by the user's program and usually a computer screen is used to standard error stream and represented as **System.err**.

Following is a simple program which creates **InputStreamReader** to read standard input stream until the user types a "q":

```
import java.io.*;

public class ReadConsole {
  public static void main(String args[]) throws IOException
  {
    InputStreamReader cin = null;

    try {
      cin = new InputStreamReader(System.in);
```

	```
	      System.out.println("Enter characters, 'q' to quit.");
	      char c;
	      do {
	        c = (char) cin.read();
	        System.out.print(c);
	      } while(c != 'q');
	    }finally {
	      if (cin != null) {
	        cin.close();
	      }
	    }
	  }
	}
	```
	Let's keep above code in ReadConsole.java file and try to compile and execute it as below. This program continues reading and outputting same character until we press 'q':
	```
	$javac ReadConsole.java
	$java ReadConsole
	Enter characters, 'q' to quit.
	1
	1
	e
	e
	q
	q
	```

	Unit-II **Part-A**
1.	**What is web 2.0?** A Web 2.0 site may allow users to interact and collaborate with each other in a social media dialogue as creators of user-generated content in a virtual community, in contrast to Web sites where people are limited to the passive viewing of content. Examples of Web 2.0 include social networking sites, blogs, wikis, folksonomies, video sharing sites, hosted services, Web applications, and mashups.
2.	**Define RIA.** A rich Internet application (RIA) is a Web application designed to deliver the same features and functions normally associated with deskop applications. RIAs generally split the processing across the Internet/network divide by locating the user interface and related activity and capability on the client side, and the data manipulation and operation on the application server side.
3.	**Define collaboration.** Collaboration is a process defined by the recursive interaction of knowledge and mutual learning between two or more people who are working together, in an intellectual endeavour, toward a common goal which is typically creative in nature.
4.	**List the Collaborations tools.** AnswerGaeden,Thinkature,DotVoting,ePals,Gaggle,Glass,Tricider.
5.	**What are the collaborative processes.** • Team Creation

	• Idea Generation • Decision-Making • Work or Production • Evaluation or Recap
6.	**Define Web services.** A *Web service* is a method of communication between two electronic devices over a network. It is a software function provided at a network address over the Web with the service *always on* as in the concept of utility computing.
7.	**Write short notes on Software as service(Soas).** **SOAs : Software as a service** (**SaaS**), sometimes referred to as "software on demand," is software that is deployed over the internet and/or is deployed to run behind a firewall on a local area network or personal computer. With SaaS, a provider licenses an application to customers either as a service on demand, through a subscription, in a "pay-as-you-go" model, or (increasingly) at no charge.
8.	**Write short notes on Social networking.** A **social network** is a social structure made up of a set of social actors (such as individuals or organizations) and a set of the dyadic ties between these actors. The social network perspective provides a set of methods for analyzing the structure of whole social entities as well as a variety of theories explaining the patterns observed in these structures.
9.	**Define Website.** A website is hosted on at least one web server, accessible via a network such as the Internet or a private local area network through an Internet address known as a uniform resource locator (URL). All publicly accessible websites collectively constitute the World Wide Web
10.	**Differences between web sites and web server.** **Website:** A website is a set of linked documents associated with a particular person, organization or topic that is held on a computer system and can be accessed as part of the world wide web. (Not to be confused with: Web page, a document on the world wide web written in HTML and displayed in a web browser.) **Web server:** The web server on the other side is a computer program, which delivers content, such as websites or web pages, for example, over the world wide web from a web server to your computer.
11.	**Define internet.** The Internet is a global system of interconnected computer networks that use the standard Internet protocol suite (TCP/IP) to link several billion devices worldwide. It is a network of networks that consists of millions of private, public, academic, business, and government networks of local to global scope, linked by a broad array of electronic, wireless, and optical networking technologies.
12.	**Define intranet.** An intranet is a computer network that uses Internet Protocol technology to share information, operational systems, or computing services within an organization. This term is used in contrast to extranet, a network between organizations, and instead refers to a network within an organization. Sometimes, the term refers only to the organization's internal website, but may be a more extensive part of the organization's information technology infrastructure, and may be composed of multiple local area networks. The objective is to organize each individual's desktop with minimal cost, time and effort to be more productive, cost efficient, timely, and competitive.
13.	**Differentiate between internet and intranet.** • Internet is general to PCs all over the world whereas Intranet is specific to few PCs. • Internet has wider access and provides a better access to websites to large population whereas Intranet is restricted. • Internet is not as safe as Intranet as Intranet can be safely privatized as per the need.
14.	**Define HTML.** HTML is a simple web page description language, which enables document creation for the web. HTML is the set

of mark-up symbols or codes placed in a file intended for display on the web browser page. These mark-up symbol and codes identify structural elements such as paragraphs, heading, and lists. HTML can be used to place media (such as graphics, video, and audio) on the Web page and describe fill-in-forms. A method is an implementation of an objects behavior.

15. **Explain about HTTP Connection**.
It is a communication channel between web browser and web server. It begins on the
client side with the browser sending a request to the web server for a document.
Request Header Fields are
1. From
2. Reference
3. If_modified_since
4. Pragma
5. User Agent

16. **Define cascading.**
Cascading refers to a certain set of rules that browsers use, in cascading order, to determine how to use the style information. Such a set of rules is useful in the event of conflicting style information because the rules would give the browser a way to determine which style is given precedence.

17. **State the use of web server logs and list the contents of a message log. (APR/MAY 2011)**
A server log is a log file (or several files) automatically created and maintained by a server of activity performed by it. A typical example is a web server log which maintains a history of page requests. The W3C maintains a standard format (the Common Log Format) for web server log files, but other proprietary formats exist.
The message log is used by a number of processes to provide debugging and troubleshooting information. You can view the message log from the process monitor after clicking on the details hyperlink for a process and the by clicking on the message log hyperlink in the actions area.

18. **How will you create a password field in a HTML form? (NOV/DEC 2011)**
<input type=”password” name=”pwd” size=”15”>

19. **List any four common browsers. (NOV/DEC 2011)**
- Google Chrome
- Netscape Navigator
- Microsoft Internet Explorer
- Mozilla

20. **State the uses of internet protocol. (APR/MAY 2012)**
- IP function: transfer data from source device to destination device
- IP source software creates a packet representing the data
- Header: source and destination IP addresses, length of data, etc.
- Data: Data itself

21. **Define Tags. What are the two different types of tags?**
Tags signal the browser to inform about the formatting details.ie how the content shouls be displayed in the browser screen. Tags are enclosed between “<” and”>”
Standalone tag only start tag is present and no end tag. Example <BR> and container tag have start and end tag will be present .Example <html>…. </html>

22. **What are the rules to define a tag?**
Attributes should be placed inside start tag, appears as Name-value pairs separted by blank spaces, Attributes should have only one value,values should be enclosed within either single(‘) or double (“) quotes.

23. **Differentiate between standalone and container tag.**

S.no	**Standalone**	**Container**
1	Only start tag is present and no end tag.	Both start and end tag will be present
2	Can have only attributes and no parameters	Can have both attributes and parameters.

	3 | Example:<BR> | Example:<html>…..</html>
24.	**What is the use of <pre> tag in HTML?** The pre tag can be used to preserve the white spaces and lines in the text.
25.	**What is cellpadding and cell spacing attributes?** The cellpadding allows to have some space between the contents of each cell and its borders. The distance between each cell is called cell spacing.
26.	**What is the need of using form in HTML?** Form is a typical layout on the web page by which user can interact with the web page. The components that can be placed on the form are text box, check box, radio buttons, and push buttons and so on. Thus form is typically used to create an interactive Graphical User Interface.
27.	**What is the purpose of using frames in HTML?** The HTML frames allows the web designer to present the web document in multiple views. Using multiple views one can keep the formation visible and at the same time other views can be scrolled or replaced.
28.	**What is the need for special character in HTML?** There are some symbols that cannot be used directly in HTML document. For example <(less than) because this symbol is also used along with the tag. Hence this is called a special symbol and can be denoted with the help of entity reference.
29.	**State how an unrecognized element or attribute treated by the HTML document?** If any unrecocognized element or attribute is written then the HTML document simply displays the contents. For example <title>testing</title> will display the string "testing" on the web page. It will not display it as a title of the web page.
30.	**What is the use of hyperlink tag in HTML?** The hyperlink tag is used to link logically with other page. Using this tag a web link can be specified. The <a> tag is used to specify the hyperlink in HTML.
31.	**What are the uses of hyperlink in HTML?** To logically link one page with another, use of link to enhance readability of the web document, the navigation from one page to another is possible.
32.	**What is BODY in HTML document?** The effects which we want in the window are mentioned with the help of tags in the body. It is the place where the actual data is written in html. All the changes can be viewed by changing the tags content in the body whereas the head part is the introduction part and the body is the actual content part.<BODY>data content</BODY>
33.	**What is an image map?** An image map allows you to link to several web pages through one image. Simply define shapes within images and link these to the pages you want. Here's a video to help you learn more about images and links in HTML.
34.	**What are style sheets?** The style sheets are the collection of styles that can be either embedded within the HTML documents or can be externally applied. The Cascading style sheet is a markup language used to apply the styles to HTML elements.
35.	**What is selector string? Specify any three forms of selectors.** The rule set in CSS consists of selector string which is basically an HTML element. These selectors can be defined with the help of properties and values.
36.	**What is the use of Universal Selector?** Using the universal selector the values can be defined for all the elements in the document. It is denoted by *.
37.	**What is generic class selector?** The generic class applied to any tag in the HTML document. And thus the values defined within that generic selector can be applied to the corresponding tag. The class selector must be preceded by the dot operator.
38.	**What are the advantages of External style sheet?** When we use external style sheet then the style is defined in one file and actual contents of the web are defined in another file. Hence if we want to change the style of presentation of web page then we can simply modify the file in which the style is defined.

39.	**What is the difference the external style sheet and embedded style sheet?** The external style sheet is a kind of style sheet in which the styles are defined in a separate.css file and this file is mentioned at the beginning of the HTML document. When we need to apply the particular style to more than one web documents then the external style sheet is used. The embedded style sheet is a method in which the style is specified within the HTML document itself. It is not defined in separate file. Due to embedded style sheet unique style can be applied to all the elements.
40.	**What do you mean by the term inline element?** The inline elements are those elements that do not form new blocks of content. The content is distributed in lines.
41.	**What are the various style sheets?** Inline, external, imported and embedded are the different types of style sheets.
42.	**Explain inline, embedded and external style sheets.** **Inline** If only a small piece of code has to be styled then inline style sheets can be used. **Embedded** Embedded style sheets are put between the <head> </head> tags. **External** If you want to apply a style to all the pages within your website by changing just one ***style sheet, then external style sheets can be used.***
43.	**Give example for inline style sheet. (APR/MAY 2013)** <h2>InLINE CSS</h2> <p style=”color:sienna;margin-left:20px”> The style ATTRIBUTE we are able to modify the appearance of HTML elements </p>

44.	**How will you embed the external style sheet? (May 2014)**
	In external style sheet is ideal when the style is applied to many pages. With an external style sheet, you can change the look of an entire Web site by changing just one file.Each page must include a link to the style sheet with the <link> tag. The <link> tag goes inside the head section:
	<head>
	<link rel="stylesheet" type="text/css" href="mystyle.css">
	</head>
	An external style sheet can be written in any text editor. The file should not contain any html tags. The style sheet file must be saved with a .css extension. An example of a style sheet file is shown below:
	"mýStyle.css":
	body {
	  background-color: lightblue;}
	h1 {
	  color: navy;
	  margin-left: 20px;}

45.	**How will you include CSS in a web site? (MAY/JUNE 2014)** **Inline** Inline styles are when you just insert the type of style you want inside another tag, using the style attribute. This is usually the least useful way to use CSS. *<p style="width:100%; color:#660099; text-align:right; background-color:#ffcc00;" >* **Embedded** Styles can also be placed in the document using the <style> tag. The <style> tag is usually placed in the head section of the document, where it will apply to the whole document. *<style> <!--* *p { color:#009900;* *font-family:"comic sans ms",sans-serif; }* *h1 { color:#660000; font-size:12pt; }* *</style>*

	External styles Styles can also be set in an external style sheet which is linked to the page with a <link> tag. For example the style sheet for this site is included like this: *<link rel="stylesheet" type="text/css" href="class.css" />*
46.	**What is the purpose of CSS Box Model and mention its parts also.** The CSS box model is essentially a box that wraps around HTML elements, and it consists of: margins, borders, padding, and the actual content. The different parts are: • Margin • Border • Padding • Content
	Part-B
1.	**Explain WWW and HTTP Protocol.** **WWW** The term WWW refers to the World Wide Web or simply the Web. The World Wide Web consists of all the public Web sites connected to the Internet worldwide, including the client devices (such as computers and cell phones) that access Web content. The WWW is just one of many applications of the Internet and computer networks. The World Web is based on these technologies: 1. HTML - Hypertext Markup Language 2. HTTP - Hypertext Transfer Protocol 3. Web servers and Web browsers **HTTP** HTTP (Hypertext Transfer Protocol) is the set of rules for transferring files (text, graphic images, sound, video, and other multimedia files) on the World Wide Web. HTTP is an application protocol that runs on top of the TCP/IP suite of protocols. HTTP concepts include (as the Hypertext part of the name implies) the idea that files can contain references to other files whose selection will elicit additional transfer requests. Any Web server machine contains, in addition to the Web page files it can serve, an HTTP daemon, a program that is designed to wait for HTTP requests and handle them when they arrive. Your Web browser is an HTTP client, sending requests to server machines. When the browser user enters file requests by either "opening" a Web file (typing in a Uniform Resource Locator or URL) or clicking on a hypertext link, the browser builds an HTTP request and sends it to the Internet Protocol address (IP address) indicated by the URL. The HTTP daemon in the destination server machine receives the request and sends back the requested file or files associated with the request. (A Web page often consists of more than one file.)
2.	**Discuss the structure of the HTTP request message. (NOV/DEC 2012)** **Structure of the request:** start line header field(s) blank line optional body 1. **Start line** Example: GET / HTTP/1.1 Three space-separated parts: HTTP request method Request-URI HTTP version *Request URI* Uniform Resource Identifier (URI) Syntax: *scheme* : *scheme-depend-part*

Ex: In http://www.example.com/ the scheme is http
Request-URI is the portion of the requested URI that follows the host name (which is supplied by the required Host header field)

Ex: / is Request-URI portion of http://www.example.com/

Request methods:

GET
- Used if link is clicked or address typed in browser
- No body in request with GET method

POST
- Used when submit button is clicked on a form
- Form information contained in body of request

HEAD
- Requests that only header fields (no body) be returned in the response
- PUT
- DELETE
- TRACE
- OPTIONS

2. Header field structure:

– *field name* : *field value*

Syntax
- – Field name is not case sensitive
- – Field value may continue on multiple lines by starting continuation lines with white space
- – Field values may contain MIME types, quality values, and wildcard characters (*'s)

3. MIME
- – Convention for specifying content type of a message
- – In HTTP, typically used to specify content type of the body of the response
- – MIME content type syntax:
- – *top-level type* / *subtype*
- – Examples: text/html, image/jpeg
- – Example header field with quality values:

```
accept:
  text/xml,text/html;q=0.9,
  text/plain;q=0.8, image/jpeg,
  image/gif;q=0.2,*/*;q=0.1
```

- – Quality value applies to all preceding items
- – Higher the value, higher the preference
- – Note use of wildcards to specify quality 0.1 for any MIME type not specified earlier

4. Common header fields:
- – Host: host name from URL (required)
- – User-Agent: type of browser sending request
- – Accept: MIME types of acceptable documents
- – Connection: value close tells server to close connection after single request/response
- – Content-Type: MIME type of (POST) body, normally application/x-www-form-urlencoded
- – Content-Length: bytes in body

Referer: URL of document containing link that supplied URI for this HTTP request

3. **Discuss the structure of the HTTP response message.[8] (NOV/DEC 2012)**

- *Structure of the response*

status line
header field(s)
blank line

optional body

- *Status line*
  - Example: HTTP/1.1 200 OK
  - Three space-separated parts:

HTTP version
status code
reason phrase (intended for human use)

- *Status code*
  - Three-digit number
  - First digit is class of the status code:

1	Informational
2	Success
3	Redirection (alternate URL is supplied)
4	Client Error
5	Server Error

  - Other two digits provide additional information
- *Header Fields*
  - Connection, Content-Type, Content-Length
  - Date: date and time at which response was generated (required)
  - Location: alternate URI if status is redirection
  - Last-Modified: date and time the requested resource was last modified on the server
  - Expires: date and time after which the client's copy of the resource will be out-of-date
  - ETag: a unique identifier for this version of the requested resource (changes if resource changes)
- *Client Caching*
  - A cache is a local copy of information obtained from some other source
  - Most web browsers use cache to store requested resources so that subsequent requests to the same resource will not necessarily require an HTTP request/response
- Ex: icon appearing multiple times in a Web page
- *Cache advantages*
  - (Much) faster than HTTP request/response
  - Less network traffic
  - Less load on server
- *Cache disadvantage*
  - Cached copy of resource may be invalid (inconsistent with remote version)
- *Validating cached resource:*
  - Send HTTP HEAD request and check Last-Modified or ETag header in response
  - Compare current date/time with Expires header sent in response containing resource
  - If no Expires header was sent, use heuristic algorithm to estimate value for Expires
  - Ex: Expires = 0.01 * (Date – Last-Modified) + Date
- *Character sets*
  - Every document is represented by a string of integer values (code points)
  - The mapping from code points to characters is defined by a character set
  - Some header fields have character set values:
  - Accept-Charset: request header listing character sets that the client can recognize
- Ex: accept-charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
  - Content-Type: can include character set used to represent the body of the HTTP message
- Ex: Content-Type: text/html; charset=UTF-8
  - Technically, many "character sets" are actually character encodings

- An encoding represents code points using variable-length byte strings
- Most common examples are Unicode-based encodings UTF-8 and UTF-16

IANA maintains complete list of Internet-recognized character sets/encodings

4. **Explain HTML elements in detail also State the types of lists supported by HTML and explain them in detail. (APR/MAY 2011)**

<u>HTML element</u>

An HTML element is an individual component of an HTML document or web page, once this has been parsed into the Document Object Model. HTML is composed of a tree of HTML elements and other nodes, such as text nodes. Each element can have HTML attributes specified. Elements can also have content, including other elements and text. HTML elements represent semantics, or meaning. For example, the title element represents the title of the document.

Heading Tags

Any document starts with a heading. You can use different sizes for your headings. HTML also has six levels of headings, which use the elements <h1>, <h2>, <h3>, <h4>, <h5>, and <h6>. While displaying any heading, browser adds one line before and one line after that heading.

Example

```
<h1>This is heading 1</h1>
<h2>This is heading 2</h2>
<h3>This is heading 3</h3>
<h4>This is heading 4</h4>
<h5>This is heading 5</h5>
<h6>This is heading 6</h6>
```

Paragraph Tag

The <p> tag offers a way to structure your text into different paragraphs. Each paragraph of text should go in between an opening <p> and a closing </p> tag as shown below in the example:

Example

```
<p>Here is a first paragraph of text.</p>
```

Line Break Tag

Whenever you use the <br /> element, anything following it starts from the next line. This tag is an example of an empty element, where you do not need opening and closing tags, as there is nothing to go in between them.

Example

```
<p>Hello<br />
You delivered your assignment ontime.<br />
    Thanks<br />
```

Centering Content

You can use <center> tag to put any content in the center of the page or any table cell.

Example

```
<center>
<p>This text is in the center.</p>
</center>
```

Horizontal Lines

Horizontal lines are used to visually break up sections of a document. The <hr> tag creates a line from the current position in the document to the right margin and breaks the line accordingly.

Example

```
<hr />
```

Nonbreaking Spaces
you should use a nonbreaking space entity &nbsp; instead of a normal space. For example, when coding the "12 Angry Men" in a paragraph, you should use something similar to the following code:
Example

```
<p>An example of this technique appears in the movie "12&nbsp;Angry&nbsp;Men."</p>
```

<u>HTML Lists</u>
HTML offers authors several mechanisms for specifying lists of information. All lists must contain one or more list elements.
<u>Unordered HTML List</u>

- The first item
- The second item
- The third item
- The fourth item
- Ordered HTML List

  Example

```
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

<u>Ordered information.</u>

1. The first item
2. The second item
3. The third item
4. The fourth item
5. HTML Description List

   Example

```
<ol>
  <li>Coffee</li>
  <li>Milk</li>
</ol>
```

Definitions.

- The first item
- Description of item
- The second item
- Description of item

  Example

```
<dl>
  <dt>Coffee</dt>
  <dd>- black hot drink</dd>
  <dt>Milk</dt>
  <dd>- white cold drink</dd>
</dl>
```

5. Discuss the various features available in HTML to format the text with example.
Basic functionality

- Server calls on TCP software and waits for connection req to one or more ports
- When a connection request is received , the server dedicates a subtask(Single copy of server software handling a single client connection)

  - Subtask establish connection and receives request
  - Subtask examines the host header field to determine the host and invokes software for this host
  - Virtual host software Map Request-URI to specific resource on the server.
  - It maps Request-URI to specific resource associated with the virtual host
- File: Return file in HTTP response (MIME Type)
- Program: Run program and return output in HTTP response
- Log information about the request and response such as IP address and the status code in a plain-text file.
- If the TCP connection is kept alive , the server subtask continues to monitor the connection, the client send another request or initiates a connection close.

Few Definitions

- All modern servers concurrently process multiple requests
- Multiple copies of the server running simultaneously(Concurrency)
- Subtask→ Single copy of server software handling a single client connection
- Virtual Host→ HTTP request include a host header field
- Multiple host names mapped by a DNS to a single IP address
- Web server determine which virtual host is being requested by examining the host header field.

Server Configuration and Tuning

- Modern servers have large number of Configuration parameters
- Server Configuration broken into two areas:
  - External Communication
  - Internal Processing
- In Tomcat two separate Java Packages:
  - Coyote
  - Catalina
- Coyote→ Provides HTTP 1.1 communication
  - Catalina→ Actual Servlet Container

  Coyote parameters affecting External Communication

- IP addresses and TCP ports
- Number of subtasks created when server initialized
- Max number of threads allowed to exist simultaneously
- Max no of TCP connection request that will be queued if server is running its max no of threads. If queue full the received connection request is refused.
- "Keep-alive" time for inactive TCP connections
- Settings of the parameter affect the performance of the server.
- Tuning the Server
  - Changing the values of these and similar parameters in order to optimize performance
- Tuning is done by trial and error
- Load generation or stress test tools used to simulate request to a web server helpful for experimenting with tuning parameters

Service has Five Components

- Connector, Host, Logger, Realm, and Valve
- Connector is a coyote component handles HTTP communication
- Clicking on the connector will produce the window containing the dropdown menus of possible action that can be performed for this component

Defining Virtual Hosts

Configuring Host Elements

- The Host element represents a virtual host, which is an association of a network name for a server (such as www.mycompany.com) with the particular server on which Tomcat is running.

Host Attributes

- The attributes shown in following table may be viewed, set, or modified for a Host.
- Web server logs record information about server activity
- Access log is a file that records information about every HTTP request processed by the server
- Message logs → variety of debugging and other information generated by web server
- Access logging is performed by adding a valve component

Logging

- Web server logs record information about server activity
- Access log is a file that records information about every HTTP request processed by the server
- Message logs → variety of debugging and other information generated by web server
- Access logging is performed by adding a valve component

Access Control

- Provide automatic password protection for resources
- Access control:
  - Password protection (e.g., admin pages)
- Users and roles defined in
  conf/tomcat-users.xml
  - Deny access to machines
- Useful for denying access to certain users by denying access from the machines they use

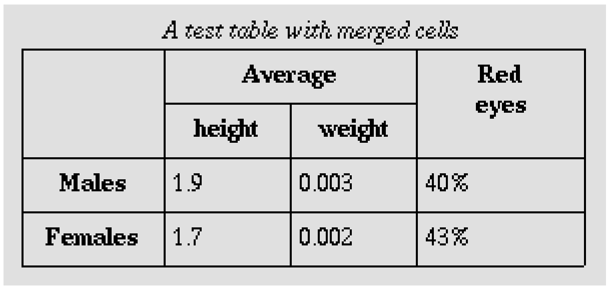
List of denied machines maintained in RemoteHostValve (deny by host name) or RemoteAddressValve (deny by IP address)

6. i) Explain how tables can be inserted into HTML document with example.

- The HTML table model allows authors to arrange data -- text, preformatted text, images, links, forms, form fields, other tables, etc. -- into rows and columns of cells.
- Each table may have an associated caption that provides a short description of the table's purpose. A longer description may also be for the benefit of people using speech or Braille-based user agents.
- Table rows may be grouped into a head, foot, and body sections, Row groups convey additional structural information and may be rendered by user agents in ways that emphasize this structure.
- User agents may exploit the head/body/foot division to support scrolling of body sections independently of the head and foot sections.
- When long tables are printed, the head and foot information may be repeated on each page that contains table data.
- Authors may also group columns to provide additional structural information that may be exploited by user agents.
- Furthermore, authors may declare column properties at the start of a table definition in a way that enables user agents to render the table incrementally rather than having to wait for all the table data to arrive before rendering.
- Table cells may either contain "header" information or "data. Cells may span multiple rows and columns.

Here's a simple table that illustrates some of the features of the HTML table model. The following table definition:

```
<TABLE border="1"
          summary="This table gives some statistics about fruit
                   flies: average height and weight, and percentage
                   with red eyes (for both males and females).">
<CAPTION><EM>A test table with merged cells</EM></CAPTION>
<TR><TH rowspan="2"><TH colspan="2">Average
    <TH rowspan="2">Red<BR>eyes
<TR><TH>height<TH>weight
<TR><TH>Males<TD>1.9<TD>0.003<TD>40%
<TR><TH>Females<TD>1.7<TD>0.002<TD>43%
</TABLE>
```


A test table with merged cells

	Average		Red eyes
	height	weight	
Males	1.9	0.003	40%
Females	1.7	0.002	43%

----

ii) What is the significance of using forms on the web page? Enlist various components used on form.

A **webform, or HTML** form on a web page allows a user to enter data that is sent to a server for processing. Forms can resemble paper or database forms because web users fill out the forms using checkboxes, radio buttons, or text fields. For example, forms can be used to enter shipping or credit card data to order a product, or can be used to retrieve search results from a search engine.

Forms are enclosed in the HTML form tag. This tag specifies the communication endpoint the data entered into the form should be submitted to, and the method of submitting the data, GET or POST.

Elements

Forms can be made up of standard graphical user interface elements:

- text input — a simple text box that allows input of a single line of text (an alternative, password, is used for security purposes, in which the character typed in are invisible or replaced by symbols such as *)
- radio — a radio button
- file — a file select control for uploading a file
- reset — a reset button that, when activated, tells the browser to restore the values to their initial values.
- submit — a button that tells the browser to take action on the form (typically to send it to a server)
- textarea — much like the text input field except a textarea allows for multiple rows of data to be shown and entered

- select — a drop-down list that displays a list of items a user can select from

uses of various elements:

- a text box asking for your name
- a pair of radio buttons asking you to pick your sex
- a select box giving you a list of eye colors to choose from
- a pair of check boxes to click on if they apply to you
- a text area to describe your athletic ability
- a submit button to send it to the server

These basic elements provide most common graphical user interface (GUI) elements, but not all. For example, there are no equivalents to a combo box, tree view, or grid view.

A grid view, however, can be mimicked by using a standard HTML table with each cell containing a text input element.

A tree view could also be mimicked through nested tables or, more semantically appropriately, nested lists.

In both cases, a server side process is responsible for processing the information, while JavaScript handles the user-interaction.

Implementations of these interface elements are available through JavaScript libraries such as jQuery.

HTML 4 introduced the label tag, which is intended to represent a caption in a user interface, and can be associated with a specific form control by specifying the id attribute of the control in the label tag's for attribute.

HTML 5 introduces a number of input tags that can be represented by other interface elements. Some are based upon text input fields and are intended to input and validate specific common data.

These include email to enter email addresses, tel for telephone numbers, number for numeric values.

There are additional attributes to specify required fields, fields that should have keyboard focus when the web page containing the form is loaded, and placeholder text that is displayed within the field but is not user input.

The date input type displays a calendar from which the user can select a date or date range.

And the color input type can be represented as an input text simply checking the value entered is a correct hexadecimal representation of a color, according to the specification, or a color picker widget.

7. **Discuss how to create list and frame using HTML. Give Example.**

HTML **frames** are used to divide your browser window into multiple sections where each section can load a separate HTML document. A collection of frames in the browser window is known as a frameset. The window is divided into frames in a similar way the tables are organized: into rows and columns.

Creating Frames

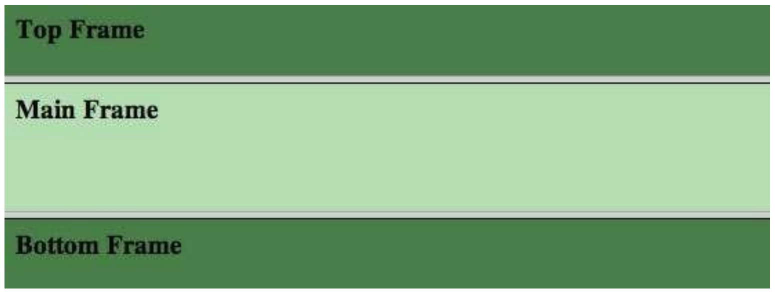
To use frames on a page we use <frameset> tag instead of <body> tag. The <frameset> tag defines how to divide the window into frames. The rows attribute of <frameset> tag defines horizontal frames and cols attribute defines vertical frames. Each frame is indicated by <frame> tag and it defines which HTML document shall open into the frame.

Example

Following is the example to create three horizontal frames:

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Frames</title>
</head>
<frameset rows="10%,80%,10%">
  <frame name="top" src="/html/top_frame.htm" />
  <frame name="main" src="/html/main_frame.htm" />
  <frame name="bottom" src="/html/bottom_frame.htm" />
  <noframes>
  <body>
    Your browser does not support frames.
  </body>
  </noframes>
</frameset>
</html>
```

This will produce following result:



The <frame> Tag Attributes

Following are important attributes of <frame> tag:

Attribute	Description
src	This attribute is used to give the file name that should be loaded in the frame. Its value can be any URL. For example, src="/html/top_frame.htm" will load an HTML file available in html directory.
name	This attribute allows you to give a name to a frame. It is used to indicate which frame a document should be loaded into. This is especially important when you want to create links in one frame that load pages into an another frame, in which case the second frame

	needs a name to identify itself as the target of the link.
frameborder	This attribute specifies whether or not the borders of that frame are shown; it overrides the value given in the frameborder attribute on the <frameset> tag if one is given, and this can take values either 1 (yes) or 0 (no).
marginwidth	This attribute allows you to specify the width of the space between the left and right of the frame's borders and the frame's content. The value is given in pixels. For example marginwidth="10".
marginheight	This attribute allows you to specify the height of the space between the top and bottom of the frame's borders and its contents. The value is given in pixels. For example marginheight="10".
noresize	By default you can resize any frame by clicking and dragging on the borders of a frame. The noresize attribute prevents a user from being able to resize the frame. For example noresize="noresize".
scrolling	This attribute controls the appearance of the scrollbars that appear on the frame. This takes values either "yes", "no" or "auto". For example scrolling="no" means it should not have scroll bars.
longdesc	This attribute allows you to provide a link to another page containing a long description of the contents of the frame. For example longdesc="framedescription.htm"

List :

HTML offers web authors three ways for specifying lists of information. All lists must contain one or more list elements. Lists may contain:

- <ul> - An unordered list. This will list items using plain bullets.
- <ol> - An ordered list. This will use different schemes of numbers to list your items.
- <dl> - A definition list. This arranges your items in the same way as they are arranged in a dictionary.

HTML Unordered Lists

An unordered list is a collection of related items that have no special order or sequence. This list is created by using HTML <ul> tag. Each item in the list is marked with a bullet.

Example

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Unordered List</title>
</head>
<body>
<ul>
<li>Beetroot</li>
<li>Ginger</li>
<li>Potato</li>
<li>Radish</li>
</ul>
</body>
</html>
```

This will produce following result:

- Beetroot
- Ginger
- Potato
- Radish

The type Attribute

You can use type attribute for <ul> tag to specify the type of bullet you like. By default it is a disc. Following are the possible options:

```
<ul type="square">
<ul type="disc">
<ul type="circle">
```

Example

Following is an example where we used <ul type="square">

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Unordered List</title>
</head>
<body>
  <ul type="square">
  <li>Beetroot</li>
  <li>Ginger</li>
  <li>Potato</li>
  <li>Radish</li>
  </ul>
</body>
</html>
```

This will produce following result:

- Beetroot
- Ginger
- Potato
- Radish

The start Attribute

You can use start attribute for <ol> tag to specify the starting point of numbering you need. Following are the possible options:

```
<ol type="1" start="4">    - Numerals starts with 4.
<ol type="I" start="4">    - Numerals starts with IV.
<ol type="i" start="4">    - Numerals starts with iv.
<ol type="a" start="4">    - Letters starts with d.
<ol type="A" start="4">    - Letters starts with D.
```

Example

Following is an example where we used <ol type="i" start="4" >

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Ordered List</title>
</head>
<body>
  <ol type="i" start="4">
  <li>Beetroot</li>
  <li>Ginger</li>
  <li>Potato</li>
  <li>Radish</li>
  </ol>
</body>
</html>
```

This will produce following result:

```
 iv.  Beetroot
  v.  Ginger
 vi.  Potato
vii.  Radish
```

8. **Explain the capabilities of Web Server (APR/MAY 2013)**

Basic functionality

- Server calls on TCP software and waits for connection req to one or more ports
- When a connection request is received , the server dedicates a subtask(Single copy of server

software handling a single client connection)

- Subtask establish connection and receives request
- Subtask examines the host header field to determine the host and invokes software for this host
- Virtual host software Map Request-URI to specific resource on the server.
- It maps Request-URI to specific resource associated with the virtual host
  - File: Return file in HTTP response (MIME Type)
  - Program: Run program and return output in HTTP response
- Log information about the request and response such as IP address and the status code in a plain-text file.
- If the TCP connection is kept alive , the server subtask continues to monitor the connection, the client send another request or initiates a connection close.

Few Definitions

- All modern servers concurrently process multiple requests
- Multiple copies of the server running simultaneously(Concurrency)
- Subtask→ Single copy of server software handling a single client connection
- Virtual Host→ HTTP request include a host header field
- Multiple host names mapped by a DNS to a single IP address
- Web server determine which virtual host is being requested by examining the host header field.

Server Configuration and Tuning

- Modern servers have large number of Configuration parameters
- Server Configuration broken into two areas:
  - External Communication
  - Internal Processing
- In Tomcat two separate Java Packages:
  - Coyote
  - Catalina
- Coyote→ Provides HTTP 1.1 communication
  - Catalina→ Actual Servlet Container

Coyote parameters affecting External Communication

- IP addresses and TCP ports
- Number of subtasks created when server initialized
- Max number of threads allowed to exist simultaneously
- Max no of TCP connection request that will be queued if server is running its max no of threads. If queue full the received connection request is refused.
- "Keep-alive" time for inactive TCP connections
- Settings of the parameter affect the performance of the server.
- Tuning the Server
  - Changing the values of these and similar parameters in order to optimize performance
- Tuning is done by trial and error
- Load generation or stress test tools used to simulate request to a web server helpful for experimenting with tuning parameters

Service has Five Components

- Connector, Host, Logger, Realm, and Valve
- Connector is a coyote component handles HTTP communication
- Clicking on the connector will produce the window containing the dropdown menus of possible

action that can be performed for this component

Defining Virtual Hosts

Configuring Host Elements

- The Host element represents a virtual host, which is an association of a network name for a server (such as www.mycompany.com) with the particular server on which Tomcat is running.

Host Attributes

- The attributes shown in following table may be viewed, set, or modified for a Host.
- Web server logs record information about server activity
- Access log is a file that records information about every HTTP request processed by the server
- Message logs → variety of debugging and other information generated by web server
- Access logging is performed by adding a valve component

Logging

- Web server logs record information about server activity
- Access log is a file that records information about every HTTP request processed by the server
- Message logs → variety of debugging and other information generated by web server
- Access logging is performed by adding a valve component

Access Control

- Provide automatic password protection for resources
- Access control:
  - – Password protection (e.g., admin pages)
    - Users and roles defined in conf/tomcat-users.xml
  - – Deny access to machines
    - Useful for denying access to certain users by denying access from the machines they use

List of denied machines maintained in RemoteHostValve (deny by host name) or RemoteAddressValve (deny by IP address)

9. **Explain about the XHTML DTD with an Example.**

DTD's

XHTML documents have three parts: the DOCTYPE (which contains the DTD declaration), the head and the body. To create web pages that properly conform to the XHTML 1.0 standard, each page must include a DTD declaration; either **strict, transitional, or frameset.**

1. Strict

You should use the strict DTD when your XHTML pages will be marked up cleanly, free of presentational clutter. You use the strict DTD together with cascading style sheets, because it doesn't allow attributes like "bgcolor" to be set for the <body> tag, etc.

The strict DTD looks like this:

```
<!DOCTYPE html
        PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
```

```
<title> Strict DTD XHTML Example </title>
</head><body><p>
Please Choose a Day:
<br /><br />
<select name="day">
<option selected="selected">Monday</option>
<option>Tuesday</option>
<option>Wednesday</option>
</select>
</p></body>
</html>
```

2. Transitional

The transitional DTD should be used when you need to take advantage of the presentational features that are available through HTML. You should also use the transitional DTD when you want to support older browsers that don't have built-in support for cascading style sheets.

The transitional DTD looks like this:

<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"DTD/xhtml1-transitional.dtd">
    <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
    <title> Transitional DTD XHTML Example </title>
    </head>

    <body bgcolor="#FFFFFF" link="#000000" text="red">
    <p>This is a transitional XHTML example</p>
    </body>
    </html>
```

3. Frameset

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">

You should use the frameset DTD when your XHTML page will contain frames. The frameset DTD looks like this:

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"DTD/xhtml1-frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title> Frameset DTD XHTML Example </title> </head>
<frameset cols="100,*">
<frame src="toc.html" />
```

	`<frame src="intro.html" name="content" />`
	`</frameset> </html>`

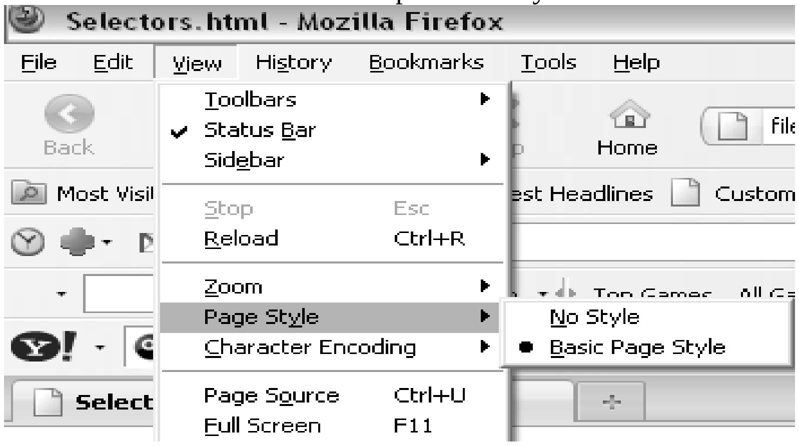
10.	**Explain the significance of XHTML with the help of a real time application. Write necessary code snippets (MAY/JUNE 2014)**
	Extensible Hypertext Markup Language (XHTML) is a family of XML markup languages that mirror or extend versions of the widely used Hypertext Markup Language (HTML), the language in which Web pages are formulated.
	<u>Significance</u>
	• **Sustainability.**
	Web applications tend towards XML. Using XHTML now instead of HTML makes any future conversion of the website easier.
	• **Extensibility.**
	Owing to the extensibility of XML, XHTML documents can be supplemented with other forms of markup, MathML (Math Markup Language) SVG (Scalable Vector Graphics) or your own markup variants, thanks to the use of namespaces.
	• **Compatibility.**
	Because XHTML documents are written in compliance with the rules of XML, XML-processing programmes can effortlessly convert an XHTML file to another format (e.g. PDF, RSS or RTF).
	• **Efficiency of processing applications.**
	Once browsers support XHTML documents and the strict rules of XML, they will become quicker thanks to shorter error processing routines. At present, a great deal of the processing power of a browser is still spent on liberal error processing of documents containing malformed HTML markup.
	<u>Features</u>
	• XHTML requires strict adherence to coding rules.
	• XHTML encourages a more structured and conceptual way of thinking about content and, combined with the style sheet, a more creative way of displaying it.
	• XHTML makes it easier for people to dream up and add new elements.
	<u>Code snippets</u>
	`<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">`
	`<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">`
	`<head>`
	`    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>`
	`    <title>Page Title</title>`
	`    <link rel="stylesheet" href="style.css" type="text/css" media="screen" charset="utf-8"/>`
	`</head>`
	`<body>`
	`</body>`
	`</html>`

11.	**Explain about Style Rule Cascading and Inheritance** **Rule cascading:** **{ font-weight : bold } applies to every element of HTML* *#p1, #p3 { background-color : aqua }* *#p3{ font-weight : bold }* **What if more than one style declaration applies to a property of an element?** • Multiple declaration • Browser applies rule cascading • A multistage sorting process that selects a single declaration that supply the property value • The CSS rule cascade determines which style rule's declaration applies

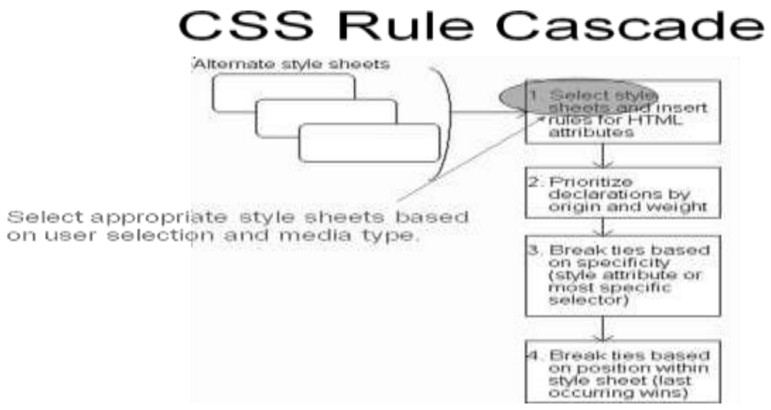
- Once declaration identified, associate **origin and weight** with every declaration
  - PERSON WHO WROTE THE DOCUMENT
  - PERSON WHO IS VIEWING THE DOCUMENT
  - PERSON WHO WROTE THE BROWSER SOFTWARE

Origin of a declaration is one of the following:

- Author-> declaration is part of an external or embed style sheet or part of the value specified by style attribute
- User agent –>A browser define default style property values for HTML elements
- User-> Modern browsers allow users to provide a style sheet



- **User style rules defined in two ways:**
  - **Edit|Preferences|Appearance,** Fonts and colors panels allow a user to select various style options
  - User can explicitly create a style sheet file the browser will input when it is started
  - Features provided by IE6
    - **Tools|Internet Options**
- **Two weight values**
  - Normal
  - Important
- User/important highest priority in CSS to accommodate users with special needs


CSS Rule Cascade
Alternate style sheets
Select appropriate style sheets based on user selection and media type.
1. Select style sheets and insert rules for HTML attributes
2. Prioritize declarations by origin and weight
3. Break ties based on specificity (style attribute or most specific selector)
4. Break ties based on position within style sheet (last occurring wins)

- Rules made important by adding "!important":

```
p { text-indent:3em; font-size:larger !important }
```

Cascading order

What style will be used when there is more than one style specified for an HTML element?

all the styles will "cascade" into a new "virtual" style sheet by the following rules, where number four has the highest priority:

1. **Browser default**
2. **External style sheet**
3. **Internal style sheet (in the head section)**
4. **Inline style (inside an HTML element)**

So, an inline style (inside an HTML element) has the highest priority, which means that it will override a style defined inside the <head> tag, or in an external style sheet, or in a browser (a default value).

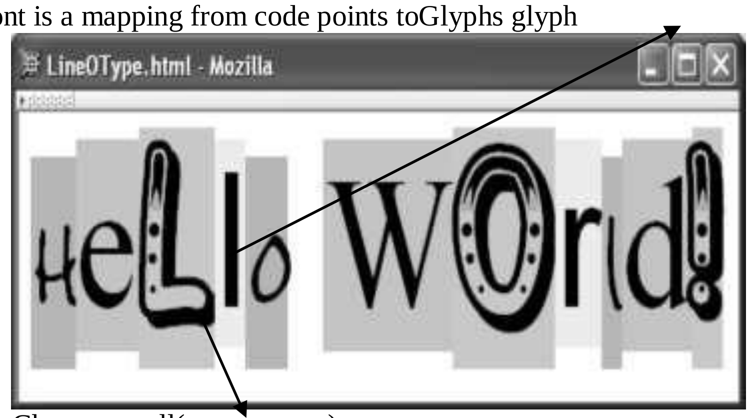
Note: If the link to the external style sheet is placed after the internal style sheet in HTML <head>, the external style sheet will override the internal style sheet!

Style Inheritance

- Cascading based on structure of style sheets
- Inheritance based on tree structure of document
- What if no style declaration applies to a property of an element?
- Generally, the property value is inherited from the nearest ancestor element that has a value for the property
- If no ancestor has a value (or the property does not inherit) then CSS defines an initial value that is used
- Property values:
  - o Specified: value contained in declaration
    - Absolute: value can be determined without reference to context (*e.g.*, 2cm)
    - Relative: value depends on context (*e.g.*, larger)
  - o Computed: browser performs calculation depends on particular relative value
  - o absolute representation of relative value (*e.g.*, larger might be 1.2 x parent font size)
  - o Actual: value actually used by browser (*e.g.*, computed value might be rounded)
- Most properties inherit computed value
  - o Exception discussed later: line-height
- A little thought can usually tell you whether a property inherits or not
  - o Example: height does not inherit

12. **Explain any eight CSS text properties.**

A font is a mapping from code points toGlyphs glyph



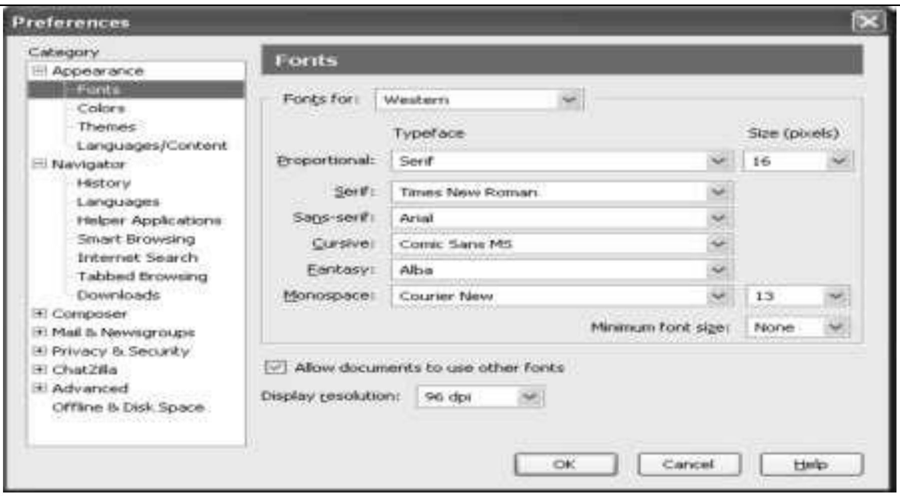
Character cell(content area)

A font family is a collection of related fonts(typically differ in size, weight, etc.)

```
<p style="font-family:'Jenkins v2.0'">
```

• font-family property can accept a list offamilies, including generic font families

```
font-family:"Edwardian Script ITC","French Script MT",cursive
```



Preferences

Category

- Appearance
  - Fonts
  - Colors
  - Themes
  - Languages/Content
- Navigator
  - History
  - Languages
  - Helper Applications
  - Smart Browsing
  - Internet Search
  - Tabbed Browsing
  - Downloads
- Composer
- Mail & Newsgroups
- Privacy & Security
- ChatZilla
- Advanced
- Offline & Disk Space

Fonts

Fonts for: Western

	Typeface	Size (pixels)
Proportional:	Serif	16
Serif:	Times New Roman	
Sans-serif:	Arial	
Cursive:	Comic Sans MS	
Fantasy:	Alba	
Monospace:	Courier New	13

Minimum font size: None

[✓] Allow documents to use other fonts

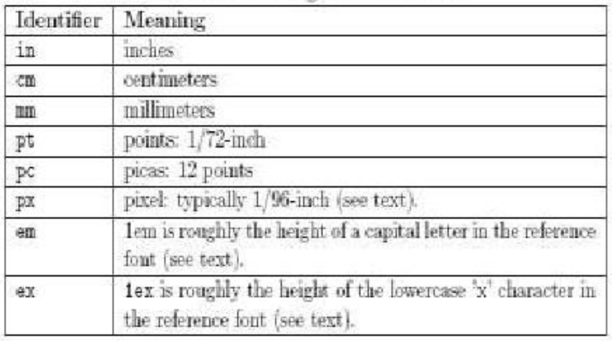
Display resolution: 96 dpi

LENGTH SPECIFICATION IN CSS:

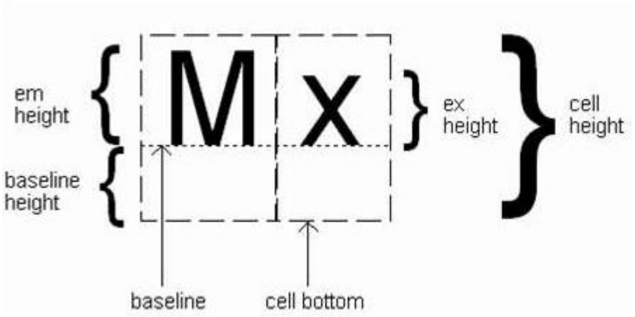
Many properties, such as font-size,have a value that is a CSS length

• All CSS length values except 0 need units

TABLE 3.4: CSS length unit identifiers.



Identifier	Meaning
in	inches
cm	centimeters
mm	millimeters
pt	points: 1/72-inch
pc	picas: 12 points
px	pixel: typically 1/96-inch (see text).
em	1em is roughly the height of a capital letter in the reference font (see text).
ex	1ex is roughly the height of the lowercase 'x' character in the reference font (see text).



Reference font defines em and ex units

– Normally, reference font is the font of theelement being styled

– Exception: Using em/ex to specify value for font-size

FONT PROPERTIES:

Other ways to specify value forfont-size:

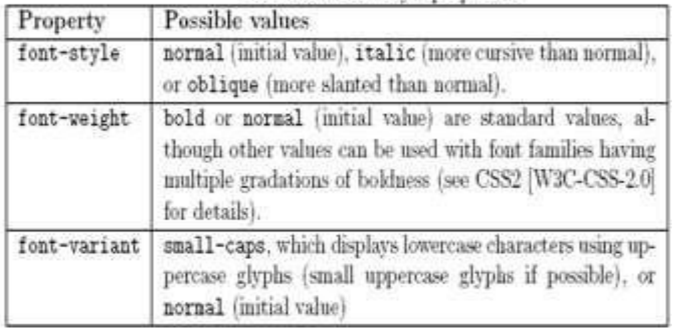
– Percentage (of parent font-size)

– Absolute size keyword: xx-small, x-small,small, medium (initial value), large,x-large, xx-large

• User agent specific; should differ by ~ 20%

– Relative size keyword: smaller, larger

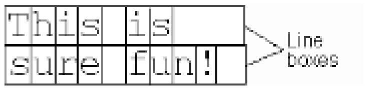
Additional font style properties.



Property	Possible values
`font-style`	**normal** (initial value), **italic** (more cursive than normal), or **oblique** (more slanted than normal).
`font-weight`	**bold** or **normal** (initial value) are standard values, although other values can be used with font families having multiple gradations of boldness (see CSS2 [W3C-CSS-2.0] for details).
`font-variant`	**small-caps**, which displays lowercase characters using uppercase glyphs (small uppercase glyphs if possible), or **normal** (initial value)

LINE BOXES:

Text is rendered using line boxes



• Height of line box given by line-height

– Initial value: normal (*i.e.*, cell height;relationship with em height is font-specific)

– Other values (following are equivalent):

```
line-height:1.5em
line-height:150%
line-height:1.5
```

font shortcut property:

`{ font: italic bold 12pt "Helvetica",sans-serif }` equals to

```
{ font-style: italic;
  font-variant: normal;
  font-weight: bold;
  font-size: 12pt;
  line-height: normal;
  font-family: "Helvetica",sans-serif }
```

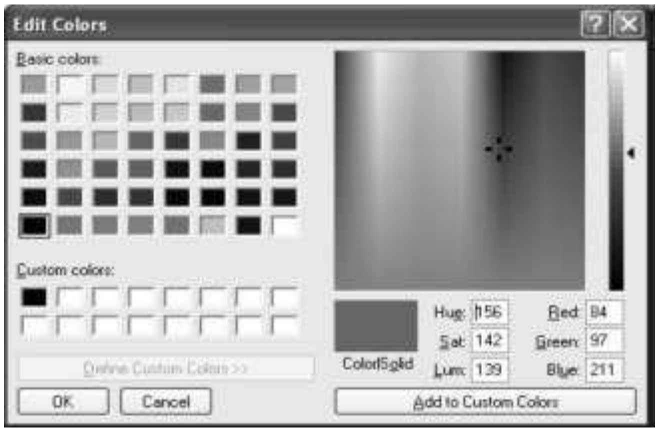
TEXT FORMATTING AND TEXT COLOR:

Font color specified by color property

• Two primary ways of specifying colors:

– Color name: black, gray, silver, white, red, lime, blue, yellow, aqua, fuchsia, maroon, green, navy, olive, teal, purple, full list at
http://www.w3.org/TR/SVG11/types.html#ColorKeywords

– red/green/blue (RGB) values



Hue: 156 Sat: 142 Lum: 139 Red: 84 Green: 97 Blue: 211

TABLE 3.7: Alternative formats for specifying numeric color values.

Format	Example	Meaning
Functional, integer arguments	`rgb(255,170,0)`	Use arguments as RGB values.
Functional, percentage arguments	`rgb(100%,66.7%,0%)`	Multiply arguments by 255 and round to obtain RGB values (at most one decimal place allowed in arguments).
Hexadecimal	`#ffaa00`	The first pair of hexadecimal digits represents the red intensity, second and third represent green and blue, respectively.
Abbreviated hexadecimal	`#fa0`	Duplicate the first hexadecimal digit to obtain red intensity, duplicate second and third to obtain green and blue, respectively.

13. Explain about the various style sheets with examples. (Internal,External,Inline) (APR/MAY 2013)

1. To create an inline style
   a. Add the style attribute to the HTML tag.
   b. The style declaration must be enclosed within double quotation marks.
2. To create an embedded style
   a. Insert a <style> tag within the head section of HTML file.
   b. Within the <style> tag, enclose the style declarations need to the entire Web page.
   c. The style sheet language identifies the type of style used in the document.
   d. The default and the most common language is “text/css” for use with CSS.
3. To create an External styles
   a. Create a text file containing style declarations
   b. Create a link to that file in each page of the Web site using a <link> tag.
   c. Specify the link attributes, such as href, rel, and type.
   d. Link a style sheet, the value of the href attribute should be the “URL” of the linked document, the value of the rel attribute should be “stylesheet” and the value of the type attribute should be

"text/css".

EXTERNAL.CSS:

```
body{ background-color: gray;}
p { color: blue; }
h3{ color: white; }
```

EXTERNAL.HTML:

```
<html>
<head>
<link rel="stylesheet" type="text/css" href="EXTERNAL.css" /><!—Link tag for External CSS-->
</head>
<body>
<h3> A White Header </h3>
<p> This paragraph has a blue font.
The background color of this page is gray because
we changed it with CSS! </p>
</body>
</html>
```

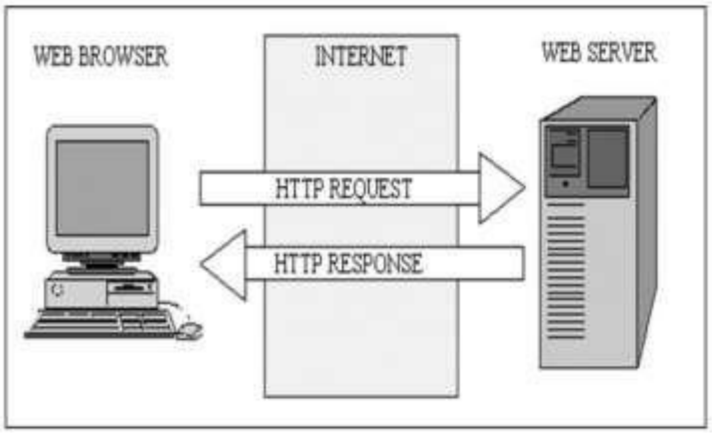
INTERNAL.HTML:

```
<html>
<head>
<style> <!—Style tag for Internal CSS-->
body { background-color: blue; }
p { color: white; }
</style>
</head>
<body>
<h2>Internal CSS</h2>
<p>This page uses internal CSS. Using the style tag we are able to modify
the appearance of HTML elements.</p>
</body>
</html>
```

INLINE.HTML:

```
<html>
<head>
</head>
<body>
<h2>InLINE CSS</h2>
<p style="color:sienna;margin-left:20px"><!—Style Attribute(INLINE)-->
This page uses INLINE CSS. Using the style ATTRIBUTE we are able to modify
the appearance of HTML elements.
</p>
</body>
</html>
```

14	**Difference between web browser and web server**



Web server and web browser are the terms which are commonly used for website. The basic purpose of both is to develop a platform for internet web directory. So that any users can anytime access any kind of website. Major difference between them is on their function and how they perform their functions. Check for the detail of both topics before understanding the differences between them.

Web Server:

Web server is a computer system, which provides the web pages via HTTP (Hypertext Transfer Protocol). IP address and a domain name is essential for every web server. Whenever, you insert a URL or web address into your web browser, this sends request to the web address where domain name of your URL is already saved. Then this server collects the all information of your web page and sends to browser, which you see in form of web page on your browser. Making a web server is not a difficult job. You can convert your computer into a web server with the help of any server software and connecting the computer to the internet. Lot of web server software are available in the market in shape of NCSA, Apache, Microsoft and Netscape. Storing, processing and delivering web pages to clients is its main function. All the communication between client (web browser) and server takes place via HTTP.

Web Browser:

Web browser is a client, program, software or tool through which we sent HTTP request to web server. The main purpose of web browser is to locate the content on the World Wide Web and display in the shape of web page, image, audio or video form. You can call it a client server because it contacts the web server for desired information. If the requested data is available in the web server data then it will send back the requested information again via web browser. Microsoft Internet Explorer, Mozilla Firefox, Safari, Opera and Google Chrome are examples of web browser and they are more advanced than earlier web browser because they are capable to understand the HTML, JavaScript, AJAX, etc. Now a days, web browser for mobiles are also available, which are called microbrowser.

Difference:

Following are the differences between web server and web browser.

- Web server is essential to store all information and data of websites. While web browser are used to access and locate these information and data.

- Web browser is used to search something on the internet via websites. While web server is used to make the links between websites and web browser.
- Web browser is a software or application which is used for collection and presentation of data in shape of websites while web server is a program server on computer or in cloud on internet that gives the data.

15 **Difference between internet and intranet.**

Internet

It is a worldwide system which has the following characteristics:

- Internet is a world-wide / global system of interconnected computer networks.
- Internet uses the standard Internet Protocol (TCP/IP)
- Every computer in internet is identified by a unique IP address.
- IP Address is a unique set of numbers (such as 110.22.33.114) which identifies a computer's location.
- A special computer DNS (Domain Name Server) is used to give name to the IP Address so that user can locate a computer by a name.
- For example, a DNS server will resolve a name **http://www.tutorialspoint.com** to a particular IP address to uniquely identify the computer on which this website is hosted.
- Internet is accessible to every user all over the world.

Intranet

- Intranet is system in which multiple PCs are connected to each other.

- PCs in intranet are not available to the world outside the intranet.
- Usually each company or organization has their own Intranet network and members/employees of that company can access the computers in their intranet.
- Each computer in Intranet is also identified by an IP Address which is unique among the computers in that Intranet.

Similarities in Internet and Intranet

- Intranet uses the internet protocols such as TCP/IP and FTP.
- Intranet sites are accessible via web browser in similar way as websites in internet. But only members of Intranet network can access intranet hosted sites.
- In Intranet, own instant messengers can be used as similar to yahoo messenger/ gtalk over the internet.

Differences in Internet and Intranet

- Internet is general to PCs all over the world whereas Intranet is specific to few PCs.
- Internet has wider access and provides a better access to websites to large population whereas Intranet is restricted.
- Internet is not as safe as Intranet as Intranet can be safely privatized as per the need.

16

Building Advanced Web 2.0 Applications.

Definition of Mash up applications

• A *mashup* is similar to a remix. You might have heard examples again from the music world where elements of Led Zeppelin are combined with Jayzee, for example, to form a weird rap/rock song. That's a mashup. The same can be done with data from the Internet.

Mashup Techniques - Mashing on the Web Server

• Every site sits on a Web server. It's the thing that serves up the page, typically Internet Information Server (IIS) in the Microsoft world.

Understanding the Architecture How it works

This use case is definitely the most straightforward:

• The Web browser communicates with the server, requesting a page using straight HTTP or HTTPS.

• That page is constructed by the Web server, which reaches out to what I'll call the *source* or *partner sites* (for example, Amazon, Yahoo, or Google, and so on). The first request in this example is to Amazon using the Simple Object Access Protocol (SOAP) over HTTP.

• Amazon returns back a SOAP response.

• The second request in this example is to Yahoo using a Representational State Transfer style approach.

• Yahoo responds with Plain Old XML over HTTP.

• Lastly, the Web server aggregates the responses, combining and rationalizing the data in whatever manner makes sense.

• The resulting data is bound to the HTML and inserted into the response, which is sent back to the browser.

Pros and Cons

• The benefits of this approach are that the browser is decoupled entirely from the partner sites supplying the data. The Web server acts as a proxy and aggregator for the responses.

• Disadvantages of this approach are that the browser requests an entire page, which typically is acceptable.

• Second, the Web server is doing all the work in terms of data manipulation. Though this is good in terms of maintenance, it's not so good in terms of scalability. When your mashup gains popularity and starts being viewed by thousands of users, the amount of work the server's doing increases, while the browser residing at the client is relatively idle.

Remote Data Communication

• Remote data communication occurs at runtime.

• Flex applications support a variety of remote data communication techniques built on standards.

• There are three basic categories of Flex application remote data communication:

HTTP request/response-style communication

• This category consists of several overlapping techniques. Utilizing the Flex framework HTTPService component or the Flash Player API URLLoader class, you can send and load uncompressed data such as text blocks, URL encoded data, and XML packets Each technique achieves the similar goal of sending requests and receiving responses using HTTP or HTTPS.

Real-time communication

• This category consists of persistent socket connections. Flash Player supports two types of socket connections: those that require a specific format for packets (XMLSocket) and those that allow raw socket connections (Socket)

File upload/download communication

• This category consists of the FileReference API which is native to Flash Player and allows file upload and download directly within Flex applications.

Understanding Strategies for Data Communication

• When you build Flex applications that utilize data communication, it's important to understand the strategies available for managing those communications and how to select the right strategy for an application. All Flex applications run in Flash Player. With the exception of some Flex applications created using Flex Data Services, almost all Flex applications are composed of precompiled *.swf* files that are loaded in Flash Player on the client.

• Because Flex applications are stateful and self-contained, they don't require new page requests and wholesale screen refreshes to make data requests and handle responses.

• The Flex framework provides components for working with data communication using standard HTTP requests as well as SOAP requests.

Working with Request/Response Data Communication

• You can work with request/response data communication in three basic ways: via simple HTTP services, web services, and Flash Remoting.

Simple HTTP Services

• The most basic type of HTTP request/response communication uses what we call *simple HTTP services*. These services include things such as text and XML resources, either in static documents or dynamically generated by something such as a ColdFusion page, a servlet, or an ASP.NET page.

HTTPService

• HTTPService is a component that allows you to make requests to simple HTTP services such as text files, XML files, or scripts and pages that return dynamic data. You must always define a value for the url property of an HTTPService object.

• The following example uses MXML to create an HTTPService object that loads text from a file called *data.txt* saved in the same directory as the compiled *.swf* file:

```
<mx:HTTPService id="textService" url="data.txt" />
```

Sending requests

• Creating an HTTPService object does not automatically make the request to load the data. In order to make the request, you must call the send( ) method. If you want to load the data when the use clicks a button, you can call the send( ) method in response to a click event:

```
textService.send( );
```

Handling results

• The send( ) method makes the request, but a response is not likely to be returned instantaneously. Instead, the application must wait for a result event. The following example displays an alert when the data loads:

```
<mx:HTTPService id="textService" url="data.txt" result="mx.controls.Alert.show('Data loaded')" />
```

Sending parameters

• When you want to pass parameters to the service, you can use the request property of the HTTPService instance. The request property requires an Object value. By default, the name/value pairs of the object are converted to URL-encoded format and are sent to the service using HTTP GET.

• The default value is object, which yields the default behavior you've already seen. You can optionally

specify any of the following values:
Text:The data is not parsed at all, but is treated as raw text.
Flashvars:The data is assumed to be in URL-encoded format, and it will be parsedinto an object with properties corresponding to the name/value pairs.
Array:The data is assumed to be in XML format, and it is parsed into objects much the same as with the object settings. However, in this case, the result is always an array. If the returned data does not automatically parse into an array, the parsed data is placed into an array.
Xml:The data is assumed to be in XML format, and it is interpreted as XML using the legacy XMLNode ActionScript class.
e4x:The data is assumed to be in XML format, and it is interpreted as XML using the ActionScript 3.0 XML class (E4X).

Using HTTPService with ActionScript

- Although the simplest and quickest way to use an HTTPService object is to primarily use MXML, this technique is best-suited to nonenterprise applications in which the data communication scenarios are quite simple.
- Because HTTPService components provide significant data conversion advantages (such as automatic serialization of data), it is still frequently a good idea to use an HTTPService object within a remote proxy. However, it is generally necessary to then work with the HTTPService component entirely with ActionScript, including constructing the object and handling the responses. URLLoader
- HTTPService allows you to use requests and handle responses to and from simple HTTP services. You can optionally use the Flash Player class called flash.net.URLLoader to accomplish the same tasks entirely with ActionScript, but at a slightly lower level.
- The first step when working with a URLLoader object is always to construct the object using the constructor method, as follows:

var loader:URLLoader = new URLLoader( );

- Once you've constructed the object, you can do the following:
    - Send requests.
    - Handle responses.
    - Send parameters.

Sending requests

- You can send requests using the load( ) method of a URLLoader object. The load( ) method requires that you pass it a flash.net.URLRequest object specifying at a minimum what URL to use when making the request. The following makes a request to a text file called *data.txt*:

loader.load(new URLRequest("data.txt"));

Handling responses

- URLLoader objects dispatch complete events when a response has been returned. Any return value is stored in the data property of the URLLoader object.

Sending parameters

- You can send parameters using URLLoader as well. In order to send parameters, you assign a value to the data property of the URLRequest object used to make the request. The URLRequest object can send binary data or string data.

Web Services

- Flash Player has no built-in support for SOAP web services. However, Flex provides a WebService component that uses built-in HTTP request/response support as well as XML support to enable you to

	work with SOAP-based web services. There are two ways you can work with the WebService components: using MXML and using ActionScript. Using WebService Components with MXML
	• You can create a WebService component instance using MXML. When you do, you should specify an id and a value for the wsdl property.\
	Eg: <mx:WebService id="statesService" wsdl="http://www.rightactionscript.com/states/ webservice/StatesService.php?wsdl" />
	Web services define one or more methods or operations. You must define the WebService instance so that it knows about the operations using nested operation tags. The operation tag requires that you specify the name at a minimum.
	
	Calling web service methods
	• All operations that you define for a WebService component instance are accessible as properties of the instance. For example, in the preceding section, we created a WebService instance called statesService with an operation called getCountries. That means you can use ActionScript to reference the operation as statesService. getCountries.
	• You can then call getCountries just as though it were a method of statesService:
	statesService.getCountries( );
	
	Handling results
	• When a web service operation returns a result, you can handle it in one of two ways: explicitly handle the result event or use data binding. Then, once a result is returned, you can retrieve the result value from the lastResult property of the operation.
	
	Using WebService Components with ActionScript
	• You can use a WebService component using ActionScript instead of MXML. This is useful in cases where you want to fully separate the view from the controller and the model, such as in the recommended remote proxy approach.
	• The MXML version of the WebService component is an instance of mx.rpc.soap.mxml.WebService, which is a subclass of mx.rpc.soap.WebService. When you use the component directly from ActionScript you should instantiate mx.rpc.soap.WebService directly:
	// Assume the code already has an import statement for mx.rpc.soap.WebService.
	var exampleService:WebService = new WebService( );
	• Next, you must call a method called loadWSDL( ). You must call the method prior to calling any of the web service operations. Assuming you set the wsdl property, you don't need to pass any parameters to loadWSDL( ):
	exampleService.loadWSDL( );

	Part – A
1.	**What is JavaScript?** JavaScript is a platform-independent, event-driven, interpreted client-side scripting language developed by Netscape Communications Corp. and Sun Microsystems.
2.	**What are the primitive data types in javascript?** JavaScript supports five primitive data types: number, string, Boolean, undefined, and null. These types are referred to as primitive types because they are the basic building blocks from which more complex types can be built. Of the five, only number, string, and Boolean are real data types in the sense of actually storing data. Undefined and null are types that arise under special circumstances.
3.	**What are the Escape Codes Supported in JavaScript?**

	The Escape codes supported in javascript are \b Backspace,\t Tab (horizontal), \n Linefeed (newline),\v Tab (vertical),\f Form feed,\r Carriage return,\" Double quote \' Single quote,\\ Backslash.
4.	**What is JavaScript name spacing? How and where is it used?** Using global variables in JavaScript is evil and a bad practice. That being said, namespacing is used to bundle up all your functionality using a unique name. In JavaScript, a namespace is really just an object that you've attached all further methods, properties and objects. It promotes modularity and code reuse in the application.
5.	**How many looping structures can you find in javascript?** If you are a programmer, you know the use of loops. It is used to run a piece of code multiple times according to some particular condition. Javascript being a popular scripting language supports the following loops for, while, do-while loop
6.	**Mention the various Java Script Object Models.** Math Object, String Object, Date Object, Boolean and Number Object, Document Object Window Object.
7.	**How Scripting Language Is Differs from HTML?** HTML is used for simple web page design, HTML with FORM is used for both form design and Reading input values from user, Scripting Language is used for Validating the given input values weather it is correct or not, if the input value is incorrect, the user can pass an error message to the user, Using form concept various controls like Text box, Radio Button, Command Button, Text Area control and List box can be created.
8.	**What are the different types of objects in JavaScript?**

Type	Example	Implementation Provided By	Governing Standard
User-defined	Programmer defined Customer or Circle	Programmer	None
Built-in	Array, Math	The browser via engine its JavaScript	ECMA-262
Browser	Window, Navigator	The browser	None (though some portions adhere to an adhoc standard)
Document	Image, HTMLInputElement	The browser via its DOM engine	W3C DOM

9.	**Justify "JavaScript" is an event-driven programming"** Javascript supports event driven programming. when user clicks the mouse or hit the keys on the keyboard or if user submits the form then these events and response to them can be handled using javascript. Hence javascript is mainly used in web programming for validating the data provided by the user.
10.	**What is the use of pop up boxes in java script?** There are three types of popup boxes used in javascript. Using these popup boxes the user can interact with the web application.
11.	**What is DOM?** Document Object Model (DOM) is a set of platform independent and language neutral application interface (API) which describes how to access and manipulate the information stored in XML, XHTML and javascript documents.
12.	**Enlist any four mouse events.** The MouseEvent are-mousedown, mouseup, mouseover, mousemove, mouseout.
13.	**List ad various level of document object modeling.** Various levels of DOM are DOM0, Dom1, Dom2, and Dom3
14.	**What are they validation properties and methods?** Validation properties and methods are checkvalidity (), validaionMessage, customerror, patternMismatch, rangeOverflow, rangeUnderflow, tooLong.
15.	**Define event bubbling.** Suppose, there is an element present inside another element. Then during the event handling, if the event which is

	present in the inner element is handled and then the event of the outer element is handled. This process of event handling is called event bubbling

16.	**How to create arrays in Javascript?**
	We can declare an array like this Var scripts = new Array();
	We can add elements to this array like this
	scripts[0] = "PHP";
	scripts[1] = "ASP";
	scripts[2] = "JavaScript";
	scripts[3] = "HTML";
	Now our array scrips has 4 elements inside it and we can print or access them by using their index number. Note that index number starts from 0. To get the third element of the array we have to use the index number 2. Here is the way to get the third element of an array. document. write (scripts[2]); We also can create an array like this var no_array = new Array(21, 22, 23, 24, 25);

17.	**Write a simple program in JavaScript to validate the email-id.**
	`<!DOCTYPE html>`
	`<html>`
	`<head>`
	`<script>`
	`function validateForm() {`
	`  var x = document.forms["myForm"]["email"].value;`
	`  var atpos = x.indexOf("@");`
	`  var dotpos = x.lastIndexOf(".");`
	`  if (atpos<1 || dotpos<atpos+2 || dotpos+2>=x.length) {`
	`    alert("Not a valid e-mail address");`
	`    return false;}}`
	`</script>`
	`</head>`
	`<body>`
	`<form name="myForm" action="demo_form.asp" onsubmit="return validateForm();" method="post">`
	`Email: <input type="text" name="email">`
	`<input type="submit" value="Submit">`
	`</form>`
	`</body>`
	`</html>`

18.	**Write short notes on JDBC.** JDBC standard is intented for people developing industrial-strength database applications.JDBC makes java effective for developing enterprise information system.java.sql is the JDBC package that contains classes & interfaces that enable a java program to interact with a database.
19.	**Write short notes on JDBC drivers.** A JDBC driver is basically an implementation of the function calls specified in the JDBC API for a particular vendor’s RDBMS. Hence, a java program with JDBC function calls can access any RDBMS that has a JDBC driver available. A driver manager is used to keep track of all the installed drivers on the system. The operations of driver manager are getDriver, registerDriver, deregisterDriver.
20.	**What are the advantages of servlet over CGI?** ➢ Performance is significantly better, servlet execute within the address space of a web server. ➢ Servlets are platform independent ➢ The java security manager on the server enforces a set of restrictions to protect the resources on a server machine. ➢ The full functionality of java class libraries is available to a servlet.
21.	**Write down the methods of servlet interface**

	void destroy() –called when the servlet is unloaded. ServletConfig getServletConfig() –returns a ServletConfig object that contains any initialization parameters. String get ServletInfo() – returns a string describing the servlet. void init(ServletConfig sc) throws ServletException –called when the servlet is initialized .Initialization parameters for servlet can be obtained from sc. An unavailable exception should be thrown if the servlet is not initialized. Void Service(ServletRequest req,ServletResponse res) throws ServletException, IOException- Called to process a request from a client. The request from the client can be read from req. response to the client can be written to res. An exception is generated if a servlet or IO problem occurs.
22.	**What is the difference between CGI and servlets?** ➢ Performance is significantly better, servlet execute within the address space of a web server. ➢ Servlets are platform independent ➢ The java security manager on the server enforces a set of restrictions to protect the resources on a server machine. ➢ The full functionality of java class libraries is available to a servlet.
23.	**Define Servlet Life Cycle?** ➢ **init( )** method - invoked when the servlet is first loaded into memory ➢ **service( ) -** called for each HTTP request (for processing) ➢ **destroy( ) -** unloads the servlet from its memory.
24.	**What is JSP?** JavaServer Pages (JSP) is a technology for developing web pages that support dynamic content which helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.
25.	**What are advantages of using JSP?** • Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself. • JSP are always compiled before it's processed by the server unlike CGI/Perl which requires the server to load an interpreter and the target script each time the page is requested.
26.	**Explain lifecycle of a JSP.** • Compilation • Initialization • Execution • Cleanup
27.	**What are the types of directive tags?** The types directive tags are as follows: • **<%@ page ... %> :** Defines page-dependent attributes, such as scripting language, error page, and buffering requirements. • **<%@ include ... %> :** Includes a file during the translation phase. • **<%@ taglib ... %> :** Declares a tag library, containing custom actions, used in the page.
28.	**What are JSP actions?** JSP actions use constructs in XML syntax to control the behavior of the servlet engine. You can dynamically insert a file, reuse JavaBeans components, forward the user to another page, or generate HTML for the Java plugin.
	Part - B
1.	**How to write function using Java Script? Give Example.** A JavaScript function is a block of code designed to perform a particular task.A JavaScript function is executed when "something" invokes it (calls it). A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses **()**. Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas: **(**parameter1, parameter2, ...**)**

The code to be executed, by the function, is placed inside curly brackets: **{}**

Example:

```
<!DOCTYPE html>
<html>

<head>
<script>
function myFunction() {
   document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</head>

<body>

<h1>My Web Page</h1>

<p id="demo">A Paragraph</p>

<button type="button" onclick="myFunction()">Try it</button>

</body>
</html>
```

2. **Explain sub classes and super classes in Javascript.**

The top-most class, the class from which all other classes are derived, is the Object classdefined in java.lang . Object is the root of a hierarchy of classes, as illustrated in the following figure. The subclass inherits state and behavior in the form of variables and methods from its superclass.

the Class.create() method. Until now the only feature of classes defined this way was that the constructor called a method called initialize automatically.

```
var Person = Class.create();
Person.prototype = {
initialize: function(name) {
   this.name = name;
  },
  say: function(message) {
   return this.name + ': ' + message;
  }
};
var guy = new Person('Miro');
guy.say('hi');
// -> "Miro: hi"
   var Pirate = Class.create();
// inherit from Person class:
Pirate.prototype = Object.extend(new Person(), {
  // redefine the speak method
  say: function(message) {
    return this.name + ': ' + message + ', yarr!';
  }
});
    var john = new Pirate('Long John');
john.say('ahoy matey');
// -> "Long John: ahoy matey, yarr!"
```

Observe the direct interaction with class prototypes and the clumsy inheritance technique using Object.extend. Also, with Pirate redefining the say() method ofPerson, there is no way of calling the overridden method like you can do in programming languages that support class-based inheritance.

Example:

```
 <script language="javascript" type="text/javascript">
<!--
 function superClass() {
  this.supertest = superTest; //attach method superTest
}
 function subClass() {
  this.inheritFrom = superClass;
  this.inheritFrom();
  this.subtest = subTest; //attach method subTest
}
 function superTest() {
  return "superTest";
}
 function subTest() {
 return "subTest";
}

 var newClass = new subClass();
 alert(newClass.subtest()); // yields "subTest"
 alert(newClass.supertest()); // yields "superTest"

//-->
</script>
```

3. **Discuss Javascript objects in detail with suitable examples. (NOV/DEC 2012, MAY/JUNE 2014)**

- An object is a set of properties
- A property consists of a unique (within an object) name with an associated value
- The type of a property depends on the type of its value and can vary dynamically
- Object properties do not have data types
  Ex: Single property prop of an object o

```
o.prop = true;       prop is Boolean
o.prop = "true";     prop is now String
o.prop = 1;          prop is now Number
```

- There are no classes in JavaScript
- Object constructors defined to create objects and automatically define properties for the objects created
- Instead, properties and methods can be created and deleted dynamically

```
var o1 = new Object();
o1.testing = "This is a test";   Create an object o1
delete o1.testing;               Create property testing
                                 Delete testing property
```

- Objects are created using new expression
- First line creates a variable named o1 initialize its value of type object by calling built-in constructor Object()

```
new Object()   Constructor and argument list
```

- Second line adds a property named testing to the o1 object and assigns a string value to this property
- A constructor is a function
    - When called via new expression, a new empty Object is created and passed to the constructor along with the argument values
    - Constructor performs initialization on the object
        - Can add properties and methods to object
        - Can add object to an inheritance hierarchy from which it can inherit additional properties and methods
- The Object() built-in constructor
    - Does not add any properties or methods directly to the object
    - default toString() and valueOf() methods (used for conversions to String and Number, resp.)

```
o1.testing = "This is a test";
```

- Assign a value to an object property
- Property does not exist in the object
- Property with the given name is created in the object and assigned the specified value
- delete used to remove a property from an object
- Object initializer notation can be used to create an object (using Object() constructor) and one or more properties in a single statement

Enumerating Properties

- To know which property an object has at any given time
- Special form of for statement used to iterate through all properties of an object:

```
var hash = new Object();

hash.kim = "85";
hash.sam = "92";
hash.lynn = "78";
for (var aName in hash) {
  window.alert(aName + " is a property of hash.");
}
```

Array notation

- To print the values of those properties
- The JavaScript object dot notation is actually shorthand for a more general associative array notation in which Strings are array indices:
- Expressions can supply property names:
- Two different notations for accessing properties
- Dot notation
- Array reference syntax à index is viewed as string value hash
- Object can be viewed as a sort of array in which the elements are indexed by strings called associative

array

Object reference

```
StringBuffer s1 = new StringBuffer("Hello");
StringBuffer s2 = s1;
```

- Single StringBuffer is created and both s1 and s2 will be references to it
- Copies the reference from s1 to s2
- If code followed by s2. append("World!");
  System.out.println(s1);

- JavaScript: Value of Object is reference to object :

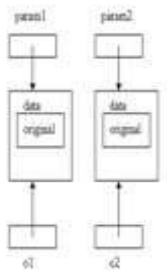
o2 is another name for o1

o1 is changed

```
var o1 = new Object();
o1.data = "Hello";
var o2 = o1;
o2.data += " World!";
window.alert(o1.data);
```

Output ? Hello World!

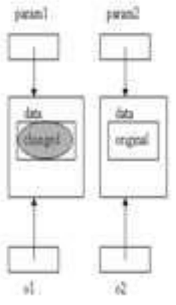
- Object argument values are references
- State of variables and parameters immediately before the first statement of the function objArgs() is executed

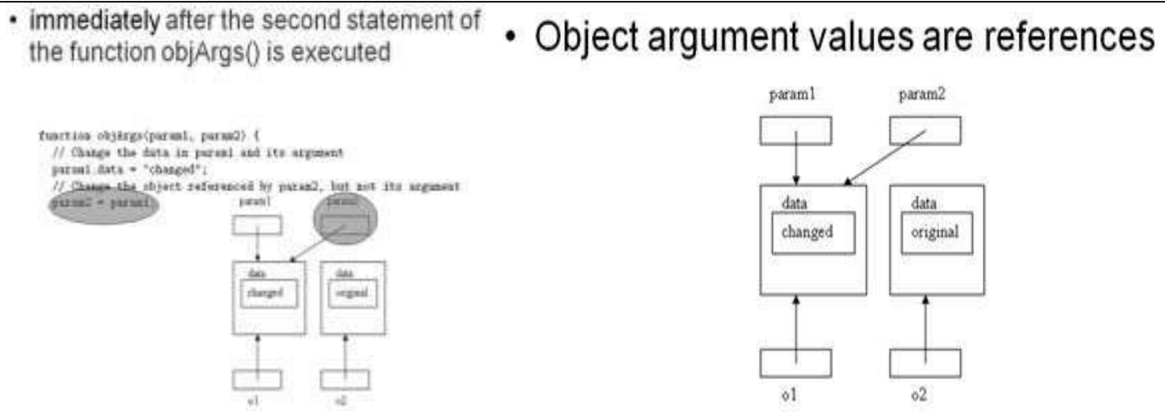


Object Values

- immediately after the first statement of the function objArgs() is executed

```
function objArgs(param1, param2)
{
 // Change the data in param1 and its argument
 param1.data = "changed";
 // Change the object referenced by param2, but not its
argument
 param2 = param1;

 window.alert("param1 is " + param1.data + "\n" +
          "param2 is " + param2.data);
 return;
}
```





- immediately after the second statement of the function objArgs() is executed
- Object argument values are references

```
function objArgs(param1, param2) {
  // Change the data in param1 and its argument
  param1.data = "changed";
  // Change the object referenced by param2, but not its argument
  param2 = param1;
```

Methods

- JavaScript functions are stored as values of type Object
- A function declaration creates a function value and stores it in a variable (property of window) having the same name as the function

A method is an object property for which the value is a function

4. **Discuss about Javascript debugging. Explain how local and global functions can be written using java script (MAY/JUNE 2012)**

A debugger is an application that places all aspects of script execution under the control of the programmer. Debuggers provide fine-grained control over the state of the script through an interface that allows you to examine and set values as well as control the flow of execution. Once a script has been loaded into a debugger, it can be run one line at a time or instructed to halt at certain breakpoints. Once execution is halted, the programmer can examine the state of the script and its variables in order to determine if something is amiss. You can also watch variables for changes in their values. The latest version of the Mozilla JavaScript Debugger for both Mozilla and Netscape browsers

<u>Local and global functions</u>

When a function is defined certain variables used for storing values are incorporated inside the function. These variables are found and used only inside these functions. Since functions are separate from the main code, it's advisable to use variables that are initialized only when a function is called and die when the execution comes out of the function. Variables that exist only inside a function are called Local variables. They have no presence outside the function. The values of such Local variables cannot be changed by the main code or other functions. This results in easy code maintenance and is especially helpful if many programmers are working together on the same project.

Variables that exist throughout the script are called Global variables. Their values can be changed anytime in the code and even by other functions.

In JavaScript, an inner (nested) function stores references to the local variables that are present in the same scope as the function itself, even after the function returns. This set of references is called a closure.

```
function myFunction() {
   var a = 4;
   return a * a;
}
```

The JavaScript global properties and functions can be used with all the built-in JavaScript objects.

```
var uri = "my test.asp?name=ståle&car=saab";
var enc = encodeURI(uri);
var dec = decodeURI(enc);
var res = enc + "<br>" + dec;
```

5. **Explain the way in which java script handles arrays with example. (MAY/JUNE 2012)**

Array

An array is a special variable, which can hold more than one value at a time.
If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
var car1 = "Saab";
var car2 = "Volvo";
var car3 = "BMW";
```

Creating an Array

Using an array literal is the easiest way to create a JavaScript Array.

Syntax:

```
var array-name = [item1, item2, ...];
```

Example:

```
var cars = ["Saab", "Volvo", "BMW"];
```

Using the JavaScript Keyword new

The following example also creates an Array, and assigns values to it:

Example: var cars = new Array("Saab", "Volvo", "BMW");

Access the Elements of an Array

You refer to an array element by referring to the index number. This statement access the value of the first element in myCars:

```
var name = cars[0];
```

This statement modifies the first element in cars:

```
cars[0] = "Opel";
```

Note [0] is the first element in an array. [1] is the second. Array indexes start with 0.

Different Objects in One Array

JavaScript variables can be objects. Arrays are special kinds of objects.
Because of this, you can have variables of different types in the same Array.
You can have objects in an Array. You can have functions in an Array. You can have arrays in an Array:

```
myArray[0] = Date.now;
myArray[1] = myFunction;
myArray[2] = myCars;
```

Arrays are Objects

Arrays are a special type of objects. The typeof operator in JavaScript returns "object" for arrays.But, JavaScript arrays are best described as arrays. Arrays use numbers to access its "elements". In this example, person[0] returns John:

Array: var person = ["John", "Doe", 46];

Object: var person = {firstName:"John", lastName:"Doe", age:46};

Array Properties and Methods

The real strength of JavaScript arrays are the built-in array properties and methods:

Examples

```
var x = cars.length;     // The length property returns the number of elements in cars
var y = cars.sort();     // The sort() method sort cars in alphabetical order
```

The length Property

The length property of an array returns the length of an array (the number of array elements).Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.length;              // the length of fruits is 4
```

Adding Array Elements

The easiest way to add a new element to an array is to use the length property:
Example

	var fruits = ["Banana", "Orange", "Apple", "Mango"];

	fruits[fruits.length] = "Lemon"; // adds a new element (Lemon) to fruits
	Adding elements with high indexes can create undefined "holes" in an array:
	Example
	var fruits = ["Banana", "Orange", "Apple", "Mango"];
	fruits[10] = "Lemon"; // adds a new element (Lemon) to fruits
	Looping Array Elements
	The best way to loop through an array is using a standard for loop:
	Example
	var index; var fruits = ["Banana", "Orange", "Apple", "Mango"];
	for (index = 0; index < fruits.length; index++) {
	text += fruits[index]; }

6.	**i) Write a Java script to find the factorial of the given number.**
	`<html>`
	`<head>`
	`<script type="text/javascript">`
	`function factorial(f,n)`
	`{`
	`l=1;`
	`for(i=1;i<=n;i++)`
	`l=l*i;`
	`f.p.value=l;`
	`}`
	`</script>`
	`</head>`
	`<body>`
	`<form>`
	`number:<input type="text" name="t"></br>`
	`<input type="button" value="submit" onClick="factorial(this.form,t.value)"></br>`
	`result:<input type="text" name="p"></br>`
	`</form>`
	`</body>`
	`</html>`
	
	ii) Write a Java script to find the prime number between 1 and 100.
	`<html>`
	`<head>`
	`<script language="javascript">`
	`var n=prompt("Enter User Value")`
	`var x=1;`
	`if(n==0 || n==1) x=0;`
	`for(i=2;i<n;i++)`
	`{`
	`if(n%i==0)`
	`{`
	`x=0;`
	`break;`
	`}`
	`}`
	`if(x==1)`

```
            {
              alert(n +" "+" is prime");
            }
            else
            {
              alert(n +" "+" is not prime");
            }

          </script>
    </head>
    <body>
    </html>
```

7. **Write a servlet program which displays the different content each time the user visits the page**

```
import java.io.*;
import java.sql.Date;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class PageHitCounter extends HttpServlet{

  private int hitCount;

  public void init()
  {
        hitCount = 0;
  }
  public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
          throws ServletException, IOException
  {
      // Set response content type
      response.setContentType("text/html");
      // This method executes whenever the servlet is hit
      // increment hitCount
      hitCount++;
      PrintWriter out = response.getWriter();
      String title = "Welcome User";
      String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
        "<html>\n" +
        "<head><title>" + title + "</title></head>\n" +
        "<body bgcolor=\"#f0f0f0\">\n" +
        "<h1 align=\"center\">" + title + "</h1>\n" +
        "<h2 align=\"center\">" + hitCount + "</h2>\n" +
        "</body></html>");
  }
  public void destroy()
  {
     // This is optional step but if you like you
     // can write hitCount value in your database.
  }
}
```

Now let us compile above servlet and create following entries in web.xml

```
....
 <servlet>
    <servlet-name>PageHitCounter</servlet-name>
    <servlet-class>PageHitCounter</servlet-class>
 </servlet>

 <servlet-mapping>
    <servlet-name>PageHitCounter</servlet-name>
    <url-pattern>/PageHitCounter</url-pattern>
 </servlet-mapping>
....
```

8. **Write a Java script program to create Popup box, alert and confirm box.**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Introduction to pop up box</title>
</head>
<body>
<p>Experiment with the popup boxes by clicking the buttons(OK and Cancel) on them</p>

    <script type="text/javascript">
     if(confirm("do you agree?"))
       alert("You have agreed");
     else
        input_text=prompt("Enter some string here..."," ");
     /*the value entered in prompt box is returned
     and stored in the variable text */
        alert("Hi "+input_text);
    </script>
</body>
```

```
</html>
```

9. **Write a Java script program to print the numbers from 0 to 50.**

```
<script type = "text/javascript">
var input;
var i=0;
input= 50;
while ( input > I )
{
document.write (i);
i++
}
</script>
```

b. Write a Java Script program to create table.

```
<html>
function createTable(a)
{
document.getElementById("tbl").innerHTML = "<table border = '1'>" +
        "<tr>" +
        "<td>1</td>" +
        "<td>abc</td>" +
        "<td>123</td>" +
        "</tr>" +
        "<tr>" +
        "<td>2</td>" +
        "<td>def</td>" +
        "<td>456</td>" +
        "</tr>" +
        "<table>";
}
</script>
<div id="cnt"></div>
<div id="tbl"></div>
<input type="text" name="txtRow" style="width:200px;height:25px;" /><button name="cmdRow"
style="width:125px;height:30px;" onclick="createTable()">Create Table</button>
</html>
```

10. **Write a Java script program to create user registration form.**

```
<!DOCTYPE html>
<html lang="en"><head>
<meta charset="utf-8">
<title>JavaScript Form Validation using a sample registration form</title>
<meta name="keywords" content="example, JavaScript Form Validation, Sample registration form" />
<meta name="description" content="This document is an example of JavaScript Form Validation using a sample
registration form. " />
```

```
<link rel='stylesheet' href='js-form-validation.css' type='text/css' />
<script src="sample-registration-form-validation.js"></script>
</head>
<body onload="document.registration.userid.focus();">
<h1>Registration Form</h1>
<p>Use tab keys to move from one input field to the next.</p>
<form name='registration' onSubmit="return formValidation();">
<ul>
<li><label for="userid">User id:</label></li>
<li><input type="text" name="userid" size="12" /></li>
<li><label for="passid">Password:</label></li>
<li><input type="password" name="passid" size="12" /></li>
<li><label for="username">Name:</label></li>
<li><input type="text" name="username" size="50" /></li>
<li><label for="address">Address:</label></li>
<li><input type="text" name="address" size="50" /></li>
<li><label for="country">Country:</label></li>
<li><select name="country">
<option selected="" value="Default">(Please select a country)</option>
<option value="AF">Australia</option>
<option value="AL">Canada</option>
<option value="DZ">India</option>
<option value="AS">Russia</option>
<option value="AD">USA</option>
</select></li>
<li><label for="zip">ZIP Code:</label></li>
<li><input type="text" name="zip" /></li>
<li><label for="email">Email:</label></li>
<li><input type="text" name="email" size="50" /></li>
<li><label id="gender">Sex:</label></li>
<li><input type="radio" name="msex" value="Male" /><span>Male</span></li>
<li><input type="radio" name="fsex" value="Female" /><span>Female</span></li>
<li><label>Language:</label></li>
<li><input type="checkbox" name="en" value="en" checked /><span>English</span></li>
<li><input type="checkbox" name="nonen" value="noen" /><span>Non English</span></li>
<li><label for="desc">About:</label></li>
<li><textarea name="desc" id="desc"></textarea></li>
<li><input type="submit" name="submit" value="Submit" /></li>
</ul>
</form>
</body>
</html>
```

11. **i) Explain any two validation function in java script.(4)**

JavaScript, provides a way to validate form's data on the client's computer before sending it to the web server. Form validation generally performs two functions.

- **Basic Validation -** First of all, the form must be checked to make sure data was entered into each form field that required it. This would need just loop through each field in the form and check for data.
- **Data Format Validation -** Secondly, the data that is entered must be checked for correct form and value. This would need to put more logic to test correctness of data.
  ```
  <input id="id1" type="number" min="100" max="300">
  ```

```
<button onclick="myFunction()">OK</button>

<p id="demo"></p>

<script>
function myFunction() {
   var inpObj = document.getElementById("id1");
   if (inpObj.checkValidity() == false) {
      document.getElementById("demo").innerHTML = inpObj.validationMessage;
   }
}
</script>
```

ii) Write a script to demonstrate the use of Date object.(6)

JavaScript Date Formats

A JavaScript date can be written as a string:

Sat Jun 13 2015 10:24:39 GMT+0530 (India Standard Time)

or as a number:

1434171279721

Dates written as numbers, specifies the number of milliseconds since January 1, 1970, 00:00:00.

Displaying Dates

```
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = Date();
</script>
```

Creating Date Objects

The Date object lets us work with dates.

A date consists of a year, a month, a day, an hour, a minute, a second, and milliseconds.

Date objects are created with the new Date () constructor.

There are 4 ways of initiating a date:

```
new Date()
new Date(milliseconds)
new Date(dateString)
new Date(year, month, day, hours, minutes, seconds, milliseconds)
```

Example:

```
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d;
</script>
```

iii)Write a java script program to generate Fibonacci series using do while loop.(6)

```
<html>
<body>
<script type="text/javascript">
var a=0,b=1,c;
document.write("Fibonacci");
```

	while (b<=10)
	{
	document.write(c);
	document.write("<br/>");
	c=a+b;
	a=b;
	b=c;
	}
	</script>
	</body>
	</html>

12.	**i) Explain JavaScript & document object model (DOM) with example.(8)**
	import java.io.File;
	import java.util.Scanner;
	import javax.xml.parsers.DocumentBuilder;
	import javax.xml.parsers.DocumentBuilderFactory;
	import org.w3c.dom.Document;
	import org.w3c.dom.Element;
	import org.w3c.dom.Node;
	import org.w3c.dom.NodeList;
	public class NewClass {
	public static void main(String args[]) {
	try {
	File stocks = new
	File("C:\\\\Users\\\\Administrator\\\\Documents\\\\NetBeansProjects\\\\WebApplication3\\\\newXMLDocument.xml");
	DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
	DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
	Document doc = dBuilder.parse(stocks);
	doc.getDocumentElement().normalize();
	System.out.println("root of xml file" + doc.getDocumentElement().getNodeName());
	NodeList nodes = doc.getElementsByTagName("stock");
	System.out.println("Enter the user id");
	Scanner scan=new Scanner(System.in);
	String s=scan.next();
	System.out.println("==========================");
	for (int i = 0; i < nodes.getLength(); i++) {
	Node node = nodes.item(i);
	if (node.getNodeType() == Node.ELEMENT_NODE) {
	Element element = (Element) node;
	if(s.equals(getValue("userid", element)))
	{
	System.out.println("Stock User Id: " + getValue("userid", element));
	System.out.println("Stock Symbol: " + getValue("symbol", element));

```
System.out.println("Stock Price: " + getValue("price", element));
System.out.println("Stock Quantity: " + getValue("quantity", element));
}
}
}
} catch (Exception ex) {
ex.printStackTrace();
}
}
private static String getValue(String tag, Element element) {
NodeList nodes = element.getElementsByTagName(tag).item(0).getChildNodes();
Node node = (Node) nodes.item(0);
return node.getNodeValue();
}
}
```

ii) Explain in details the JDBC CONNECTIVITY with example program.(8)

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
 public class DatabaseAccess extends HttpServlet{

  public void doGet(HttpServletRequest request, HttpServletResponse response)
           throws ServletException, IOException
  {
      // JDBC driver name and database URL
      static final String JDBC_DRIVER="com.mysql.jdbc.Driver";
      static final String DB_URL="jdbc:mysql://localhost/TEST";

      // Database credentials
      static final String USER = "root";
      static final String PASS = "password";

      // Set response content type
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String title = "Database Result";
      String docType =
        "<!doctype html public \"-//w3c//dtd html 4.0 " +
    "transitional//en\">\n";
    out.println(docType  +
    "<html>\n" +
    "<head><title>" + title + "</title></head>\n" +
    "<body bgcolor=\"#f0f0f0\">\n" +
    "<h1 align=\"center\">" + title + "</h1>\n");
  try{
    //          Register          JDBC          driver
    Class.forName("com.mysql.jdbc.Driver");

    // Open a connection
    conn = DriverManager.getConnection(DB_URL,USER,PASS);

    // Execute SQL query
    stmt = conn.createStatement();
    String sql;
    sql = "SELECT id, first, last, age FROM Employees";
    ResultSet rs = stmt.executeQuery(sql);

    // Extract data from result set
    while(rs.next()){
      //Retrieve by column name
      int id = rs.getInt("id");
      int age = rs.getInt("age");
      String first = rs.getString("first");
      String last = rs.getString("last");

      //Display values
      out.println("ID: " + id + "<br>");
      out.println(", Age: " + age + "<br>");
      out.println(", First: " + first + "<br>");
      out.println(", Last: " + last + "<br>");
    }
    out.println("</body></html>");

    // Clean-up environment
    rs.close();
    stmt.close();
    conn.close();
  }catch(SQLException se){
        //Handle errors for JDBC
        se.printStackTrace();
      }catch(Exception e){
        //Handle errors for Class.forName
        e.printStackTrace();
      }finally{
        //finally block used to close resources
        try{
          if(stmt!=null)
            stmt.close();
        }catch(SQLException se2){
        }// nothing we can do
        try{
          if(conn!=null)
          conn.close();
        }catch(SQLException se){
          se.printStackTrace();
        }//end finally try
      } //end try
    }
  }
```

13. **Explain the JDBC database access in detail. Write a java servlet to conduct online examination. (APR/MAY 2013)**

Index.html

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head><title>Database Test</title></head>
<body>
<center>
<h1>Online Examination</h1>
</center>
<form action="Servlet_tier1" method="POST">
<!—SERVLET NAME IN ACTION ATTRIBUTE -->
<div align="left"><br></div>
<b>Seat Number:</b> <input type="text" name="Seat_no">
<div align="Right">
```

```
<b>Name:</b> <input type="text" name="Name" size="50"><br>
</div>
<br><br>
<b>1. Every host implements transport layer.</b><br/>
<input type="radio" name="group1" value="True">True
<input type="radio" name="group1" value="False">False<br>
<b>2. It is a network layer's responsibility to forward packets reliably from source to destination</b><br/>
<input type="radio" name="group2" value="True">True
<input type="radio" name="group2" value="False">False<br>
<b>3. Packet switching is more useful in bursty traffic</b><br/>
<input type="radio" name="group3" value="True">True
<input type="radio" name="group3" value="False">False<br>
<b>4. A phone network uses packet switching</b><br/>
<input type="radio" name="group4" value="True">True
<input type="radio" name="group4" value="False">False<br>
<b>5. HTML1 is a Protocol for describing web contents</b><br/>
<input type="radio" name="group5" value="True">True
<input type="radio" name="group5" value="False">False<br>
<br><br><br>
<center>
<input type="submit" value="Submit"><br><br>
</center>
</form>
</body>
</html>
```

SERVLET_TIER1.JAVA:

```
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
@WebServlet(name="Servlet_tier1", urlPatterns={"/Servlet_tier1"})
public class Servlet_tier1 extends HttpServlet
```

```
{
String message,Seat_no,Name,ans1,ans2,ans3,ans4,ans5;
int Total=0;
java.sql.Connection connect;
java.sql.Statement stmt=null;
java.sql.ResultSet rs=null;
public void doPost(HttpServletRequest request,HttpServletResponse response) throws
ServletException,IOException
{
try
{
String url="jdbc:odbc:NEO";// Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
connect=DriverManager.getConnection(url,"","");
message="Thank you for participating in online Exam";
}
catch(ClassNotFoundException cnfex){
cnfex.printStackTrace();
}
catch(SQLException sqlex){
sqlex.printStackTrace();
}
catch(Exception excp){
   excp.printStackTrace();
}
Seat_no=request.getParameter("Seat_no");
Name=request.getParameter("Name");
ans1=request.getParameter("group1");
ans2=request.getParameter("group2");
ans3=request.getParameter("group3");
ans4=request.getParameter("group4");
ans5=request.getParameter("group5");
if(ans1.equals("True"))
Total+=2;
```

```
if(ans2.equals("False"))
Total+=2;
if(ans3.equals("True"))
Total+=2;
if(ans4.equals("False"))
Total+=2;
if(ans5.equals("False"))
Total+=2;
try
{
stmt=connect.createStatement();
String query="INSERT INTO
student("+"Seat_no,Name,Total"+")VALUES('"+Seat_no+"','"+Name+"','"+Total+"')";
stmt.executeUpdate(query);
stmt.close();
}catch(SQLException ex){
}
response.setContentType("text/html");
PrintWriter out=response.getWriter();
out.println("<html>");
out.println("<head>");
out.println("</head>");
out.println("<body bgcolor=cyan>");
out.println("<center>");
out.println("<h1>"+message+"</h1>\n");
out.println("<h3>Yours results stored in our database</h3>");
out.print("<br><br>");
out.println("<b>"+"Participants and their Marks"+"</b>");
out.println("<table border=5>");
try
{
java.sql.Statement stmt2=connect.createStatement();
String query="SELECT * FROM student";
```

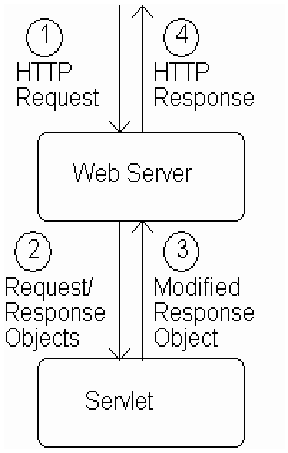
	`rs=stmt2.executeQuery(query);`
	`out.println("<th>"+"Seat_no"+"</th>");`
	`out.println("<th>"+"Name"+"</th>");`
	`out.println("<th>"+"Marks"+"</th>");`
	`while(rs.next())`
	`{`
	`out.println("<tr>");`
	`out.print("<td>"+rs.getString(1)+"</td>");`
	`out.print("<td>"+rs.getString(2)+"</td>");`
	`out.print("<td>"+rs.getString(3)+"</td>");`
	`out.println("</tr>");`
	`}`
	`out.println("</table>");`
	`}`
	`catch(SQLException ex){ }`
	`finally`
	`{`
	`try`
	`{`
	`if(rs!=null)`
	`rs.close();`
	`if(stmt!=null)`
	`stmt.close();`
	`if(connect!=null)`
	`connect.close();`
	`}catch(SQLException e){ }`
	`}`
	`out.println("</center>");`
	`out.println("</body></html>");`
	`Total=0;`
	`} }`

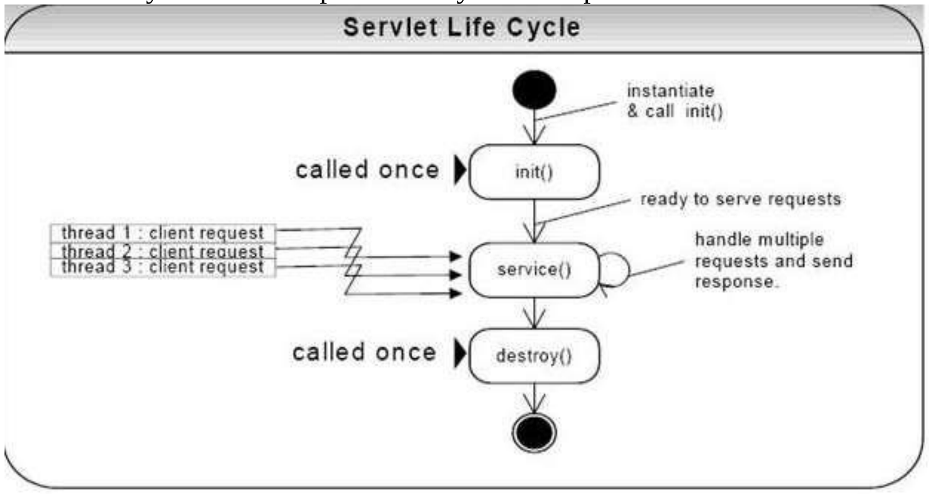
14.	**What is a servlet? Explain briefly the Servlet life cycle and Servlet HTTP package?** A servlet is a Java programming language class that is used to extend the capabilities of servers that host

applications accessed by means of a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers. For such applications, Java Servlet technology defines HTTP-specific servlet classes.

The `javax.servlet` and `javax.servlet.http` packages provide interfaces and classes for writing servlets. All servlets must implement the `Servlet` interface, which defines life-cycle methods. When implementing a generic service, you can use or extend the `GenericServlet` class provided with the Java Servlet API. The`HttpServlet` class provides methods, such as `doGet` and `doPost`, for handling HTTP-specific services.

  - Static: HTML document is retrieved from the file system and returned to the client
  - Dynamic: HTML document is generated by a program in response to an HTTP request
- Java servlets are one technology for producing dynamic server responses
  - Servlet is a class instantiated by the server to produce a dynamic response




Servlet Life Cycle
instantiate & call init()
called once
init()
ready to serve requests
thread 1 : client request
thread 2 : client request
thread 3 : client request
service()
handle multiple requests and send response.
called once
destroy()

What are all the Servlet API life cycle methods
Servlet API life cycle methods
- init(): called when servlet is instantiated; must return before any other methods will be called
- service(): method called directly by server when an HTTP request is received; default service() method calls doGet() (or related methods covered later)
- destroy(): called when server shuts down

PARAMETER DATA:
- The request object (which implements HttpServletRequest) provides information from the HTTP request to the servlet
- One type of information is parameter data, which is information from the query string portion of the HTTP request

query string with one parameter

```
http://www.example.com/servlet/PrintThis?arg=aString
```

parameter name: arg
parameter value: aString

GET vs. POST method for forms:
- GET:
  - Query string is part of URL
  - Length of query string may be limited

- Recommended when parameter data is not stored but used only to request information (*e.g.*, search engine query)
- POST:
  - Query string is sent as body of HTTP request
  - Length of query string is unlimited
  - Recommended if parameter data is intended to cause the server to update stored data
  - Most browsers will warn you if they are about to resubmit POST data to avoid duplicate updates

15. **List out the classes and interfaces available in javax.servlet.http package?**

The javax.servlet.http package contains a number of classes and interfaces that describe and define the contracts between a servlet class running under the HTTP protocol and the runtime environment provided for an instance of such a class by a conforming servlet container.

Interface:

HttpServletRequest	Extends the ServletRequest interface to provide request information for HTTP servlets.
HttpServletResponse	Extends the ServletResponse interface to provide HTTP-specific functionality in sending a response.
HttpSession	Provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.
HttpSessionActivationListener	Objects that are bound to a session may listen to container events notifying them that sessions will be passivated and that session will be activated.
HttpSessionAttributeListener	This listener interface can be implemented in order to get notifications of changes to the attribute lists of sessions within this web application.
HttpSessionBindingListener	Causes an object to be notified when it is bound to or unbound from a session.
HttpSessionContext	Deprecated. As of Java(tm) Servlet API 2.1 for security reasons, with no replacement.
HttpSessionListener	Implementations of this interface are notified of changes to the list of active sessions in a web application.

Class:

Cookie	Creates a cookie, a small amount of information sent by a servlet to a Web browser, saved by the browser, and later sent back to the server.
HttpServlet	Provides an abstract class to be subclassed to create an HTTP servlet suitable for a Web site.
HttpServletRequestWrapper	Provides a convenient implementation of the HttpServletRequest interface that can be subclassed by developers wishing to adapt the request to a Servlet.
HttpServletResponseWrapper	Provides a convenient implementation of the HttpServletResponse interface that can be subclassed by developers wishing to adapt the response from a Servlet.
HttpSessionBindingEvent	Events of this type are either sent to an object that

		implements HttpSessionBindingListener when it is bound or unbound from a session, or to aHttpSessionAttributeListener that has been configured in the deployment descriptor when any attribute is bound, unbound or replaced in a session.
	HttpSessionEvent	This is the class representing event notifications for changes to sessions within a web application.
	HttpUtils	Deprecated. As of Java(tm) Servlet API 2.3.

16. **Write short notes on the following servlet classes**
GenericServlet, ServletInputStream, ServletOutputStream and ServletException

GenericServlet:

GenericServlet class implements Servlet, ServletConfig andSerializable interfaces. It provides the implementaion of all the methods of these interfaces except the service method.

GenericServlet class can handle any type of request so it is protocol-independent.

You may create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.

Methods of GenericServlet class

There are many methods in GenericServlet class. They are as follows:

1. public void init(ServletConfig config) is used to initialize the servlet.
2. public abstract void service(ServletRequest request, ServletResponse response) provides service for the incoming request. It is invoked at each time when user requests for a servlet.
3. public void destroy() is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
4. public ServletConfig getServletConfig() returns the object of ServletConfig.
5. public String getServletInfo() returns information about servlet such as writer, copyright, version etc.
6. public void init() it is a convenient method for the servlet programmers, now there is no need to call super.init(config)
7. public ServletContext getServletContext() returns the object of ServletContext.
8. public String getInitParameter(String name) returns the parameter value for the given parameter name.
9. public Enumeration getInitParameterNames() returns all the parameters defined in the web.xml file.
10. public String getServletName() returns the name of the servlet object.
11. public void log(String msg) writes the given message in the servlet log file.
12. public void log(String msg,Throwable t) writes the explanatory message in the servlet log file and a stack trace.

ServletInputStream:

ServletInputStream class provides stream to read binary data such as image etc. from the request object. It is an abstract class.

The getInputStream() method of ServletRequest interface returns the instance of ServletInputStream class. So can be get as:

ServletInputStream sin=request.getInputStream();

Method of ServletInputStream class

There are only one method defined in the ServletInputStream class.

int readLine(byte[] b, int off, int len) it reads the input stream.

ServletOutputStream:

ServletOutputStream class provides a stream to write binary data into the response. It is an abstract class.

The getOutputStream() method of ServletResponse interface returns the instance of ServletOutputStream class. It may be get as:

ServletOutputStream out=response.getOutputStream();

Methods of ServletOutputStream class

The ServletOutputStream class provides print() and println() methods that are overloaded.

1. void print(boolean b){}
2. void print(char c){}
3. void print(int i){}
4. void print(long l){}
5. void print(float f){}
6. void print(double d){}
7. void print(String s){}
8. void println{}
9. void println(boolean b){}
10. void println(char c){}
11. void println(int i){}
12. void println(long l){}
13. void println(float f){}
14. void println(double d){}
15. void println(String s){}
16. public **ServletException**(java.lang.Throwable rootCause)

ServletException:

public ServletException(java.lang.String message,java.lang.Throwable rootCause)

Constructs a new servlet exception when the servlet needs to throw an exception and include a message about the "root cause" exception that interfered with its normal operation, including a description message.

Parameters:

message - a String containing the text of the exception message

rootCause - the Throwable exception that interfered with the servlet's normal operation, making this servlet exception necessary

Constructs a new servlet exception when the servlet needs to throw an exception and include a message about the "root cause" exception that interfered with its normal operation. The exception's message is based on the localized message of the underlying exception.

This method calls the getLocalizedMessage method on the Throwable exception to get a localized exception message. When subclassing ServletException, this method can be overridden to create an exception message designed for a specific locale.

Parameters:

	rootCause - the Throwable exception that interfered with the servlet's normal operation, making the servlet exception necessary

17.	**Write a servlet program which displays the different image each time the user visits the page and the images are links**
	`package com.javatpoint;`
	`import java.io.*;`
	`import javax.servlet.*;`
	`import javax.servlet.http.*;`
	`public class DisplayImage extends HttpServlet {`
	
	`    public void doGet(HttpServletRequest request,HttpServletResponse response)`
	`            throws IOException`
	`    {`
	`    response.setContentType("image/jpeg");`
	`    ServletOutputStream out;`
	`    out = response.getOutputStream();`
	`    FileInputStream fin = new FileInputStream("c:\\test\\java.jpg");`
	
	`    BufferedInputStream bin = new BufferedInputStream(fin);`
	`    BufferedOutputStream bout = new BufferedOutputStream(out);`
	`    int ch =0; ;`
	`    while((ch=bin.read())!=-1)`
	`    {`
	`    bout.write(ch);`
	`    }`
	
	`    bin.close();`
	`    fin.close();`
	`    bout.close();`
	`    out.close();`
	`    } }`

18.	**Explain in detail about Servlet Database Connectivity with an example of Student database.**
	a. Write a servlet for creating Student Entry form.
	Index.html
	
	`<html>`
	`<head>`
	`<title>Student Registrion index page </title>`
	`</head>`
	`<body>`
	`<center>`
	`<form action="InsertRecord" method="post">`
	` <fieldset><legend><font face="Courier New" size="+1"`
	` color="red">Student Registration</font></legend>`
	`   <table>`
	
	`   <tr>`
	`      <td>Student Name &nbsp;&nbsp; :</td>`

```
        <td> <input type="text" name="name"> </td>
    </tr>
    <tr>
        <td>Enrollment Number &nbsp;&nbsp; :</td>
        <td> <input type="text" name="enrolmentNo"> </td>
    </tr>

    <tr>
        <td>Program Name &nbsp;&nbsp; : </td>
        <td> <input type="text" name="program"> </td>
    </tr>
    <tr>
        <td>Gender &nbsp;&nbsp; : </td>
        <td><select name="gender">
            <option value="male"> Male</option>
            <option value="female"> Female</option>

            </select>
        </td>

    </tr>


    <tr>
        <td>Address &nbsp;&nbsp; :</td>
        <td> <textarea cols="20" rows="2" name="address">
            </textarea> </td>
    </tr>

    <tr>
        <td colspan=2>
            <input type="submit" value="Register"> </td>
    </tr>

    </table>
</fieldset>

</form>
</center>
</body>
</html>
```

<u>InsertRecord.java</u>

```
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class InsertRecord extends HttpServlet {
  public void doPost(HttpServletRequest request,
  HttpServletResponse response)
    throws ServletException, IOException {

  response.setContentType("text/html");
  PrintWriter out = response.getWriter();

  String name=request.getParameter("name");
  String en=request.getParameter("enrolmentNo");
  String program=request.getParameter("program");
  String gender=request.getParameter("gender");
  String address=request.getParameter("address");
  int id=0;
  int enrol=0;


  if(name.equals("") || en.equals("") ||
    program.equals("") || gender.equals("") ||
    address.equals(""))
  {
    out.println("Please insert valid data");
    RequestDispatcher rd =
        request.getRequestDispatcher("/index.html");
    rd.include(request, response);

  }
  else
  {
    enrol=Integer.parseInt(en);

  try{
    Class.forName("oracle.jdbc.driver.OracleDriver");
    Connection con=DriverManager.getConnection(
    "jdbc:oracle:thin:@localhost:1521:xe","system","sunil");

    PreparedStatement pst=con.prepareStatement(
    "SELECT id FROM STUDENT");
    ResultSet rs=pst.executeQuery();
    while(rs.next())
    {
      id=rs.getInt(1);
    }

    PreparedStatement ps=con.prepareStatement(
    "insert into STUDENT values(?,?,?,?,?,?)");

    ps.setInt(1,id+1);
    ps.setString(2,name);
    ps.setInt(3,enrol);
    ps.setString(4,program);
    ps.setString(5,gender);
    ps.setString(6,address);

    int i=ps.executeUpdate();
    if(i>0)
    out.print("Student record successfully inserted");
    out.print("<BR>");
    out.print("Insert another record ...");
    RequestDispatcher
      rd = request.getRequestDispatcher("/index.html");
    rd.include(request, response);
    }
    catch (Exception e) {
      System.out.println(e);
    }
  }
  out.close();
  }

}
```

<u>Web.html</u>

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.5">
```

	`<display-name>Student</display-name>`
	
	`<servlet>`
	`  <description></description>`
	`  <display-name>InsertRecord</display-name>`
	`  <servlet-name>InsertRecord</servlet-name>`
	`  <servlet-class>InsertRecord</servlet-class>`
	`</servlet>`
	`  <servlet-mapping>`
	`  <servlet-name>InsertRecord</servlet-name>`
	`  <url-pattern>/InsertRecord</url-pattern>`
	`</servlet-mapping>`
	`  <welcome-file-list>`
	`  <welcome-file>index.html</welcome-file>`
	`</welcome-file-list>`
	`</web-app>`

19.	**Explain in detail about JSP with an example of current date and simple message.**
	`<%@page contentType="text/html" import="java.util.*" %>`
	`<html>`
	`<body>`
	`<p>&nbsp;</p>`
	`<div align="center">`
	`<center>`
	`<table border="0" cellpadding="0" cellspacing="0" width="460" bgcolor="#EEFFCA">`
	`<tr>`
	`<td width="100%"><font size="6" color="#008000">&nbsp;Date Example</font></td>`
	`</tr>`
	`<tr>`
	`<td width="100%"><b>&nbsp;Current Date and time is:&nbsp; <font color="#FF0000">`
	`<%= new java.util.Date() %>`
	`</font></b></td>`
	`</tr>`
	`</table>`
	`</center>`
	`</div>`
	`</body>`
	`</html>`

20.	**Discuss in detail about Action elements in JSP with an example of display current time and color.** **ACTION:** • Standard: provided by JSP itself • Custom: provided by a tag library such as JSTL. • **JSTL** is divided into several functional areas, each with its own namespace:

TABLE 8.6: JSTL functional areas.

Functional Area	Namespace Name Suffix
Core	core
XML Processing	xml
Functions	functions
Database	sql
Internationalization	fmt

Namespace prefix is
`http://java.sun.com/jsp/jstl/`

JSTL CORE ACTIONS:

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

TABLE 8.7: Some JSTL core actions.

Action	Purpose
set	Assign a value to a scoped variable, creating the variable if necessary
remove	Destroy a scoped variable
out	Write data to out implicit object, escaping XML special characters
url	Create a URL with query string
if	Conditional (if-then) processing
choose	Conditional (if-then-elseif) processing
forEach	Iterate over a collection of items

- Common variables:
  - var
    - Represents name of a scoped variable that is assigned to by the action
    - Must be a string literal, not an EL expression
  - scope
    - Specifies scope of scoped variable as one of the literals page, request, session, or application

set action
- Setting (and creating) a scoped variable

out action
- Normally used to write a string to the out JSP implicit object
- Automatically escapes XML special characters

if action
- General form includes scoped variable to receive test value

```
<c:set var="age" value="20" scope="session"></c:set>

<c:if test="${age ge 18}" var="x">
<h3><c:out value="WELCOME"></c:out></h3>
</c:if>

<h3><c:out value="${x}"></c:out></h3>

<c:if test="${x}">
   <h3><c:out value="WELCOME"></c:out></h3>
</c:if>
```

Output:

```
WELCOME
true
```

WELCOME

remove action

- Only attributes are var and scope
- Removes reference to the specified scoped variable from the scope object

```
<c:set var="x" value="10" scope="session"></c:set>
<c:set var="y" value="20" scope="session"></c:set>
<h3>Product :<c:out value="${x*y}"></c:out></h3>
<c:remove var="x" scope="session"/>
<c:remove var="y" scope="session"/>
<h3>Product :<c:out value="${x*y}"></c:out></h3>
```

Output:

```
Product :200

Product :0
```

choose action:

```
<c:set var="salary" scope="session" value="5000"/>
      <p>Your salary is : <c:out value="${salary}"/></p>
<c:choose>
   <c:when test="${salary > 1000}">
      Salary is very good.
   </c:when>
   <c:otherwise>
     Salary is very low
   </c:otherwise>
</c:choose>
```

Output:

```
Your salary is : 5000
Salary is very good.
```

forEach action:

Used to increment a variable (writes 2, 4, 6, 8 to the out object)

```
<c:forEach begin="1" end="10" step="4" var="x">
   Begin Index value :${x}<br>
   </c:forEach>
```

Output:

```
Begin Index value :1
Begin Index value :5
Begin Index value :9
```

url action

- value attribute is a URL to be written to the out JSP implicit object
- URL's beginning with / are assumed relative to context path
- param elements can be used to define parameters that will be URL encoded

curl.jsp:

```
<c:url value="/URLTest.jsp" var="x" >
     <c:param name="d1" value="SCJP"/>
     <c:param name ="d2" value="SCWCD"/>
  </c:url>
<h1>The modified url : ${x},</h1><br>
```

```
<a href="${x}">click Here </a>
Output:
The modified url : /JSTL/URLTest.jsp?d1=SCJP&d2=SCWCD,
URLTest.jsp
        <h2> This is URLTest JSP </h2>
        <h2>First Parameter : : :${param.d1}</h2>
        <h2>Second Parameter : : :${param.d2}</h2>

Output:
URL:
This is URLTest  JSP
First Parameter : : :SCJP
Second Parameter : : :SCWCD
```

21. **Explain about JSP object in detail.**

JSP Implicit Objects are the Java objects that the JSP Container makes available to developers in each page and developer can call them directly without being explicitly declared. JSP Implicit Objects are also called pre-defined variables.

JSP supports nine Implicit Objects which are listed below:

Object	Description
request	This is the HttpServletRequest object associated with the request.
response	This is the HttpServletResponse object associated with the response to the client.
out	This is the PrintWriter object used to send output to the client.
session	This is the HttpSession object associated with the request.
application	This is the ServletContext object associated with application context.
config	This is the ServletConfig object associated with the page.
pageContext	This encapsulates use of server-specific features like higher performance JspWriters.
page	This is simply a synonym for this, and is used to call the methods defined by the
Exception	The Exception object allows the exception data to be accessed by designated JSP.

The request Object:

The request object is an instance of a javax.servlet.http.HttpServletRequest object. Each time a client requests a page the JSP engine creates a new object to represent that request.

The request object provides methods to get HTTP header information including form data, cookies, HTTP methods etc.

We would see complete set of methods associated with request object in coming chapter: JSP - Client Request.

The response Object:

The response object is an instance of a javax.servlet.http.HttpServletResponse object. Just as the server creates the request object, it also creates an object to represent the response to the client.
The response object also defines the interfaces that deal with creating new HTTP headers. Through this object the JSP programmer can add new cookies or date stamps, HTTP status codes etc.
We would see complete set of methods associated with response object in coming chapter: JSP - Server Response.
The out Object:
The out implicit object is an instance of a javax.servlet.jsp.JspWriter object and is used to send content in a response.
The initial JspWriter object is instantiated differently depending on whether the page is buffered or not. Buffering can be easily turned off by using the buffered='false' attribute of the page directive.
The JspWriter object contains most of the same methods as the java.io.PrintWriter class. However, JspWriter has some additional methods designed to deal with buffering. Unlike the PrintWriter object, JspWriter throws IOExceptions.
Following are the important methods which we would use to write boolean char, int, double, object, String etc.

Method	Description
out.print(dataType dt)	Print a data type value
out.println(dataType dt)	Print a data type value then terminate the line with new line character.
out.flush()	Flush the stream.

The session Object:
The session object is an instance of javax.servlet.http.HttpSession and behaves exactly the same way that session objects behave under Java Servlets.
The session object is used to track client session between client requests. We would see complete usage of session object in coming chapter: JSP - Session Tracking.
The application Object:
The application object is direct wrapper around the ServletContext object for the generated Servlet and in reality an instance of a javax.servlet.ServletContext object.
This object is a representation of the JSP page through its entire lifecycle. This object is created when the JSP page is initialized and will be removed when the JSP page is removed by the jspDestroy() method.
By adding an attribute to application, you can ensure that all JSP files that make up your web application have access to it.
You can check a simple use of Application Object in chapter: JSP - Hits Counter
The config Object:
The config object is an instantiation of javax.servlet.ServletConfig and is a direct wrapper around the ServletConfig object for the generated servlet.
This object allows the JSP programmer access to the Servlet or JSP engine initialization parameters such as the paths or file locations etc.
The following config method is the only one you might ever use, and its usage is trivial:

```
config.getServletName();
```
This returns the servlet name, which is the string contained in the <servlet-name> element defined in the WEB-INF\web.xml file

The pageContext Object:
The pageContext object is an instance of a javax.servlet.jsp.PageContext object. The pageContext object is used to represent the entire JSP page.
This object is intended as a means to access information about the page while avoiding most of the implementation details.
This object stores references to the request and response objects for each request. The application, config, session, and out objects are derived by accessing attributes of this object.
The pageContext object also contains information about the directives issued to the JSP page, including the buffering information, the errorPageURL, and page scope.
The PageContext class defines several fields, including PAGE_SCOPE, REQUEST_SCOPE, SESSION_SCOPE, and APPLICATION_SCOPE, which identify the four scopes. It also supports more than 40 methods, about half of which are inherited from the javax.servlet.jsp. JspContext class.
One of the important methods is removeAttribute, which accepts either one or two arguments. For example, pageContext.removeAttribute ("attrName") removes the attribute from all scopes, while the following code only removes it from the page scope:

pageContext.removeAttribute("attrName", PAGE_SCOPE);
You can check a very good usage of pageContext in coming chapter: JSP - File Uploading.
The page Object:
This object is an actual reference to the instance of the page. It can be thought of as an object that represents the entire JSP page.
The page object is really a direct synonym for the this object.
The exception Object:
The exception object is a wrapper containing the exception thrown from the previous page. It is typically used to generate an appropriate response to the error condition.

Unit – IV

Part - A

1. What is PHP?
PHP - Hypertext Preprocessor -one of the most popular server-side scripting languages for creating dynamic Web pages.
- an open-source technology
- platform independent

2. List the data types used in PHP.

Data types	**Description**
Integer	Whole numbers (i.e., numbers without a decimal point)
Double	Real numbers (i.e., numbers containing a decimal point)
String	Text enclosed in either single ('') or double ("") quotes.
Boolean	True or false
Array	Group of elements of the same type
Object	Group of associated data and methods
Resource	An external data source

3. How type conversion is done in PHP?
In PHP, data-type conversion can be performed by passing the data type as an argument to function settype.
Function settype takes two arguments: The variable whose data type is to be changed and the variable 's new data type.
E.g., settype($testString, "double");

4. Write the uses of text manipulation with regular expression in PHP.
- PHP processes text data easily and efficiently, enabling straightforward searching, substitution, extraction and concatenation of strings.

	• Text manipulation in PHP is usually done with regular expressions — a series of characters that serve as pattern-matching templates (or search criteria) in strings, text files and databases. • This feature allows complex searching and string processing to be performed using relatively simple expressions
5.	**List the important characteristics of PHP.** The main characteristics of PHP are: • PHP is web-specific and open source • Scripts are embedded into static HTML files • Fast execution of scripts • Fast access to the database tier of applications • Supported by most web servers and operating systems • Supports many standard network protocols libraries available for IMAP, NNTP, SMTP, POP3 • Supports many database management systems libraries available for UNIX DBM, MySQL, Oracle, • Dynamic Output any text, HTML XHTML and any other XML file. • Also Dynamic Output images, PDF files and even Flash m ovies • Text processing features, from the POSIX Extended or Perl regular expressions to parsing XML documents. • A fully featured programming language suitable for complex systems development
6.	**How to Include PHP in a Web Page?** There are 4 ways of including PHP in a web page 1. <?php echo("Hello world"); ?> 2. <script language = "php"> echo("Hello world"); </script> 3. <? echo("Hello world"); ?> 4. <% echo("Hello world"); %> we can also use print instead of echo • Method (1) is clear and unambiguous • Method (2) is useful in environments supporting mixed scripting languages in the same HTML file • Methods (3) and (4) depend on the server configuration

7.	**Write a simple PHP Script.**
	Here is PHP script which is embedded in HTML using level one header with the PHP output text. The name of this file is called hello.php.
	`<html>`
	`  <head>`
	`    <title>Hello world</title>`
	`  </head>`
	`  <body>`
	`    <h1><?php echo("Hello world"); ?></h1>`
	`    <h1><?php print("This prints the same thing!");?></h1>`
	`  </body>`
	`<html>`

8.	**How do you include comments in PHP?** PHP supports three types of comments: 1. Shell style comments - denoted #THIS IS A COMMENT 2. C++ style comments - denoted THIS IS A COMMENT— 3. C style comments - denoted /* ALL THIS COMMENTED! */
9.	**What are variables in PHP?** Variables start with the $ symbol. E.g.: $myInteger = 3; $myString = "Hello world";

	$myFloat = 3.145;
10.	**How do you declare a variable using PHP data types?** Data types are not explicitly defined: • Variable type is determined by assignment • Strings can be defined with single (') and double (") quotes. • PHP has a Boolean type: Defined as false – An integer or float value of 0 or – The keyword false – The empty string ‘‘’’ or the string ‘‘0’’ – An empty array or object – The NULL value Defined as true – Any non-zero integer or float value – The keyword true • Standard operators with standard syntax applied to variables
11.	**How do you declare and initialize an array in PHP?** Two ways of declaring and initializing an array: a) Individual element initialization in an array $myArray[0]= "Apples"; $myArray[1]= "Bananas"; b) Arrays can be constructed using the array() keyword $person = array("Dave", "Adam", "Ralph");
12.	**What are associative arrays in PHP?** $myArray["Monday"]= "Apples"; $myArray["Tuesday"]= "Bananas"; Associative Arrays can also be constructed using the array( ) keyword. $food = array("Monday"=>"Apples","Tuesday"=> "Bananas"); The symbol => delimits the hash name from the hash value.
13.	**What is the scope of variables in PHP?** Once PHP variables have been defined they are known for the rest of the Web page: • Obeying standard scoping rules of course. • Variables can be local to functions etc, much like any languages.
14.	**List some built in functions in PHP.** Mathematical functions:- abs, ceil, cos, log, min, rand, sqrt File handling:- fopen, flock, feof, fgets, fputs, fclose

15.	**List the PHP standard Flow-controls statements**
	if,
	if/else
	switch
	while
	for

16.	$a=3;
	Function what()
	{
	++$a;
	echo "a=$a\n";
	}
	what();
	echo "a=$a\n";

	What is the output? **1 3**
17.	**List the functions to create a pattern.** Preg_match, Preg_matchall, Preg_replace, Preg_split

18.	**Write a PHP script to set the background colour to blue on Tuesday in a given date.**
	`<?php`
	`  if(date("D") == "Tue") $colour = "blue";`
	` else $colour = "red";`
	`?>`
	`<html>`
	`  <head>`
	`     <title>Welcome</title>`
	`  </head>`
	`  <body bgcolor = <?php echo($colour) ?>>`
	`      <h1>Welcome</h1>`
	`  </body>`
	`</html>`

19.	**What is cookie? Give example in PHP**
	A cookie is a text string stored on the client machine by your script (to track users and manage transactions). Cookies are automatically returned (by the client), and can be accessed using a variable of the same name
	• The following script reads and displays a cookie, and sets it with a new value (string) that was passed to the script as a parameter.
	• The cookie will expire after 20 minutes (1200 seconds)
	`<?php setCookie("CookieTest", $val, time()+1200); ?>`
	`<html>`
	`   <head><title>Welcome</title></head>`
	`     <body>`
	`        <?php echo("<h2>The cookie is: $CookieTest</h1>`
	`     </body>`
	`</html>`

20.	**What is XML ?** Extensible markup language. It offer a standard, flexible and inherently extensible data format, XML significantly reduces the burden of deploying the many technologies needed to ensure the success of Web services.
21.	**Define XML attributes** • XML elements can have attributes in the start tag, just like HTML. • Attributes are used to provide additional information about elements. • Attributes cannot contain multiple values (child elements can) • Attributes are not easily expandable (for future changes)
22.	**Write the main difference between XML and HTML.** ***Main Difference between XML and HTML*** XML was designed to carry data. XML is not a replacement for HTML. XML and HTML were designed with different goals: XML was designed to describe data and to focus on what data is. HTML was designed to display data and to focus on how data looks. HTML is about displaying information, while XML is about describing information
23.	**What is meant by a XML namespace? (APR/MAY 2011)**

XML Namespaces provide a method to avoid element name conflicts. When using prefixes in XML, a so-called **namespace** for the prefix must be defined. The namespace is defined by the **xmlns attribute** in the start tag of an element. The namespace declaration has the following syntax. **xmlns:*prefix*="*URI*".**

```
<root> <h:table xmlns:h="http://www.w3.org/TR/html4/">
<h:tr>
<h:td>Apples</h:td>
<h:td>Bananas</h:td>
</h:tr> </h:table>
<f:table xmlns:f="http://www.w3schools.com/furniture">
<f:name>African Coffee Table</f:name>
<f:width>80</f:width>
 <f:length>120</f:length> </f:table> </root>
```

24. What is XML namespace? (NOV/DEC 2012)

XML allows document authors to create custom elements.

- This extensibility can result in naming collisions (i.e. different elements that have the same name) among elements in an XML document.

An XML namespace is a collection of element and attribute names. Each namespace has a unique name that provides a means for document authors to unambiguously refer to elements with the same name (i.e. prevent collisions).

25. What is the purpose of namespace? (MAY/JUNE 2014)

XML Namespaces provide a method to avoid element name conflicts. In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications.

26. Compare DOM and SAX in XML processing. (MAY/JUNE 2013)

DOM	**SAX**
DOM is an interface-oriented Application Programming Interface.	SAX parser works incrementally and generates events that are passed to the application.
It allows for navigation of the entire document.	DOM parser reads the whole XML document and returns a DOM tree representation of xml document.
DOM allows you to read and write.	SAX is essentially an API for reading XML

27. What are complex types?

complex types are an important aspects of xml schema that allow application developers to define application-specific data types that can be checked by programs that check XML document for validity. XML schema divides complex types into two categories: those with *simple content* & those with *complex content.*

28. What are all the Transformation techniques?

- XSLT - it is an XML- based languages used to transform XML documents into others format such as HTML for web display.
- XLINK - highlighting that element or taking the user directly to that point in the document.
- XPATH - xpath gets its name from its use of a payh notation to navigate through the hierarchical tree structure of an XML document
- XQUERY - it is W3C initiative to define a standard set of constructs for querying & searching XML document.

29. What is XSLT?

- XSLT stands for XSL Transformations
- XSLT is the most important part of XSL
- XSLT transforms an XML document into another XML document
- XSLT uses XPath to navigate in XML documents

XSLT is a W3C Recommendation

30. Define the term DTD.

	A Document Type Definition (DTD) defines the legal building blocks of an XML document. It defines the document structure with a list of legal elements and attributes.
31.	**List two types of DTD declaration** DTD is stands for Document Type Definition which is used to structure the XML document. The type of DTD are as follows i) Internal Declaration ii) External Declaration.
32.	**How to declare DTD attributes?** An attribute declaration has the following syntax: <!ATTLIST element-name attribute-name attribute-type default-value> DTD example: <!ATTLIST payment type CDATA "check"> XML example: <payment type="check" />
33.	**What is XML schema?** An XML schema is itself an XML document. It provides more detail about the kind of data that can appear as part of an XML document.
34.	**What is the purpose of XML schema? (APR/MAY 2013)** • The schemas are more specific and provide the support for data types. • The schema is aware of namespace • The XML Schema is written in XML itself and has a large number of built-in and derived types. • The xml schema is the W3C recommendation. Hence it is supported by various XML validator and XML Processors.
35.	**What are the disadvantages of schema?** • The XML schema is complex to design and hard to learn • The XML document cannot be if the corresponding schema file is absent. • Maintaining the schema for large and complex operations sometimes slows down the processing of XML document
36.	**Explain DTD for XML Schemas.** • XML documents are processed by applications • Applications have assumptions about XML documents • DTDs allow to formalize some of these constraints • Part of the constraint checking must still be programmed
37.	**List some browsers that support XML and XSL** ***Mozilla Firefox*** As of version 1.0.2, Firefox has support for XML and XSLT (and CSS). ***Mozilla:*** *Mozilla includes Expat for XML parsing and has support to display XML + CSS. Mozilla also has some support for Namespaces. Mozilla is available with an XSLT implementation.* ***Netscape:*** *As of version 8, Netscape uses the Mozilla engine, and therefore it has the same XML / XSLT support as Mozilla.* ***Opera:*** *As of version 9, Opera has support for XML and XSLT (and CSS). Version 8 supports only XML + CSS.* ***Internet Explorer:*** *As of version 6, Internet Explorer supports XML, Namespaces, CSS, XSLT, and XPath. Version 5 is NOT compatible with the official W3C XSL Recommendation.*
38.	**What is XML presentation technique?** XML presentation technologies provide a modular way to deliver and display content to a variety of devices. There are different presentation technologies used in XML to display the content. Eg: CSS
39.	**List some of presentation technologies.** Presentation technologies provide a modular way to deliver and display content to a variety of devices. i) CSS ii) XSL iii) XFORMS iv) XHTML
40.	**Write about DOM.** DOM is W3c supported standard application programming interface(API) that provides a platform and

	language- neutral interface to allow developers to programmatically access and modify the content and structure documents.
41.	**What is SAX?** SAX is an example of a grass- roots development effort to provide a simple; Java based API for processing XML.
42.	**What are the levels of DOM?** DOM provides a platform and language- neutral interface to allow developers to programmatically access and modify the content and structure documents. It has Level 0, Level 1, Level 2, Level 3
43.	**Compare CSS and XSL.** CSS can be used with HTML.But XSL can't be used in HTML Both can be used in XML CSS is not a transformation language but XSL.
	Part – B

1.	List and explain the XML syntax rules in detail. Explain how a XML document can be displayed on a browser. (APR/MAY 2011)
	• ***All XML Elements Must Have a Closing Tag***
	In HTML, some elements do not have to have a closing tag:
	In XML, it is illegal to omit the closing tag. All elements **must** have a closing tag:
	`<p>This is a paragraph.</p>`
	`<br />`
	
	• ***XML Tags are Case Sensitive***
	XML tags are case sensitive. The tag <Letter> is different from the tag <letter>.
	Opening and closing tags must be written with the same case:
	`<Message>This is incorrect</message>`
	`<message>This is correct</message>`
	Note: "Opening and closing tags" are often referred to as "Start and end tags". Use whatever you prefer. It is exactly the same thing.
	
	• ***XML Elements Must be Properly Nested***
	In XML, all elements **must** be properly nested within each other:
	`<b><i>This text is bold and italic</i></b>`
	In the example above, "Properly nested" simply means that since the <i> element is opened inside the <b> element, it must be closed inside the <b> element.
	
	• ***XML Documents Must Have a Root Element***
	XML documents must contain one element that is the **parent** of all other elements. This element is called the **root** element.
	`<root>`
	`  <child>`
	`    <subchild>.... </subchild>`
	`  </child>`
	`</root>`
	
	• ***XML Attribute Values Must be Quoted***
	XML elements can have attributes in name/value pairs just like in HTML.
	In XML, the attribute values must always be quoted.
	
	`<note date="12/11/2007">`
	`  <to>Tove</to>`

```
  <from>Jani</from>
</note>
```

The error in the first document is that the date attribute in the note element is not quoted.

- ***Entity References***

Some characters have a special meaning in XML.
If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element.
This will generate an XML error:

```
<message>if salary < 1000 then</message>
```

To avoid this error, replace the "<" character with an **entity reference**:

```
<message>if salary &lt; 1000 then</message>
```

There are 5 pre-defined entity references in XML:

&lt;	<	less than
&gt;	>	greater than
&amp;	&	ampersand
&apos;	'	apostrophe
&quot;	"	quotation mark

Note: Only the characters "<" and "&" are strictly illegal in XML. The greater than character is legal, but it is a good habit to replace it.

- ***Comments in XML***

The syntax for writing comments in XML is similar to that of HTML.
<!-- This is a comment -->

- ***White-space is Preserved in XML***

XML does not truncate multiple white-spaces in a document (while HTML truncates multiple white-spaces to one single white-space):

```
XML:   Hello         Tove
HTML: Hello Tove
```

- ***XML Stores New Line as LF***

Windows applications store a new line as: carriage return and line feed (CR+LF).
Unix and Mac OSX uses LF.
Old Mac systems uses CR.
XML stores a new line as LF.

- ***Well Formed XML***

XML documents that conform to the syntax rules above are said to be "Well Formed" XML documents.

Displaying XML document on a browser

XML documents do not carry information about how to display the data. XML is a technology for describing the structure of data.
Since XML tags are "invented" by the author of the XML document, browsers do not know if a tag like <table> describes an HTML table or a dining table.
Without any information about how to display the data, most browsers will just display the XML document as it is.

```
<?xml version="1.0" encoding="UTF-8"?>
 <note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
 </note>
```

Look at the XML file above in the browser: note.xml
An XML document will be displayed with color-coded root and child elements. A plus (+) or minus sign (-) to the left of the elements can be clicked to expand or collapse the element structure. To view the raw XML source (without the + and - signs), select "View Page Source" or "View Source" from the browser menu.

2. Explain the role of XML namespaces with examples. (MAY/JUNE 2012)

XML allows document authors to create custom elements. This extensibility can result in naming collisions (i.e. different elements that have the same name) among elements in an XML document.
An XML namespace is a collection of element and attribute names. Each namespace has a unique name that provides a means for document authors to unambiguously refer to elements with the same name (i.e. prevent collisions).
For example,
<subject>Geometry</subject>
and
<subject>Cardiology</subject>
use element subject to markup data. In the first case the subject is something one studies in school, whereas in the second case the subject is in the field of medicine. Namespaces can differentiate these two subject elements. For example,
<school:subject>Math</school:subject>
and
<medical:subject>Thrombosis</medical:subject>
Both school and medical are namespace prefixes. A document author prepends a namespace prefix to an element or attribute name to specify the namespace for that element or attribute. Each namespace prefix has a corresponding uniform resource identifier (URI) that uniquely identifies the namespace. A URI is simply a series of characters for differentiating names.
For example, the string urn:deitel:book could be a URI for a namespace that contains elements and attributes related to Deitel & Associates, Inc. publications. Document authors can create their own namespace prefixes using virtually any name, except the reserved namespace xml.

Differentiating Elements with namespaces:

Namespaces differentiate two distinct elements i.e., **file** element related to text file and **file** document related to an image file.

```
<?xml version="1.0"?>
<!-- namespace.xml -->
<text:directory
      xmlns:text="urn:deitel:textInfo"
      xmlns:image="urn:deitel:imageInfo">

  <text : file filename="book.xml">
      <text : description>A book list</text:description>
  </text : file>

  <image : file filename="funny.jpg">
```

```
        <image : description>A funny picture</image:description>
        <image : size height="100" width="200"/>
    </image : file>
</text : directory>
```

xmlns attribute: - to create namespaces prefixes. Eg. text and image. Each name space prefix is bound to a series of characters called a Uniform Resource Identifier (URI) that uniquely identifies the name space. A URI is a way to identifying a resource on the Internet. Two popular types of URI are Uniform Resource Name (URN) and Uniform Resource Locator (URL).

Another common practice is to use URL – specify the location of a file or a resource on the Internet.

```
<text:directory
        Xmlns : text="http://www.deitel.com/xmlns-text”
        Xmlns : image="http://deitel.com/xmlns-image">
```

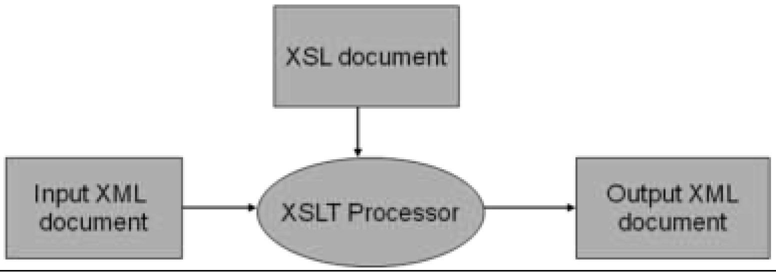
Namespace prefix are required for elements such as file, description etc.,
Attributes do not require namespace prefix because each attribute Is already part of an element the specifies the namespace prfix.

Specifying a Default Namespace:
To eliminate the need to place namespace prefixes in each element, document authors may specify a default namespace for an element and its children.

```
<?xml version="1.0"?>
<!--defaultnamespace.xml -->

<directory xmlns : image="urn:deitel:imageInfo"
             xmlns="urn:deitel:textInfo">

    <file filename="book.xml">
        <description>A book list</description>
    </file>

    <image : file filename="funny.jpg">
        <image:description>A funny picture</image:description>
        <image:size height="100" width="200"/>
    </image:file>

</directory>
```

3. **Given an XSLT document and a source XML document explain the XSLT transformation process that produ a single result XML document. (NOV/DEC 2012)**

- The Extensible Stylesheet Language (XSL) is an XML vocabulary typically used to transform XML documents from one form to another form



- JAXP allows a Java program to use the **Extensible Stylesheet Language (XSL)** to extract data from one XML document, process that data, and produce another XML document containing the processed data.

For example, XSL can be used to extract information from an XML document and embed it within an XHTML document so that the information can be viewed using a web browser.

TRANSFORMER:
Main.java:

```
import java.io.FileOutputStream;
import javax.xml.transform.Transformer;
import   javax.xml.transform.TransformerFactory;
import  javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;
public class Main
{
   static String xml="D://WebTech//trans.XML";
   static String xslt="D://WebTech//trans.XSL";
   static String output="D://WebTech//trans.HTML";
   public static void main(String[] args)
   {
      try
      {
      TransformerFactory tf = TransformerFactory.newInstance();
      Transformer tr = tf.newTransformer(new StreamSource(xslt));
      tr.transform(new StreamSource(xml),new StreamResult(new FileOutputStream(output)));
      System.out.println("Output to " + output);
      }
      catch(Exception e)
      {
       }
   }
}
```

trans.xsl:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
      <html>
         <body>
            <h1>Indian Languages details</h1>
            <table border="1">
               <tr>
                  <th>Language</th>
                  <th>Family/Origin</th>
                  <th>No. of speakers</th>
```

```
                <th>Region</th>
              </tr>
      <xsl:for-each select="language">
              <tr>
                <td><xsl:value-of select="name"/></td>
                <td><xsl:value-of select="family"/></td>
                <td><xsl:value-of select="users"/></td>
                <td><xsl:value-of select="region"/></td>
              </tr>
            </xsl:for-each>
           </table>
         </body>
       </html>
  </xsl:template>
</xsl:stylesheet>
```

trans.xml:

```
<?xml version="1.0"?>
<!--<?xml-stylesheet type="text/xsl" href="trans.xsl"?>-->
<language>
<name>Kannada</name>
<region>Karnataka</region>
<users>38M</users>
<family>Dravidian</family>
</language>
```

trans.html:

Indian Languages details

Language	Family/Origin	No. of speakers	Region
Kannada	Dravidian	38M	Karnataka

4. **Write short notes on Event-oriented parsing (MAY/JUNE 2014)**

SAX:

- An alternative approach is to have the parser interact with an application as it reads an XML document. This is the approach taken by SAX (Simple API for XML).
- SAX allows an application to register event listeners with the XML parser.
- SAX parser calls these listeners as events occur and passes them the information about the events.

Main.Java:

```
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
public class Main
{
    public static void main(String args[])
   {
   try
```

```
        {
        SAXParserFactory factory = SAXParserFactory.newInstance();
        SAXParser saxParser = factory.newSAXParser();
        saxParser.parse("D://Staff1.XML",new CountHelper());
        }
catch(Exception e)
    {
    e.printStackTrace();
    }
}
}
```

CountHelper.JAVA:

```
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;
public class CountHelper extends DefaultHandler
{
int no_elms;
/*CountHelper()
{
super();
}*/
    @Override
public void startDocument() throws SAXException
    {
    no_elms=0;
    //return;
    }
    @Override
public void startElement(String u,String ln,String qname,Attributes atts)
            throws SAXException
    {
        if(qname.equals("firstname"))
        {
            no_elms++;
        }
       // return;
    }

 @Override
public void endDocument() throws SAXException
    {
        System.out.println("I/p Doc has " + no_elms + "firstname Elements");

    }
}
```

Staff1.xml:

```
<?xml version="1.0"?>
<company>
        <staff id="1001">
                <firstname>yong</firstname>
                <lastname>mook kim</lastname>
                <nickname>mkyong</nickname>
                <salary>100000</salary>
        </staff>
        <staff id="2001">
                <firstname>low</firstname>
                <lastname>yin fong</lastname>
                <nickname>fong fong</nickname>
                <salary>200000</salary>
        </staff>
</company>
```

OUTPUT:

DOM OUTPUT:

```
RootSystem element :company
Input Elements has:2nodes
```

SAX OUTPUT:

```
I/p Doc has 2 firstname Elements
```

5. Explain the following: i) XML namespace ii) XML style sheet. iii) XML attributes iv) XML Schema

i) XML Namespaces

XML Namespaces provide a method to avoid element name conflicts.

Name Conflicts

In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications. This XML carries HTML table information:

```
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

This XML carries information about a table (a piece of furniture):

```
<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

If these XML fragments were added together, there would be a name conflict. Both contain a

<table> element, but the elements have different content and meaning.
A user or an XML application will not know how to handle these differences.

Solving the Name Conflict Using a Prefix

Name conflicts in XML can easily be avoided using a name prefix. This XML carries information about an HTML table, and a piece of furniture:

```
<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table>
  <f:name>African Coffee
Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```

In the example above, there will be no conflict because the two <table> elements have different names.

XML Namespaces - The xmlns Attribute

When using prefixes in XML, a so-called **namespace** for the prefix must be defined. The namespace is defined by the **xmlns attribute** in the start tag of an element. The namespace declaration has the following syntax. xmlns:prefix="URI".

```
<root>

<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table xmlns:f="http://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>

</root>
```

In the example above, the xmlns attribute in the <table> tag give the h: and f: prefixes a qualified namespace. When a namespace is defined for an element, all child elements with the same prefix are associated with the same namespace. Namespaces can be declared in the elements where they are used or in the XML root element:

```
<root
xmlns:h="http://www.w3.org/TR/html4/"
xmlns:f="http://www.w3schools.com/furnitur
e">
<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
</root>
```

Uniform Resource Identifier (URI)

A **Uniform Resource Identifier** (URI) is a string of characters which identifies an Internet Resource. The most common URI is the **Uniform Resource Locator** (URL) which identifies an Internet domain address. Another, not so common type of URI is the **Universal Resource Name** (URN). In our examples we will only use URLs.

Default Namespaces

Defining a default namespace for an element saves us from using prefixes in all the child elements. It has the following syntax:

```
xmlns="namespaceURI"

This XML carries HTML table information:
<table xmlns="http://www.w3.org/TR/html4/">
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>

This XML carries information about a piece of
furniture:
<table
xmlns="http://www.w3schools.com/furniture">
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

Namespaces in Real Use

XSLT is an XML language that can be used to transform XML documents into other formats, like HTML. In the XSLT document below, you can see that most of the tags are HTML tags. The tags that are not HTML tags have the prefix xsl, identified by the namespace

```
xmlns:xsl="http://www.w3.org/1999/XSL/Transform":
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr>
      <th style="text-align:left">Title</th>
      <th style="text-align:left">Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
    </xsl:for-each>
  </table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

ii) XML Stylesheet

Displaying XML with XSLT

XSLT (eXtensible Stylesheet Language Transformations) is the recommended style sheet language for XML.

XSLT is far more sophisticated than CSS. With XSLT you can add/remove elements and attributes to or from the output file. You can also rearrange and sort elements, perform tests and make decisions about which elements to hide and display, and a lot more.

XSLT uses XPath to find information in an XML document.

XSLT Example

We will use the following XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<breakfast_menu>

<food>
<name>Belgian Waffles</name>
<price>$5.95</price>
<description>Two of our famous Belgian Waffles with plenty of real maple syrup</description>
<calories>650</calories>
</food>

<food>
<name>Strawberry Belgian Waffles</name>
<price>$7.95</price>
<description>Light Belgian waffles covered with strawberries and whipped cream</description>
<calories>900</calories>
</food>

<food>
<name>Berry-Berry Belgian Waffles</name>
```

```
<price>$8.95</price>
<description>Light Belgian waffles covered with an assortment of fresh berries and whipped
cream</description>
<calories>900</calories>
</food>

<food>
<name>French Toast</name>
<price>$4.50</price>
<description>Thick slices made from our homemade sourdough bread</description>
<calories>600</calories>
</food>

<food>
<name>Homestyle Breakfast</name>
<price>$6.95</price>
<description>Two eggs, bacon or sausage, toast, and our ever-popular hash browns</description>
<calories>950</calories>
</food>

</breakfast_menu>
```

Use XSLT to transform XML into HTML, before it is displayed in a browser:

Example XSLT Stylesheet:

```
<?xml version="1.0" encoding="UTF-8"?>
<html xsl:version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<body style="font-family:Arial;font-size:12pt;background-color:#EEEEEE">
<xsl:for-each select="breakfast_menu/food">
  <div style="background-color:teal;color:white;padding:4px">
    <span style="font-weight:bold"><xsl:value-of select="name"/> - </span>
    <xsl:value-of select="price"/>
    </div>
  <div style="margin-left:20px;margin-bottom:1em;font-size:10pt">
    <p>
    <xsl:value-of select="description"/>
    <span style="font-style:italic"> (<xsl:value-of select="calories"/> calories per serving)</span>
    </p>
  </div>
</xsl:for-each>
</body>
</html>
```

iii) XML Attributes:

XML elements can have attributes, just like HTML. Attributes provide additional information about an element.

XML Attributes

In HTML, attributes provide additional information about elements:

```
<img src="computer.gif">
<a href="demo.asp">
```

Attributes often provide information that is not a part of the data. In the example below, the file type is irrelevant to the data, but can be important to the software that wants to manipulate the element:

```
<file type="gif">computer.gif</file>
```

- **XML Attributes Must be Quoted**

  Attribute values must always be quoted. Either single or double quotes can be used. For a person's gender, the person element can be written like this:

  `<person gender="female">` or like this:

  `<person gender='female'>`

  If the attribute value itself contains double quotes you can use single quotes, like in this example:

  `<gangster name='George "Shotgun" Ziegler'>` or you can use character entities:

  `<gangster name="George &quot;Shotgun&quot; Ziegler">`

- **XML Elements vs. Attributes**

  Take a look at these examples:

```
<person gender="female">
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

```
<person>
  <gender>female</gender>
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

In the first example gender is an attribute. In the last, gender is an element. Both examples provide the same information. There are no rules about when to use attributes or when to use elements.

- **Avoid XML Attributes?**

Some of the problems with using attributes are:

- attributes cannot contain multiple values (elements can)
- attributes cannot contain tree structures (elements can)
- attributes are not easily expandable (for future changes)

Attributes are difficult to read and maintain. Use elements for data. Use attributes for information that is not relevant to the data.

- **XML Attributes for Metadata**

Sometimes ID references are assigned to elements. These IDs can be used to identify XML elements in much the same way as the id attribute in HTML. This example demonstrates this:

```
<messages>
  <note id="501">
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
  </note>
  <note id="502">
    <to>Jani</to>
    <from>Tove</from>
    <heading>Re: Reminder</heading>
    <body>I will not</body>
  </note>
</messages>
```

The id attributes above are for identifying the different notes. It is not a part of the note itself. The metadata (data about data) should be stored as attributes, and the data itself should be stored as elements.

iv) XML Schema

- An XML Schema describes the structure of an XML document, just like a DTD.
- An XML document with correct syntax is called "Well Formed".
- An XML document validated against an XML Schema is both "Well Formed" and "Valid".

The purpose of an XML Schema is to define the legal building blocks of an XML document, just like a DTD.

An XML Schema:

- defines elements that can appear in a document
- defines attributes that can appear in a document
- defines which elements are child elements
- defines the order of child elements
- defines the number of child elements
- defines whether an element is empty or can include text
- defines data types for elements and attributes
- defines default and fixed values for elements and attributes

XML Schema Example

```
<?xml version="1.0"?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

The Schema above is interpreted like this:

- <xs:element name="note"> defines the element called "note"
- <xs:complexType> the "note" element is a complex type
- <xs:sequence> the complex type is a sequence of elements
- <xs:element name="to" type="xs:string"> the element "to" is of type string (text)
- <xs:element name="from" type="xs:string"> the element "from" is of type string
- <xs:element name="heading" type="xs:string"> the element "heading" is of type string
- <xs:element name="body" type="xs:string"> the element "body" is of type string
  Everything is wrapped in "Well Formed" XML.

6. **Explain XSL with suitable exemple**

XSL is a language for expressing style sheets. An XSL style sheet is, like with CSS, a file that describes how to display an XML document of a given type. XSL shares the functionality and is compatible with CSS2 (although it uses a different syntax). It also adds:

- A transformation language for XML documents: **XSLT**. Originally intended to perform complex styling operations, like the generation of tables of contents and indexes, it is now used as a general purpose XML processing language. XSLT is thus widely used for purposes other than XSL, like generating HTML web pages from XML data.
- Advanced styling features, expressed by an XML document type which defines a set of elements called **Formatting Objects**, and attributes (in part borrowed from CSS2 properties and adding more complex ones.

How Does It Work?

Styling requires a source XML documents, containing the information that the style sheet will display and the style sheet itself which describes how to display a document of a given type.

Example:

In this XSL example, two lists in XML and use the XSL style sheet to set the style display the way I want o se
created. Also HTML is embedded in the style sheet which allows an actual ordered list to be created.

The XML File

```
<?xml version="1.0"?>
<!DOCTYPE LANGLIST SYSTEM "langlist.dtd">
<?xml-stylesheet type="text/xsl" href="xmlstyle.xsl"?>
<LANGLIST>
   <TITLE>List of Items Important to Markup Languages</TITLE>
   <TITLE1>Languages</TITLE1>
      <LIST1>
      <LANGUAGES>SGML</LANGUAGES>
      <LANGUAGES>XML</LANGUAGES>
      <LANGUAGES>HTML</LANGUAGES>
      </LIST1>
   <TITLE2>Other Support</TITLE2>
      <LIST2>
      <OTHER>DTD</OTHER>
      <OTHER>DSSSL</OTHER>
      <OTHER>Style Sheets</OTHER>
     </LIST2>
</LANGLIST>
```

The DTD File

```
<!ELEMENT LANGLIST ANY>
<!ENTITY % Shape "(rect|circle|poly|default)">
<!ELEMENT TITLE  (#PCDATA)>
<!ELEMENT TITLE1  (#PCDATA)>
<!ELEMENT TITLE2  (#PCDATA)>
<!ELEMENT LIST1 (LANGUAGES)+>
<!ELEMENT LIST2 (OTHER)+>
<!ELEMENT LANGUAGES  (#PCDATA)>
<!ATTLIST LANGUAGES
   type %Shape; #IMPLIED>
<!ELEMENT OTHER  (#PCDATA)>
<!ATTLIST OTHER
   type (disc|square|circle) #IMPLIED>
```

This DTD file employs an ENTITY, just for demonstration purposes. An entity, in this case is similar to a variable with a value that is set to a string value. In this case the entity name is "Shape" and the string value is "(rect|circle|poly|default)". This value is used to set the attribute list (ATTLIST) for the LANGUAGES and OTHER elements, although these attributes are not used in this example.

The XML Style File

```
<?xml version="1.0"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

      <xsl:template match="/">
```

```
            <xsl:apply-templates select="LANGLIST/TITLE" />
            <xsl:apply-templates select="LANGLIST/TITLE1" />
            <xsl:apply-templates select="LANGLIST/LIST1" />
            <xsl:apply-templates select="LANGLIST/TITLE2" />
            <xsl:apply-templates select="LANGLIST/LIST2" />
      </xsl:template>

      <xsl:template match="TITLE">
            <SPAN STYLE="display: 'block'; font-family: 'arial';
color: '#008000'; font-weight: '600'; font-size: '22'; margin-top:
'12pt'; text-align: 'center'">
            <xsl:value-of />
            </SPAN>
            <BR/>
      </xsl:template>

      <xsl:template match="TITLE1">
            <SPAN STYLE="display: 'block'; font-family: 'arial';
color: '#000080'; font-weight: '400'; font-size: '20'; margin-top:
'12pt'">
            <xsl:value-of />
            </SPAN>
            <BR/>
      </xsl:template>

      <xsl:template match="LIST1">
            <UL style="display: 'list-item'; list-style-image:
url('bullet8.gif'); font-family: 'arial'; color: '#000000'; font-
weight: '400';  margin-left: '15pt'; margin-top: '12pt'; font-size:
'18'">
            <xsl:for-each select="LANGUAGES">
                  <LI style="display: 'list-item'; list-style-type:
'square'; list-style-image: url('bullet8.gif'); font-family: 'arial';
color: '#ff0000'; font-weight: '300';  margin-left: '15pt'; margin-
top: '12pt'; font-size: '16'">
                  <xsl:value-of />
                  </LI>
            </xsl:for-each>
            </UL>
      </xsl:template>

      <xsl:template match="TITLE2">
            <SPAN STYLE="display: 'block'; font-family: 'arial';
color: '#000080'; font-weight: '400'; font-size: '20'; margin-top:
'12pt'">
            <xsl:value-of />
            </SPAN>
```

	`<BR/>`
	`</xsl:template>`
	
	`<xsl:template match="LIST2">`
	`<UL style="display: 'list-item'; list-style-image: url('bullet8.gif'); font-family: 'arial'; color: '#000000'; font-weight: '400';  margin-left: '15pt'; margin-top: '12pt'; font-size: '18'">`
	`<xsl:for-each select="OTHER">`
	`<LI style="display: 'list-item'; list-style-type: 'square'; list-style-image: url('bullet8.gif'); font-family: 'arial'; color: '#0000ff'; font-weight: '200';  margin-left: '15pt'; margin-top: '12pt'; font-size: '14'">`
	`<xsl:value-of "."/>`
	`</LI>`
	`</xsl:for-each>`
	`</UL>`
	`</xsl:template>`
	
	`</xsl:stylesheet>`
	Between the <xsl:template match="/"> tag and </xsl:template> tag, the order of the "apply-templates" statements is very important since this determines the order they are displayed in. Also note that the text <xsl:value-of "."/> is used to refer to the XML object currently being processed in order to display the value of that object.

7.	**Explain the architectural revolution of XML.** ***<u>XML: The Three Revolutions</u>*** Three areas of impact are i) **data**, which XML frees from the confines of fixed, ii) program-dependent formats; **architecture**, iii) with a change in emphasis from tightly coupled distributed systems to a more loosely coupled confederation based on the Web; and **software**, with the realization that software evolution is a better path to managing complexity than building monolithic applications.

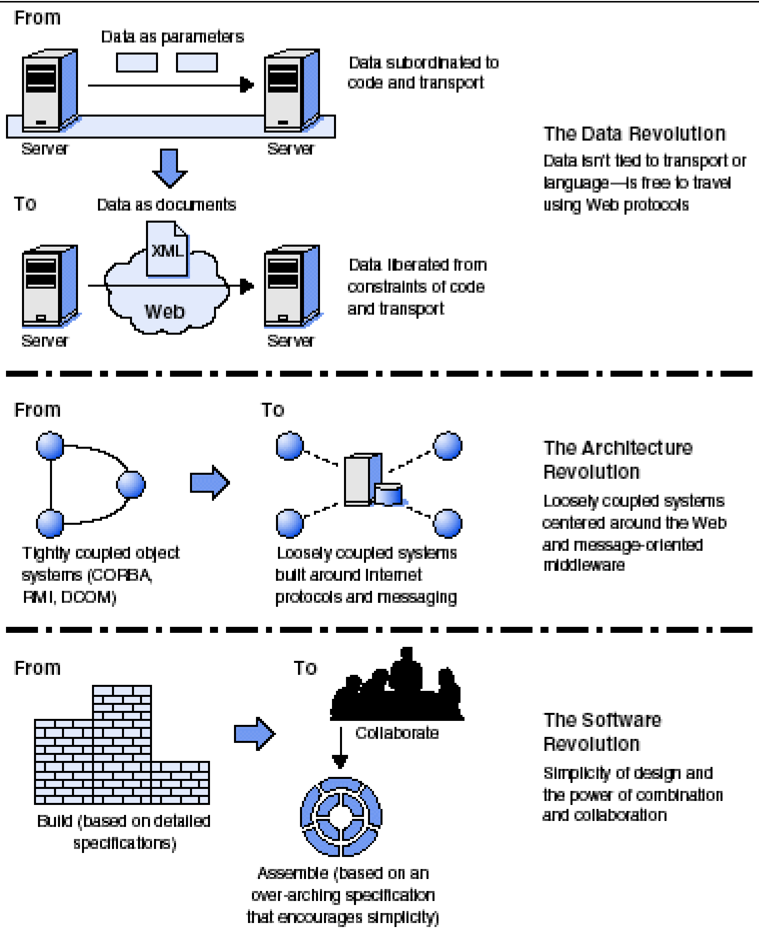
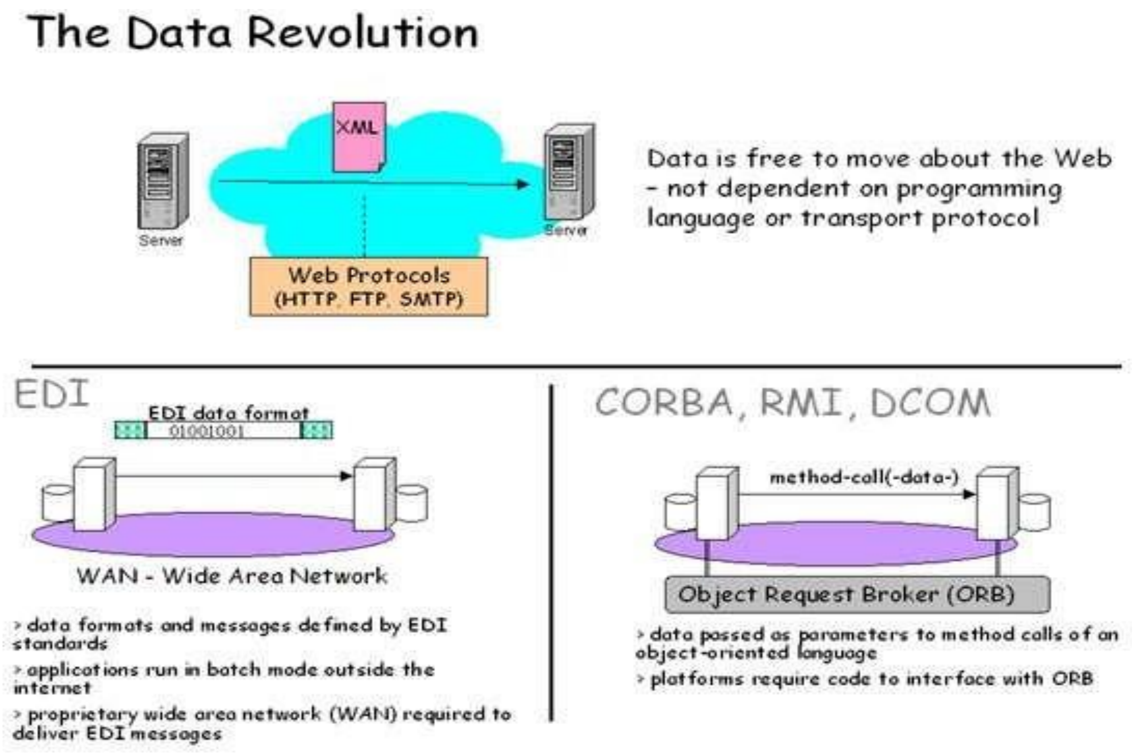

From
Data as parameters
Server
Server
Data subordinated to code and transport
The Data Revolution
Data isn't tied to transport or language—is free to travel using Web protocols
To
Data as documents
XML
Web
Server
Server
Data liberated from constraints of code and transport
From
To
The Architecture Revolution
Loosely coupled systems centered around the Web and message-oriented middleware
Tightly coupled object systems (CORBA, RMI, DCOM)
Loosely coupled systems built around Internet protocols and messaging
From
To
Collaborate
The Software Revolution
Simplicity of design and the power of combination and collaboration
Build (based on detailed specifications)
Assemble (based on an over-arching specification that encourages simplicity)

Figure: The three XML revolutions: data, architecture, and software.

<u>The Data Revolution</u>
Prior to XML, data was very much proprietary, closely associated with applications that understood how data was formatted and how to process it. Now, XML-based industry-specific data vocabularies provide alternatives to specialized Electronic Data Interchange (EDI) solutions by facilitating B2B data exchange and playing a key role as a messaging infrastructure for distributed computing.

XML's strength is its data independence. XML is pure data description, not tied to any programming language, operating system, or transport protocol. In the grand scheme of distributed computing this is a radical idea. The implication is that we don't require lock-in to programmatic infrastructures to make data available to Web-connected platforms. In effect, data is free to move about globally without the constraints imposed by tightly coupled transport-dependent architectures. XML's sole focus on data means that a variety of transport technologies may be used to move XML across the Web. As a result, protocols such as HTTP have had a tremendous impact on XML's viability and have opened the door to alternatives to CORBA, RMI, and DCOM, which don't work over TCP/IP. XML does this by focusing on data and leaving other issues to supporting technologies.



Data is free to move about the Web – not dependent on programming language or transport protocol

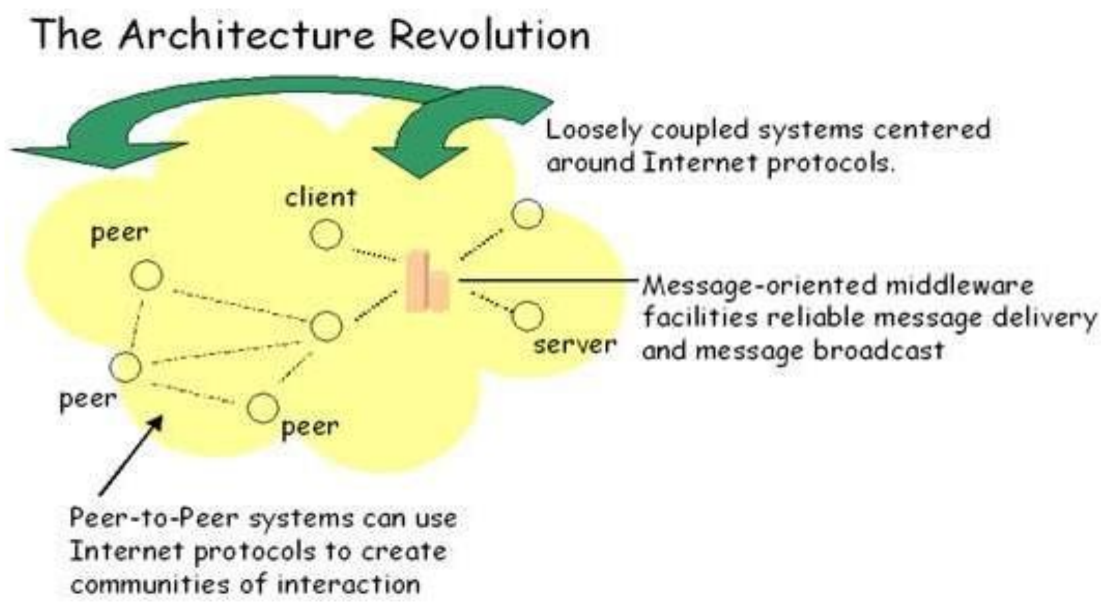
› data formats and messages defined by EDI standards
› applications run in batch mode outside the internet
› proprietary wide area network (WAN) required to deliver EDI messages

› data passed as parameters to method calls of an object-oriented language
› platforms require code to interface with ORB

The Data Revolution and The Architectural Revolution

Together these XML-based technology initiatives open up new possibilities for distributed computing that leverage the existing infrastructure of the Web and create a transition from object-based distributed systems to architectures based on Web services that can be discovered, accessed, and assembled using open Web technologies. The focal point of this change in architectural thinking has been a move from tightly coupled systems based on established infrastructures such as CORBA, RMI, and DCOM, each with their own transport protocol, to loosely coupled systems riding atop standard Web protocols such as TCP/IP. Although the transport protocols underlying CORBA, RMI, and DCOM provide for efficient communication between nodes, their drawback is their inability to communicate with other tightly coupled systems or directly with the Web.

Loosely coupled Web-based systems, on the other hand, provide what has long been considered the Holy Grail of computing: universal connectivity. Using TCP/IP as the transport, systems can establish connections with each other using common open-Web protocols. Although it is possible to build software bridges linking tightly coupled systems with each other and the Web, such efforts are not trivial and add another layer of complexity on top of an already complex infrastructure.


The Architecture Revolution
Loosely coupled systems centered around Internet protocols.
client
peer
peer
peer
server
Message-oriented middleware facilities reliable message delivery and message broadcast
Peer-to-Peer systems can use Internet protocols to create communities of interaction

The Architecture Revolution and The Software Revolution

XML is also part of a revolution in how we build software. During the 1970s and 1980s, software was constructed as monolithic applications built to solve specific problems. The problem with large software projects is that, by trying to tackle multiple problems at once, the software is often ill-suited to adding new functionality and adapting to technological change. In the 1990s a different model for software emerged based on the concept of simplicity.

8. Write a program using PHP that creates the web application for result publication

```
<?php
$username = "your_name";
$password = "your_password";
$hostname = "localhost";

//connection to the database
$dbhandle = mysql_connect($hostname, $username, $password)
 or die("Unable to connect to MySQL");
echo "Connected to MySQL<br>";

//select a database to work with
$selected = mysql_select_db("results",$dbhandle)
  or die("Could not select examples");

//execute the SQL query and return records
$result = mysql_query("SELECT regno, sub1,sub2 FROM res_tab");
```

```
//fetch tha data from the database
while ($row = mysql_fetch_array($result)) {
  echo "REGNO:".$row{'regno'}." SUB1:".$row{'sub1'}."SUB2: ". //display the results
  $row{'sub2'}."<br>";
}
//close the connection
mysql_close($dbhandle);
?>
```

To create 'results' database on your MySQL server you should run the following script:

```
CREATE DATABASE `results`;
USE `results`;
CREATE TABLE `res_tab` (
  `regno` int UNIQUE NOT NULL,
  `sub1` varchar(2),
  `sub2` varchar(2),
  PRIMARY KEY(id)
);
INSERT INTO cars VALUES(1,'A','B');
INSERT INTO cars VALUES(2,'C','D');
INSERT INTO cars VALUES(3,'Si','A');
```

9. **a) Design simple calculator using PHP.**

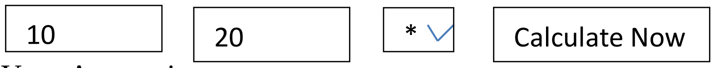
HTML TextBox value, in other words operand value is accepted in this way:

$_REQUEST['name']

In that way you can simply store the TextBox value or drop down list value for calculation in a PHP variable.

```
<html>
<head>
<title>Simple PHP Calculator</title>
</head>

<body>
<form method='post' action='calculator.php'>

<input type='text' name='value1'>
<input type='text' name='value2'><select name='action'>
<option>+</option>
<option>-</option>
<option>*</option>
<option>/</option>
</select>
<input type='submit' name='submit' value='Calculate Now'></form>

<?php
if(isset($_POST['submit']))
{$value1 = $_POST['value1'];
$value2 = $_POST['value2'];
$action = $_POST['action'];
if($action=="+"){
echo "<b>Your Answer is:</b><br>";
echo $value1+$value2;
}

if($action=="-"){
echo "<b>Your Answer is:</b><br>";
echo $value1-$value2;
}

if($action=="*"){
echo "<b>Your Answer is:</b><br>";
echo $value1*$value2;
}

if($action=="/"){
echo "<b>Your Answer is:</b><br>";
echo $value1/$value2;
}
}
?>

</body>
</html>
```

Output:


10 20 *

```
Your Answer is:
200
```

---

b) Design application to send a email using PHP

Description of mail function:

bool **mail** (string `$to` , string `$subject` , string `$message` [, string `$additional_headers` [, string `$additional_parameters`]])

```
<?php
//if "email" variable is filled out, send email
  if (isset($_REQUEST['email'])) {

  //Email information
  $admin_email = "someone@example.com";
  $email = $_REQUEST['email'];
  $subject = $_REQUEST['subject'];
  $comment = $_REQUEST['comment'];

  //send email
  mail($email, "$subject", $comment, "From:" . $admin_email);
  //Email response
  echo "Thank you for contacting us!";
  }

  //if "email" variable is not filled out, display the form
  else {
?>

 <form method="post">
  Email: <input name="email" type="text" /><br />
  Subject: <input name="subject" type="text" /><br />
  Message:<br />
  <textarea name="comment" rows="15" cols="40"></textarea><br />
  <input type="submit" value="Submit" />
</form>

<?php
  }
?>
```

Output:

10. **Develop a shopping cart application using PHP with use of cookies.**

Use of each files and folders in the shopping cart application.

1. config is a folder that contains another file called **db_connect.php** which is the file used to connect to the MySQL database for retrieving product information.

2. ***add_to_cart.php*** is executed when the user clicks on the "add to cart" button. It processes the product data by adding it to the cookie JSON string to make it a cart item. Each cart item is processed and converted to a JSON string to be stored in the cookie.

3. ***cart.php*** is where the cart items are read and displayed to the user. We retrieve the JSON string from a cookie variable, convert it to an associated array and loop through it to be displayed on an HTML table.

4. layout_foot.php is used as a closing HTML wrapper, and is always included the the end of **cart.php** and **products.php** because it closes the tags started by layout_head.php, it also contains any JavaScript source and jQuery codes used in this script.

5. layout_head.php is used as an opening HTML page wrapper. It is always included at the beginning of **cart.php** and **products.php** because it contains the opening HTML tag, head tags, title tags and any CSS resource and scripts used.

6. navigation.php contains the links to products.php and cart.php pages that the user can click. The cart item count is also shown here, beside the “Cart” link.
7. products.php displays all the products retrieve from the database. We are using PDO extension to connect and retrieve data from the MySQL database.
8. remove_from_cart.php is executed when the user clicks on the “Remove from cart” button. It removes the product item from the JSON string saved in the cookie variable.
Important note: This source code focuses on using cookies for storing shopping cart items, the “checkout” is not in its scope.
Step 1: create a database and run the following SQL queries to create the sample tables.

```
CREATE TABLE IF NOT EXISTS `products` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(32) NOT NULL,
  `price` int(11) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=4 ;

--
-- Dumping data for table `products`
--

INSERT INTO `products` (`id`, `name`, `price`) VALUES
(1, 'LG P880 4X HD', 336),
(2, 'Google Nexus 4', 299),
(3, 'Samsung Galaxy S4', 600);
```

Step 2: In your chosen root directory, create a folder named config.
Step 3: **Inside that config folder, create a file named db_connect.php**, **and put the following code inside it, just change to database credentials to your own.**

```
<?php
$host = "your_host";
$db_name = "your_database_name";
$username = "your_database_username";
$password = "your_database_password";

try {
      $con = new PDO("mysql:host={$host};dbname={$db_name}",
$username, $password);
}

//to handle connection error
catch(PDOException $exception){
      echo "Connection error: " . $exception->getMessage();
}
?>
```

Step 4: **Create a file called products.php, we will retrieve the products using the code below.**

```
<?php
$page_title="Products";
```

```
include 'layout_head.php';

// to prevent undefined index notice
$action = isset($_GET['action']) ? $_GET['action'] : "";
$name = isset($_GET['name']) ? $_GET['name'] : "";

// show messages based on given action
if($action=='added'){
	echo "<div class='alert alert-info'>";
		echo "<strong>{$name}</strong> was added to your cart!";
	echo "</div>";
}

else if($action=='exists'){
	echo "<div class='alert alert-info'>";
		echo "<strong>{$name}</strong> already exists in your
cart!";
	echo "</div>";
}

// query the products
$query = "SELECT id, name, price FROM products ORDER BY name";
$stmt = $con->prepare( $query );
$stmt->execute();

$num = $stmt->rowCount();

if($num>0){
	//start table
	echo "<table class='table table-hover table-responsive table-
bordered'>";

        // our table heading
        echo "<tr>";
            echo "<th class='textAlignLeft'>Product Name</th>";
            echo "<th>Price (USD)</th>";
                   echo "<th>Action</th>";
        echo "</tr>";

        while ($row = $stmt->fetch(PDO::FETCH_ASSOC)){
            extract($row);

            //creating new table row per record
            echo "<tr>";
                echo "<td>{$name}</td>";
                echo "<td>&#36;{$price}</td>";
                         echo "<td>";
```

```
                                echo "<a
href='add_to_cart.php?id={$id}&name={$name}' class='btn btn-
primary'>";
                                      echo "<span class='glyphicon
glyphicon-shopping-cart'></span> Add to cart";
                                echo "</a>";
                          echo "</td>";
             echo "</tr>";
        }

    echo "</table>";
}

// tell the user if there's no products in the database
else{
    echo "No products found.";
}

include 'layout_foot.php';
?>
```

Step 5: products.php on step 4 above will not actually work without the layout_head.php and layout_foot.php, so first, we’ll create the layout_head.php with the following code:

```
<?php
// connect to database
include 'config/db_connect.php';
?>
<!DOCTYPE html>
<html lang="en">
<head>

    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1">

    <title><?php echo isset($page_title) ? $page_title : "The Code of
a Ninja"; ?> - LIVE DEMO</title>

    <!-- Bootstrap -->
    <!-- Latest compiled and minified CSS -->
    <link rel="stylesheet"
href="//netdna.bootstrapcdn.com/bootstrap/3.1.1/css/bootstrap.min.css"
>

    <!-- HTML5 Shiv and Respond.js IE8 support of HTML5 elements and
media queries -->
```

```
    <!-- WARNING: Respond.js doesn't work if you view the page via
file:// -->
    <!--[if lt IE 9]>
    <script
src="https://oss.maxcdn.com/libs/html5shiv/3.7.0/html5shiv.js"></scrip
t>
    <script
src="https://oss.maxcdn.com/libs/respond.js/1.4.2/respond.min.js"></sc
ript>
    <![endif]-->

</head>
<body>

      <?php include 'navigation.php'; ?>

    <!-- container -->
    <div class="container">

        <div class="page-header">
            <h1><?php echo isset($page_title) ? $page_title : "The
Code of a Ninja"; ?></h1>
        </div>
```

Step 6: **layout_head.php includes another PHP file called navigation.php, so we'll create it and put the following code.**

```
<!-- navbar -->
<div class="navbar navbar-default navbar-static-top"
role="navigation">
      <div class="container">

            <div class="navbar-header">
                  <button type="button" class="navbar-toggle" data-
toggle="collapse" data-target=".navbar-collapse">
                  <span class="sr-only">Toggle navigation</span>
                  <span class="icon-bar"></span>
                  <span class="icon-bar"></span>
                  <span class="icon-bar"></span>
                  </button>
                  <a class="navbar-brand" href="products.php">Your
Site</a>
            </div>

            <div class="navbar-collapse collapse">
                  <ul class="nav navbar-nav">
                        <li <?php echo $page_title=="Products" ?
```

```
"class='active'" : ""; ?> >
                                <a href="products.php">Products</a>
                        </li>
                        <li <?php echo $page_title=="Cart" ?
"class='active'" : ""; ?> >
                                <a href="cart.php">
                                        <?php
                                        // count products in cart
                                        $cookie =
$_COOKIE['cart_items_cookie'];
                                        $cookie = stripslashes($cookie);
                                        $saved_cart_items =
json_decode($cookie, true);

      $cart_count=count($saved_cart_items);
                                        ?>
                                        Cart <span class="badge"
id="comparison-count"><?php echo $cart_count; ?></span>
                                </a>
                        </li>
                  </ul>
             </div><!--/.nav-collapse -->

      </div>
</div>
<!-- /navbar -->
```

Step 7: **Now we'll create the layout_foot.php**

```
      </div>
      <!-- /container -->

<!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.0/jquery.min.js
"></script>

<!-- Include all compiled plugins (below), or include individual files
as needed -->
<!-- Latest compiled and minified JavaScript -->
<script
src="//netdna.bootstrapcdn.com/bootstrap/3.1.1/js/bootstrap.min.js"></
script>

</body>
</html>
```

Step 8: **products.php has links to the add_to_cart.php** file, **we'll create that file and put the code below.**

```
<?php
// initialize empty cart items array
$cart_items=array();

// get the product id and name
$id = isset($_GET['id']) ? $_GET['id'] : "";
$name = isset($_GET['name']) ? $_GET['name'] : "";

// add new item on array
$cart_items[$id]=$name;

// read the cookie
$cookie = $_COOKIE['cart_items_cookie'];
$cookie = stripslashes($cookie);
$saved_cart_items = json_decode($cookie, true);

// if $saved_cart_items is null, prevent null error
if(!$saved_cart_items){
      $saved_cart_items=array();
}

// check if the item is in the array, if it is, do not add
if(array_key_exists($id, $saved_cart_items)){
      // redirect to product list and tell the user it was already
added to the cart
      header('Location: products.php?action=exists&id' . $id .
'&name=' . $name);
}

else{
      // if cart has contents
      if(count($saved_cart_items)>0){
            foreach($saved_cart_items as $key=>$value){
                  // add old item to array, it will prevent duplicate
keys
                  $cart_items[$key]=$value;
            }
      }

      // put item to cookie
      $json = json_encode($cart_items, true);
      setcookie('cart_items_cookie', $json);

      // redirect
      header('Location: products.php?action=added&id=' . $id .
'&name=' . $name);
}
```

```
?>
```

Step 9: **Now if the products were able to be added on the cart, we'll have to view it using cart.php**, **we'll create that file with the following codes.**

```
<?php
$page_title="Cart";
include 'layout_head.php';

$action = isset($_GET['action']) ? $_GET['action'] : "";
$name = isset($_GET['name']) ? $_GET['name'] : "";

if($action=='removed'){
      echo "<div class='alert alert-info'>";
            echo "<strong>{$name}</strong> was removed from your
cart!";
      echo "</div>";
}

$cookie = $_COOKIE['cart_items_cookie'];
$cookie = stripslashes($cookie);
$saved_cart_items = json_decode($cookie, true);

if(count($saved_cart_items)>0){
      // get the product ids
      $ids = "";
      foreach($saved_cart_items as $id=>$name){
            $ids = $ids . $id . ",";
      }

      // remove the last comma
      $ids = rtrim($ids, ',');

      //start table
      echo "<table class='table table-hover table-responsive table-
bordered'>";

        // our table heading
        echo "<tr>";
            echo "<th class='textAlignLeft'>Product Name</th>";
            echo "<th>Price (USD)</th>";
                  echo "<th>Action</th>";
        echo "</tr>";

            $query = "SELECT id, name, price FROM products WHERE id
IN ({$ids}) ORDER BY name";
            $stmt = $con->prepare( $query );
            $stmt->execute();

            $total_price=0;
            while ($row = $stmt->fetch(PDO::FETCH_ASSOC)){
           extract($row);

                  echo "<tr>";
                        echo "<td>{$name}</td>";
                        echo "<td>&#36;{$price}</td>";
                        echo "<td>";
                              echo "<a
href='remove_from_cart.php?id={$id}&name={$name}' class='btn btn-
danger'>";
                                    echo "<span class='glyphicon
glyphicon-remove'></span> Remove from cart";
                              echo "</a>";
                        echo "</td>";
            echo "</tr>";

                  $total_price+=$price;
            }

            echo "<tr>";
                        echo "<td><b>Total</b></td>";
                        echo "<td>&#36;{$total_price}</td>";
                        echo "<td>";
                              echo "<a href='#' class='btn btn-
success'>";
                                    echo "<span class='glyphicon
glyphicon-shopping-cart'></span> Checkout";
                              echo "</a>";
                        echo "</td>";
            echo "</tr>";

      echo "</table>";
}

else{
      echo "<div class='alert alert-danger'>";
            echo "<strong>No products found</strong> in your cart!";
      echo "</div>";
}

include 'layout_foot.php';
?>
```

Step 10: **cart.php links to a file called remove_from_cart.php, to remove an item from the cart.**

We'll create remove_from_cart.php with the codes below.

```
<?php
// get the product id
$id = isset($_GET['id']) ? $_GET['id'] : "";
$name = isset($_GET['name']) ? $_GET['name'] : "";

// read
$cookie = $_COOKIE['cart_items_cookie'];
$cookie = stripslashes($cookie);
$saved_cart_items = json_decode($cookie, true);

// remove the item from the array
$saved_cart_items = array_diff($saved_cart_items, array($id=>$name));

// delete cookie value
setcookie("cart_items_cookie", "", time()-3600);

// enter new value
$json = json_encode($saved_cart_items, true);
setcookie('cart_items_cookie', $json);

// redirect to product list and tell the user it was added to cart
header('Location: cart.php?action=removed&id=' . $id . '&name=' .
$name);

?>
```

Enter or Update Cart Item Quantity

But what if your users want to enter or update the quantity of items in the cart? It is possible using cookies.

1 Add a new column named "Quantity" in **products.php** and **cart.php**. The word "Quantity" will be the column header and for the rest of the table row, it will be input or text boxes where the user can enter the product quantity before clicking the "Add to cart" or "Update cart" button.

2 On **products.php**, the "Add to cart" button will not be a direct link to add_to_cart.php file. It will be a button that will execute a jQuery code because it has to get the quantity entered by the user.

3 On **cart.php**, there will be an "Update cart" button beside the quantity textbox. Clicking it will run a jQuery script that gets the new quantity entered by the user and saves the changes with the help of a new file called "update_quantity.php".

11. **Explain about the control statements in PHP with example.**

PHP Conditional Statements

In PHP we have the following conditional statements:

- **if statement** - executes some code only if a specified condition is true
- **if...else statement** - executes some code if a condition is true and another code if the condition is false
- **if...elseif... else statement** - specifies a new condition to test, if the first condition is false

- **switch statement** - selects one of many blocks of code to be executed

<u>The if Statement</u>

The if statement is used to execute some code **only if a specified condition is true**.

Syntax

```
if (condition) {
   code to be executed if condition is true;
}
```

The example below will output "Have a good day!" if the current time (HOUR) is less than 20:

Example

```
<?php
$t = date("H");
if ($t < "20") {
   echo "Have a good day!";
}
?>
```

<u>The if...else Statement</u>

Use the if. ..else statement to execute some code **if a condition is true and another code if the condition is false**.

Syntax

```
if (condition) {
   code to be executed if condition is true;
} else {
   code to be executed if condition is false;
}
```

The example below will output "Have a good day!" if the current time is less than 20, and "Have a good night!" otherwise:

Example

```
<?php
$t = date("H");

if ($t < "20") {
   echo "Have a good day!";
} else {
   echo "Have a good night!";
}
?>
```

<u>The if...elseif. ..else Statement</u>

Use the if....elseif. else statement to **specify a new condition to test, if the first condition is false.**

Syntax

```
if (condition) {
   code to be executed if condition is true;
} elseif (condition) {
   code to be executed if condition is true;
} else {
   code to be executed if condition is false;
}
```

The example below will output "Have a good morning!" if the current time is less than 10, and "Have a good day!" if the current time is less than 20. Otherwise it will output "Have a good night!":

Example

```
<?php
$t = date("H");

if ($t < "10") {
   echo "Have a good morning!";
} elseif ($t < "20") {
   echo "Have a good day!";
} else {
   echo "Have a good night!";
}
?>
```

PHP - The switch Statement

Use the switch statement to **select one of many blocks of code to be executed**.

Syntax

```
switch (n) {
   case label1:
      code to be executed if n=label1;
      break;
   case label2:
      code to be executed if n=label2;
      break;
   case label3:
      code to be executed if n=label3;
      break;
   ...
   default:
      code to be executed if n is different from all labels;
}
```

how it works: First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use **break** to prevent the code from running into the next case automatically. The **default** statement is used if no match is found.

Example

```
<?php
$favcolor = "red";

switch ($favcolor) {
   case "red":
      echo "Your favorite color is red!";
```

```
        break;
    case "blue":
        echo "Your favorite color is blue!";
        break;
    case "green":
        echo "Your favorite color is green!";
        break;
    default:
        echo "Your favorite color is neither red, blue, or green!";
}
?>
```

12. Explain about cookies in PHP with example.

Cookie:

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

Create Cookies With PHP

A cookie is created with the setcookie() function.

Syntax

setcookie(**name, value, expire, path, domain, secure, httponly**);

Only the **name** parameter is required. All other parameters are optional.

PHP Create/Retrieve a Cookie

The following example creates a cookie named "user" with the value "John Doe". The cookie will expire after 30 days (86400 * 30). The "/" means that the cookie is available in entire website (otherwise, select the directory you prefer).

We then retrieve the value of the cookie "user" (using the global variable $_COOKIE). We also use the isset() function to find out if the cookie is set:

Example

```
<?php
$cookie_name = "user";
$cookie_value = "John Doe";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); // 86400 = 1 day
?>
<html>
<body>
<?php
if(!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
    echo "Cookie '" . $cookie_name . "' is set!<br>";
    echo "Value is: " . $_COOKIE[$cookie_name];
}
```

```
?>
</body>
</html>
```

Note: The setcookie() function must appear BEFORE the <html> tag.

Note: The value of the cookie is automatically URLencoded when sending the cookie, and automatically decoded when received (to prevent URLencoding, use setrawcookie() instead).

Modify a Cookie Value

To modify a cookie, just set (again) the cookie using the setcookie() function:

Example

```
<?php
$cookie_name = "user";
$cookie_value = "Alex Porter";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/");
?>
<html>
<body>
<?php
if(!isset($_COOKIE[$cookie_name])) {
   echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
   echo "Cookie '" . $cookie_name . "' is set!<br>";
   echo "Value is: " . $_COOKIE[$cookie_name];
}
?>
</body>
</html>
```

Output:

```
Cookie 'user' is set!
Value is: John Doe
```

Note: You might have to reload the page to see the new value of the cookie.

Delete a Cookie

To delete a cookie, use the setcookie() function with an expiration date in the past:

Example

```
<?php
// set the expiration date to one hour ago
setcookie("user", "", time() - 3600);
?>
<html>
<body>
<?php
echo "Cookie 'user' is deleted.";
```

```
?>
</body>
</html>
```

Output:

```
Cookie 'user' is deleted.
```

Check if Cookies are Enabled

The following example creates a small script that checks whether cookies are enabled. First, try to create a test cookie with the setcookie() function, then count the $_COOKIE array variable:

Example

```
<?php
setcookie("test_cookie", "test", time() + 3600, '/');
?>
<html>
<body>
<?php
if(count($_COOKIE) > 0) {
   echo "Cookies are enabled.";
} else {
   echo "Cookies are disabled.";
}
?>
</body>
</html>
```

Output:

```
Cookies are enabled.
```

13. Describe the data base connections in PHP with suitable example.

Connection to MySQL database using PHP

Before you can get content out of your MySQL database, you must know how to establish a connection to MySQL from inside a PHP script. To perform basic queries from within MySQL is very easy.

The first thing to do is connect to the database. The function to connect to MySQL is called mysql_connect. This function returns a resource which is a pointer to the database connection. It's also called a database handle, and we'll use it in later functions.

```
<?php
$username = "your_name";
$password = "your_password";
$hostname = "localhost";

//connection to the database
$dbhandle = mysql_connect($hostname, $username, $password)
```

```
  or die("Unable to connect to MySQL");
echo "Connected to MySQL<br>";
?>
```

All going well, you should see "Connected to MySQL" when you run this script. If you can't connect to the se sure your password, username and hostname are correct.

Once you've connected, you're going to want to select a database to work with. Let's assume the database is ca 'examples'. To start working in this database, you'll need the mysql_select_db() function:

```
<?php
//select a database to work with
$selected = mysql_select_db("examples",$dbhandle)
  or die("Could not select examples");
?>
```

Now that you're connected, let's try and run some queries. The function used to perform queries is named - my The function returns a resource that contains the results of the query, called the result set. To examine the resu going to use the mysql_fetch_array function, which returns the results row by row. In the case of a query that results, the resource that the function returns is simply a value true or false.

A convenient way to access all the rows is with a while loop. Let's add the code to our script:

```
<?php
//execute the SQL query and return records
$result = mysql_query("SELECT id, model, year FROM cars");
//fetch tha data from the database
while ($row = mysql_fetch_array($result)) {
   echo "ID:".$row{'id'}." Name:".$row{'model'}."
   ".$row{'year'}."<br>";
}
?>
```

Finally, we close the connection. Although this isn't strictly speaking necessary, PHP will automatically close connection when the script ends, you should get into the habit of closing what you open.

```
<?php
//close the connection
mysql_close($dbhandle);
?>
```

Here is a code in full:

```
<?php
$username = "your_name";
```

```
$password = "your_password";
$hostname = "localhost";

//connection to the database
$dbhandle = mysql_connect($hostname, $username, $password)
 or die("Unable to connect to MySQL");
echo "Connected to MySQL<br>";

//select a database to work with
$selected = mysql_select_db("examples",$dbhandle)
  or die("Could not select examples");

//execute the SQL query and return records
$result = mysql_query("SELECT id, model,year FROM cars");

//fetch tha data from the database
while ($row = mysql_fetch_array($result)) {
   echo "ID:".$row{'id'}." Name:".$row{'model'}."Year: ". //display the results
   $row{'year'}."<br>";
}
//close the connection
mysql_close($dbhandle);
?>
```

To create 'examples' database on your MySQL server you should run the following script:

```
CREATE DATABASE `examples`;
USE `examples`;
CREATE TABLE `cars` (
   `id` int UNIQUE NOT NULL,
   `name` varchar(40),
   `year` varchar(50),
   PRIMARY KEY(id)
);
INSERT INTO cars VALUES(1,'Mercedes','2000');
INSERT INTO cars VALUES(2,'BMW','2004');
INSERT INTO cars VALUES(3,'Audi','2001');
```

14. **Explain the steps in the PHP code for querying a database with suitable examples.**

• Create a database connection
• Select database you wish to use
• Perform a SQL query
• Do some processing on query results
• Close database connection

1. Creating Database Connection

• Use either mysql_connect or mysql_pconnect to create database connection
  – mysql_connect: connection is closed at end of script (end of page)
  – mysql_pconnect: creates persistent connection
• connection remains even after end of the page
• Parameters
  – Server – hostname of server
  – Username – username on the database
  – Password – password on the database
  – New Link (mysql_connect only) – reuse database connection created by previous call to mysql_connect
  – Client Flags
• MYSQL_CLIENT_SSL :: Use SSL
• MYSQL_CLIENT_COMPRESS :: Compress data sent to MySQL
Username and password fields imply that database password is sitting there in the source code
  – If someone gains access to source code, can compromise the database
  – Servers are sometimes configured to view PHP source code when a resource is requested with ".phps" instead of ".php"
  – One approach to avoid this: put this information in Web server config. File
• Then ensure the Web server config. file is not externally accessible

<u>2. Selecting a Database</u>
• mysql_select_db()
  – Pass it the database name
• Related:
  – mysql_list_dbs()
• List databases available
  – Mysql_list_tables()
• List database tables available
<u>3. Perform SQL Query</u>
• Create query string
  – $query = '*SQL formatted string*'
  – $query = 'SELECT * FROM *table*'
• Submit query to database for processing
  – $result = mysql_query($query);
  – For UPDATE, DELETE, DROP, etc, returns TRUE or FALSE
  – For SELECT, SHOW, DESCRIBE or EXPLAIN, $result is an identifier for the results, and does not contain the results themselves
• $result is called a "resource" in this case
• A result of FALSE indicates an error
• If there is an error
– mysql_error() returns error string from last MySQL call
<u>4. Process Results</u>
• Many functions exist to work with database results
• mysql_num_rows()
  – Number of rows in the result set
  – Useful for iterating over result set

• mysql_fetch_array()
- – Returns a result row as an array
- – Can be associative or numeric or both (default)
- – $row = mysql_fetch_array($result);
- – $row['*column name*'] :: value comes from database row with specified column name
- – $row[0] :: value comes from first field in result set

Process Results Loop

• Easy loop for processing results:

```
$result = mysql_query($qstring);
$num_rows = mysql_num_rows($result);
for ($i=0; $i<$num_rows; $i++)
 {
   $row = mysql_fetch_array($result);
  // take action on database results here
  }
```

<u>5. Closing Database Connection</u>

• mysql_close()
- – Closes database connection
- – Only works for connections opened with mysql_connect()
- – Connections opened with mysql_pconnect() ignore this call
- – Often not necessary to call this, as connections created by mysql_connect are closed at the end of the script anyway

15. **With example explain about XSL and XSLT transformation**

XSL stands for EXtensible Stylesheet Language.

<u>XSLT:</u>

- XSLT stands for XSL Transformations
- XSLT is the most important part of XSL
- XSLT transforms an XML document into another XML document
- XSLT uses XPath to navigate in XML documents
- XSLT is a W3C Recommendation.

<u>XSLT Elements</u>

Description of all the XSLT elements from the W3C Recommendation, and information about browser support.

<u>XSLT Functions</u>

XSLT includes over 100 built-in functions. There are functions for string values, numeric values, date and time comparison, node and QName manipulation, sequence manipulation, Boolean values, and more.

XSLT = XSL Transformations

XSLT is the most important part of XSL. XSLT is used to transform an XML document into another XML document, or another type of document that is recognized by a browser, like HTML and XHTML. Normally XSLT does this by transforming each XML element into an (X)HTML element. With XSLT you can add/remove elements and attributes to or from the output file. You can also rearrange and sort elements, perform tests and make decisions about which elements to hide and display, and a lot more. A common way to describe the transformation process is to say that **XSLT transforms an XML source-tree into an XML result-tree**.

XSLT Uses XPath

XSLT uses XPath to find information in an XML document. XPath is used to navigate through elements and attributes in XML documents.

Correct Style Sheet Declaration

The root element that declares the document to be an XSL style sheet is <xsl:stylesheet> or <xsl:transform>.

Note: <xsl:stylesheet> and <xsl:transform> are completely synonymous and either can be used!

The correct way to declare an XSL style sheet according to the W3C XSLT Recommendation is:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

To get access to the XSLT elements, attributes and features we must declare the XSLT namespace at the top of the document.

The xmlns:xsl="http://www.w3.org/1999/XSL/Transform" points to the official W3C XSLT namespace. If you use this namespace, you must also include the attribute version="1.0".

Start with a Raw XML Document

We want to transform the following XML document ("cdcatalog.xml") into XHTML:

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
.
.
</catalog>
```

1. Create an XSL Style Sheet

Then you create an XSL Style Sheet ("cdcatalog.xsl") with a transformation template:

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

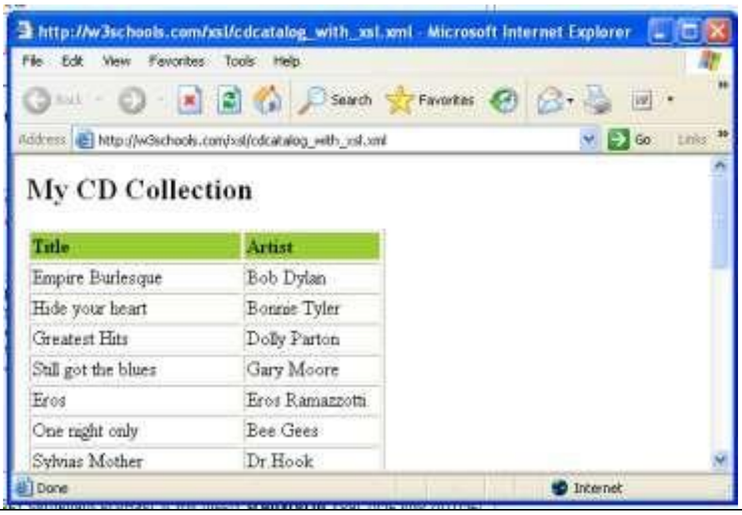
2. Link the XSL Style Sheet to the XML Document

Add the XSL style sheet reference to your XML document ("cdcatalog.xml"):

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="cdcatalog.xsl"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
.
```

.
</catalog>

The result is:



http://w3schools.com/xsl/cdcatalog_with_xsl.xml - Microsoft Internet Explorer

Address http://w3schools.com/xsl/cdcatalog_with_xsl.xml

My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tyler
Greatest Hits	Dolly Parton
Still got the blues	Gary Moore
Eros	Eros Ramazzotti
One night only	Bee Gees
Sylvias Mother	Dr.Hook

Done Internet

16. **Explain about DOM with the XML data processing.**
A huge benefit of XML – standard parsers and standard (cross-language) APIs for processing it

The DOM is a W3C (World Wide Web Consortium) standard.

The DOM defines a standard for accessing documents like XML and HTML:

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

DOM: an object-oriented representation of the XML parse tree (roughly like the Data Model graph)

- **DOM objects have methods like "getFirstChild()", "getNextSibling"**
- **Common way of traversing the tree**
- **Can also modify the DOM tree – alter the XML – via insertAfter(), etc.**

The XML DOM is:

- A standard object model for XML
- A standard programming interface for XML
- Platform- and language-independent
- A W3C standard

The XML DOM defines the **objects and properties** of all XML elements, and the **methods** (interface) to access them.

DOM Nodes

According to the DOM, everything in an XML document is a **node**.
The DOM says:

- The entire document is a document node
- Every XML element is an element node

- The text in the XML elements are text nodes
- Every attribute is an attribute node
- Comments are comment nodes

DOM Example

Look at the following XML file (books.xml):

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
 <book category="cooking">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
 </book>
 <book category="children">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
 </book>
 <book category="web">
  <title lang="en">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
  <year>2003</year>
  <price>49.99</price>
 </book>
 <book category="web" cover="paperback">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
 </book>
</bookstore>
```

The root node in the XML above is named <bookstore>. All other nodes in the document are contained within <bookstore>.

The root node <bookstore> holds four <book> nodes.

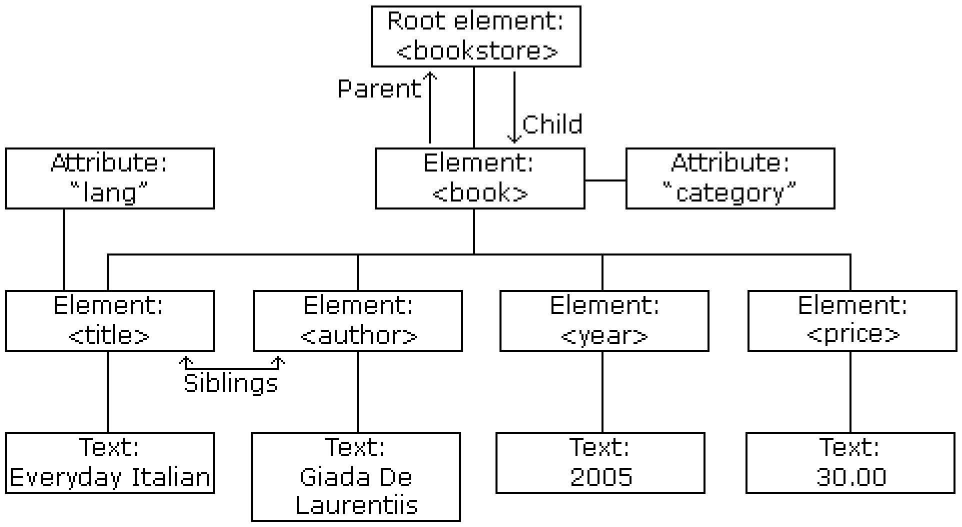
The first <book> node holds four nodes: <title>, <author>, <year>, and <price>, which contains one text node each, "Everyday Italian", "Giada De Laurentiis", "2005", and "30.00".

Text is Always Stored in Text Nodes

A common error in DOM processing is to expect an element node to contain text. However, the text of an element node is stored in a text node. In this example: **<year>2005</year>**, the element node <year>, holds a text node with the value "2005". "2005" is **not** the value of the <year> element!

The XML DOM Node Tree

The XML DOM views an XML document as a tree-structure. The tree structure is called a **node-tree.** All nodes can be accessed through the tree. Their contents can be modified or deleted, and new elements can be created.
The node tree shows the set of nodes, and the connections between them. The tree starts at the root node and branches out to the text nodes at the lowest level of the tree:

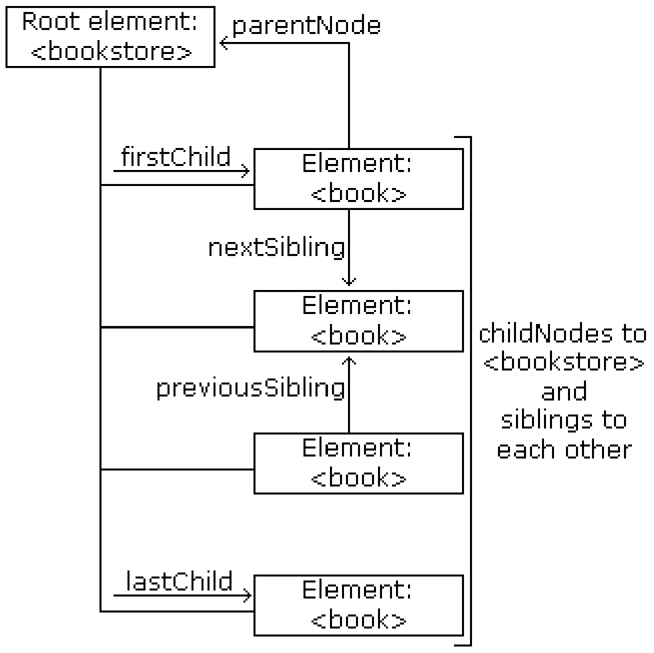


Node Parents, Children, and Siblings

The nodes in the node tree have a hierarchical relationship to each other.
The terms parent, child, and sibling are used to describe the relationships. Parent nodes have children. Children on the same level are called siblings (brothers or sisters).

- In a node tree, the top node is called the root
- Every node, except the root, has exactly one parent node
- A node can have any number of children
- A leaf is a node with no children
- Siblings are nodes with the same parent

The following image illustrates a part of the node tree and the relationship between the nodes:



XML Parser

The XML DOM contains methods to traverse XML trees, access, insert, and delete nodes.
However, before an XML document can be accessed and manipulated, it must be loaded into an XML DOM object.
An XML parser reads XML, and converts it into an XML DOM object that can be accessed with JavaScript.
Most browsers have a built-in XML parser.

Load an XML Document

The following JavaScript fragment loads an XML document ("books.xml"):

```
if (window.XMLHttpRequest)
  {
  xhttp=new XMLHttpRequest();
  }
else // code for IE5 and IE6
  {
  xhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
xhttp.open("GET","books.xml",false);
xhttp.send();
xmlDoc=xhttp.responseXML;
```

The following code loads and parses an XML string:

```
if (window.DOMParser)
  {
  parser=new DOMParser();
  xmlDoc=parser.parseFromString(text,"text/xml");
```

```
 }
else // code for IE
 {
 xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
 xmlDoc.async=false;
 xmlDoc.loadXML(text);
 }
```

Accessing Nodes

You can access a node in three ways:
1. By using the getElementsByTagName() method
2. By looping through (traversing) the nodes tree.
3. By navigating the node tree, using the node relationships.

The getElementsByTagName() Method

getElementsByTagName() returns all elements with a specified tag name.

Syntax

node.getElementsByTagName(*"tagname"*);

Example

The following example returns all <title> elements under the x element:

```
x.getElementsByTagName("title");
```

Note that the example above only returns <title> elements under the x node. To return all <title> elements in the XML document use:

```
xmlDoc.getElementsByTagName("title");
```

where xmlDoc is the document itself (document node).

DOM Node List

The getElementsByTagName() method returns a node list. A node list is an array of nodes.
The following code loads "books.xml" into xmlDoc using loadXMLDoc() and stores a list of <title> nodes (a node list) in the variable x:

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("title");
```

The <title> elements in x can be accessed by index number. To access the third <title> you can write::

```
y=x[2];
```

Note: The index starts at 0.
You will learn more about node lists in a later chapter of this tutorial.

DOM Node List Length

The length property defines the length of a node list (the number of nodes).
You can loop through a node list by using the length property:

Example

```
var xmlDoc=loadXMLDoc("books.xml");
var x=xmlDoc.getElementsByTagName("title");
for (i=0;i<x.length;i++)
 {
```

```
 document.write(x[i].childNodes[0].nodeValue);
 document.write("<br>");
 }
```

Node Types

The **documentElement** property of the XML document is the root node.
The **nodeName** property of a node is the name of the node.
The **nodeType** property of a node is the type of the node.
You will learn more about the node properties in the next chapter of this tutorial.

Traversing Nodes

The following code loops through the child nodes, that are also element nodes, of the root node:

Example

```
var xmlDoc=loadXMLDoc("books.xml");
var x=xmlDoc.documentElement.childNodes;

for (i=0;i<x.length;i++)
 {
 // Process only element nodes (type 1)
 if (x[i].nodeType==1)
  {
  document.write(x[i].nodeName);
  document.write("<br>");
  }
 }
```

Example explained:

1. Load "books.xml" into xmlDoc using loadXMLDoc()
2. Get the child nodes of the root element
3. For each child node, check the node type of the node. If the node type is "1" it is an element node
4. Output the name of the node if it is an element node

Navigating Node Relationships

The following code navigates the node tree using the node relationships:

Example

```
var xmlDoc=loadXMLDoc("books.xml");

var x=xmlDoc.getElementsByTagName("book")[0].childNodes;
var y=xmlDoc.getElementsByTagName("book")[0].firstChild;

for (i=0;i<x.length;i++)
 {
 // Process only element nodes (type 1)
 if (y.nodeType==1)
```

```
  {
  document.write(y.nodeName + "<br>");
  }
 y=y.nextSibling;
 }
```

1. Load "books.xml" into xmlDoc using loadXMLDoc()
2. Get the child nodes of the first book element
3. Set the "y" variable to be the first child node of the first book element
4. For each child node (starting with the first child node "y"):
5. Check the node type. If the node type is "1" it is an element node
6. Output the name of the node if it is an element node
7. Set the "y" variable to be the next sibling node, and run through the loop again

Traversing the Node Tree

Often you want to loop an XML document, for example: when you want to extract the value of each element.
This is called "Traversing the node tree"
The example below loops through all child nodes of <book>, and displays their names and values:

Example

```
<html>
<head>
<script src="loadxmlstring.js"></script>
</head>
<body>
<script>
var text="<book>";
text=text+"<title>Everyday Italian</title>";
text=text+"<author>Giada De Laurentiis</author>";
text=text+"<year>2005</year>";
text=text+"</book>";

var xmlDoc=loadXMLString(text);

// documentElement always represents the root node
var x=xmlDoc.documentElement.childNodes;

for (i=0;i<x.length;i++)
 {
 document.write(x[i].nodeName);
 document.write(": ");
 document.write(x[i].childNodes[0].nodeValue);
 document.write("<br>");
 }
</script>
</body>
```

```
</html>
```

Output:
title: Everyday Italian
author: Giada De Laurentiis
year: 2005

Example explained:

1. loadXMLString() loads the XML string into xmlDoc
2. Get the child nodes of the root element
3. For each child node, output the node name and the node value of the text node

17. **Discuss in detail about the XML DTD**

XML DTD

An XML document with correct syntax is called "Well Formed".
An XML document validated against a DTD is "Well Formed" and "Valid".

Valid XML Documents

A "Valid" XML document is a "Well Formed" XML document, which also conforms to the rules of a DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note SYSTEM "Note.dtd">
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

The DOCTYPE declaration, in the example above, is a reference to an external DTD file. The content of the file is shown in the paragraph below.

The purpose of a DTD is to define the structure of an XML document. It defines the structure with a list of legal elements:

```
<!DOCTYPE note
[
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

The DTD above is interpreted like this:

- !DOCTYPE note defines that the root element of the document is note
- !ELEMENT note defines that the note element must contain four elements: "to, from, heading, body"
- !ELEMENT to defines the to element to be of type "#PCDATA"
- !ELEMENT from defines the from element to be of type "#PCDATA"
- !ELEMENT heading defines the heading element to be of type "#PCDATA"
- !ELEMENT body defines the body element to be of type "#PCDATA"

#PCDATA means parse-able text data.

Using DTD for Entity Declaration

A doctype declaration can also be used to define special characters and character strings, used in the document:

Example

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE note [
<!ENTITY nbsp "&#xA0;">
<!ENTITY writer "Writer: Donald Duck.">
<!ENTITY copyright "Copyright: W3Schools.">
]>

<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
<footer>&writer;&nbsp;&copyright;</footer>
</note>
```

With a DTD, independent groups of people can agree on a standard for interchanging data. With a DTD, you can verify that the data you receive from the outside world is valid.

	Unit-V
	Part – A
1.	**What is Ajax?** Ajax is a set of client side technologies that provides asynchronous communication between user interfaces and web server. So the advantages of using Ajax are asynchronous communication, minimal data transfer and server is not overloaded with unnecessary load.
2.	**What technologies are being used in AJAX?** AJAX uses four technologies, which are as follows: JavaScript, XMLHttpRequest, Document Object Model (DOM), Extensible HTML (XHTML) and Cascading Style Sheets (CSS)
3.	**Explain the limitations of AJAX.**

	It is difficult to bookmark a particular state of the application,Function provided in the code-behind file do not work because the dynamic pages cannot register themselves on browsers history engine automatically
4.	**Describe AJAX Control Extender Toolkit.** AJAX Control Toolkit is a set of extenders that are used to extend the functionalities of the ASP.NET controls. The extenders use a block of JavaScript code to add new and enhanced capabilities to the ASP.NET controls. AJAX Control Toolkit is a free download available on the Microsoft site. You need to install this toolkit on your system before using extenders.
5.	**30) What is the syntax to create AJAX objects?** AJAX uses the following syntax to create an object: Var myobject = new AjaxObject("page path"); The page path is the URL of the Web page containing the object that you want to call. The URL must be of the same domain as the Web page.
6.	**How can you find out that an AJAX request has been completed?** You can find out that an AJAX request has been completed by using the readyState property. If the value of this property equals to four, it means that the request has been completed and the data is available.
7.	**What are the different ways to pass parameters to the server?** We can pass parameters to the server using either the GET or POST method. The following code snippets show the example of both the methods: Get: XmlHttpObject.Open("GET","file1.txt", true); Post: XmlHttpObject.Open("POST", "file2.txt", true);
8.	**What are the extender controls?** The extender controls uses a block of JavaScript code to add new and enhanced capabilities to ASP.NET. The developers can use a set of sample extender controls through a separate download - AJAX Control Toolkit (ACT).
9.	**List out the advantages of AJAX. (May 2014)** • Better interactivity • Easier navigation • Compact • Backed by reputed brands
10.	**Define Web service? (Nov 2011)** A Web service is a method of communication between two electronic devices over the web. The W3C defines a "Web service" as "a software system designed to support interoperable machine-to-machine interaction over a network". It has an interface described in a machine-processable format specifically Web Services Description Language (WSDL).
11.	**What are the different applications that could use web services??** ❖ **Data providers**, for example, those that provide data such as a stock quote ❖ **Business-to-business process integrations**, such as those that send a purchase order from one company to another ❖ **Integration with multiple partners**, and even with competitors ❖ **Enterprise application integration**, for example, integration of a company's e-mail database with its human resources (HR) database
12.	**What are the features of web service?** Web services are having the features such as heterogeneous, interoperable, loosely coupled, and implementation-independent programs and modular design
13.	**What are the rules to be followed in designing the web service?** ❖ Allow extensibility points. ❖ Keep your namespaces easy to version by placing dates in them. ❖ Don't try to solve every problem with one schema, WSDL, or other file. Break out the problem into pieces
14.	**What is meant by WSDL? (APR/MAY 2011)** • WSDL stands for Web Services Description Language • WSDL is based on XML

- WSDL is used to describe Web services
- WSDL is used to locate Web services
- WSDL is an XML-based language for locating and describing Web services

15. Why do you want to describe a web service? (MAY/JUNE 2014)

Web Services Description Language (WSDL) is a document written in XML. The document describes a Web service. It specifies the location of the service and the operations (or methods) the service exposes.

16. 17. What are the elements of WSDL?

Element Name	Description
types	A container for abstract type definitions defined using XML Schema
message	A definition of an abstract message that may consist of multiple parts, each part may be of a different type
portType	An abstract set of operations supported by one or more endpoints (commonly known as an interface); operations are defined by an exchange of messages
binding	A concrete protocol and data format specification for a particular portType
service	A collection of related endpoints, where an endpoint is defined as a combination of a binding and an address (URI)

18. What is the use of web services?

- Web services encompass a set of related standards that can enable two computers
- The data is passed back and forth using standard protocols such as HTTP, the same protocol used to transfer ordinary web pages.
- Web services operate using open, text-based standards that enable components written in different languages and on different platforms to communicate.
- They are ready to use pieces of software on the Internet. XML, SOAP, Web Services Description Language (WSDL) and Universal Description, Discovery and Integration (UDDI) are the standards on which web services rely.
- UDDI is another XML based format that enables developers and business to publish and locate Web services on a network.

19. State the uses of WSDL. (APR/MAY 2012)

- WSDL stands for Web Services Description Language.
- WSDL is a document written in XML.
- WSDL is an XML-based language for locating and describing Web services.

20. What are the four transmission types of WSDL?

- ❖ One-way
- ❖ Request–response
- ❖ Solicit–response
- ❖ Notification

21. State the significance of a WSDL document. (NOV/DEC 2012)

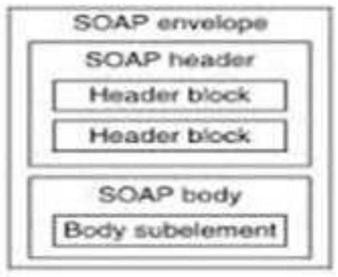
The WSDL is a Web Service Descriptor Language which is based on XML.

ELEMENT	DESCRIPTION
Types	It Specifies the data types of the symbols used by the web services.
Messages	It specifies the messages used by the web services.
Porttype	It specifies the name of the operations
Binding	It specifies the name of the protocol of the web services, typically it is SOAP.

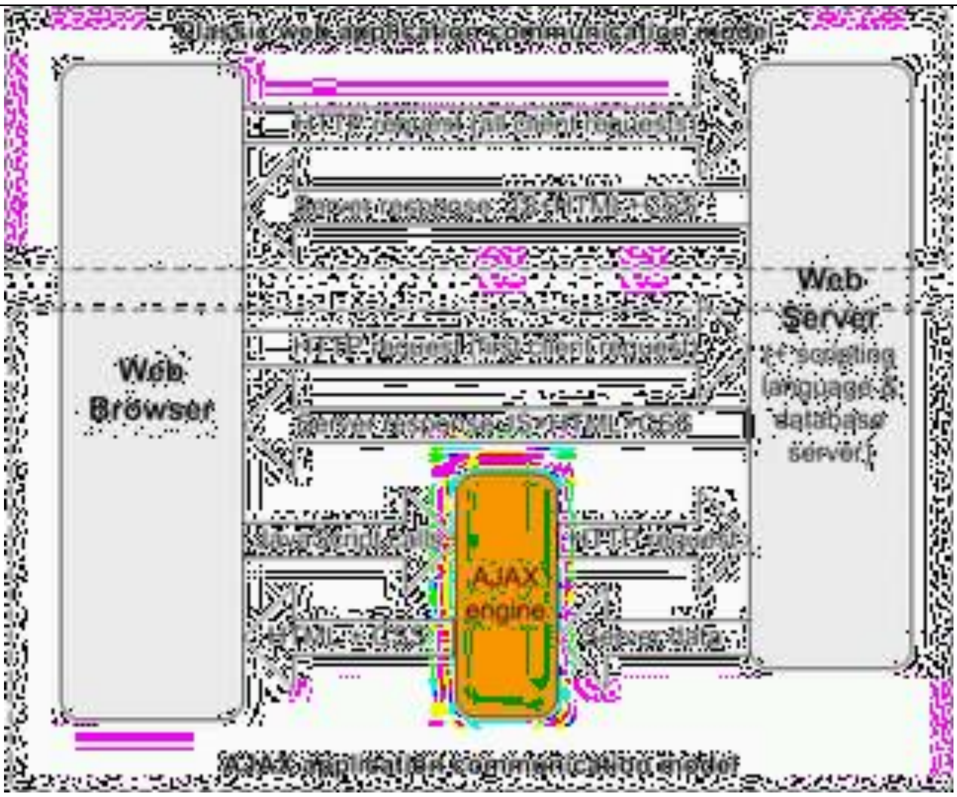
22. What is UDDI? (NOV/DEC 2011)

UDDI means Universal Description, Discovery and Integration.

UDDI - platform-independent framework for describing services, discovering businesses, and integrating business

	services by using the Internet. - directory for storing information about web services - directory of web service interfaces described by WSDL - communicates via SOAP - The core of UDDI is the UDDI Business Registry, a global, pubic, online directory.
23.	**What are the benefits of UDDI?** Problems the UDDI specification can help to solve: • Making it possible to discover the right business from the millions currently online • Defining how to enable commerce once the preferred business is discovered • Reaching new customers and increasing access to current customers • Expanding offerings and extending market reach • Solving customer-driven need to remove barriers to allow for rapid participation in the global Internet economy • Describing services and business processes programmatically in a single, open, and secure environment
24.	**What are the core elements of UDDI?** UDDI defines four core data elements within the data model: • businessEntity (modeling business information) • businessService (describing a service) • tModel (describing specifications, classifications, or identifications) • binding Template (mapping between a businessService and the set of tModels that describe its technical fingerprint)
25.	**List some examples of web services. (APR/MAY 2012)** • Geo IP: http://www.webservicex.net/geoipservice.asmx?op=GetGeoIP • Whois: http://www.webservicex.net/whois.asmx?op=GetWhoIS SMS: http://www.webservicex.net/sendsmsworld.asmx
26.	**List out some web service technologies?** • XML • SOAP • WSDL
27.	**What is SOAP?** **SOAP - Simple Object Access Protocol** - protocol specification for exchanging structured information in the implementation of Web Services in computer networks. - relies on Extensible Markup Language (XML) for its message format, and usually relies on other Application Layer protocols, most notably Hypertext Transfer Protocol (HTTP) and Simple Mail Transfer Protocol (SMTP), for message negotiation and transmission.
28.	**Define SOAP structure.**  SOAP envelope SOAP header Header block Header block SOAP body Body subelement **The SOAP envelope** <Envelope> is the root element in every SOAP message, and contains two child elements, an optional <Header> element, and a mandatory <Body> element. **The SOAP header**

	<Header> is an optional sub element of the SOAP envelope, and is used to pass application-related information that is to be processed by SOAP nodes along the message path; see The SOAP header. **The SOAP body** <Body> is a mandatory sub element of the SOAP envelope, which contains information intended for the ultimate recipient of the message; see The SOAP body. **The SOAP fault** <Fault> is a sub element of the SOAP body, which is used for reporting errors; see The SOAP fault. XML elements in <Header> and <Body> are defined by the applications that make use of them, although the SOAP specification imposes some constraints on their structure.
29.	**Define the need for SOAP. (APR/MAY 2013)** Simple Object Access Protocol (SOAP) is a protocol based on XML. It is used by the web services for exchange of information. The Client- Server communication is based on RPC. The HTTP does not design to handle the distributed objects that are required by the RPC. Hence another application protocol is build over HTTP which popularly known as SOAP. SOAP allows talking different applications that are running in two different operating systems.
30.	**What are the descriptions in SOAP service?** To describe everything a SOAP service needs to describe the following: • The operations • The schema for each message in an operation • The SOAPAction headers • The URL endpoint of the service
31.	**Give an example of a web services registry and its function. (NOV/DEC 2012)** It refers to a place in which service providers can impart information about their offered services and potential clients can search for services *Example*: IBM - WebSphere Service Registry, Oracle Service Registry etc.,
32.	**Mention some of the disadvantageous of web services (MAY/JUNE 2014)** Web services standards features such as transactions are currently nonexistent or still in their infancy compared to more mature distributed computing open standards such as CORBA. Web services may suffer from poor performance compared to other distributed computing approaches such as RMI, CORBA, or DCOM.
33.	**What is JWSDP?** Java Web Service Developer Pack (JWSDP) is a tool. Using the JWSDP tool the simple implementation files written in java can be converted to Web Service.
34.	**What are the specifications of web service architecture?** The specifications are ❖ Standards based ❖ Modular ❖ Federated ❖ General purpose
	Part-B
1.	**Explain about the object that helps AJAX reload parts of a web page without reloading the whole page. (NOV/DEC 2011, MAY/JUNE 2014)** AJAX = Asynchronous JavaScript and XML. AJAX is a technique for creating fast and dynamic web pages. AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page. Classic web pages, (which do not use AJAX) must reload the entire page if the content should change. **AJAX is about updating parts of a web page, without reloading the whole page**.



Ajax request and response objects

The **core of AJAX is the XMLHttpRequest object** (available in client side scripting languages like javascript). The XMLHttpRequest object is used to exchange data with a server behind the scenes. All modern browsers (IE7+, Firefox, Chrome, Safari, and Opera) have a built-in XMLHttpRequest object. **If you are using IE 5 or IE6** (I wonder if someone still uses it), then **ActiveXObject will be used** for server communication to send ajax requests.

A new object of XMLHttpRequest is created like this :

```
//Creating a new XMLHttpRequest object
var xmlhttp;
if (window.XMLHttpRequest)
{
    xmlhttp = new XMLHttpRequest(); //for IE7+, Firefox,
Chrome, Opera, Safari
}
else
{
    xmlhttp = new ActiveXObject("Microsoft.XMLHTTP"); //for
IE6, IE5
}
```

This xmlhttp variable can be re-used to send multiple ajax requests, without creating new objects. **XMLHttpRequest is subject to the browser's same origin policy for security reasons**. It means that requests will only succeed if they are made to the same server that served the original web page.

Useful methods to work with XMLHttpRequest

To send request and set request attributes, XMLHttpRequest object has some methods.

a) open(method, url, isAsync, userName, password)
The HTTP and HTTPS requests of the **XMLHttpRequest object must be initialized through the open method**. This method specifies the type of request (GET, POST etc.), the URL, and if the request should be handled asynchronously or not. I will cover this third parameter in next section.
The fourth and fifth parameters are the username and password, respectively. These parameters, or just the username, may be provided for authentication and authorization if required by the server for this request.
Example:

```
xmlhttp.open("GET","report_data.xml",true);
xmlhttp.open("GET","sensitive_data.xml",false);
xmlhttp.open("POST","saveData",true,"myUserName","som
ePassord");
```

b) setRequestHeader(name, value)
Upon successful initialization of a request, the setRequestHeader method of the XMLHttpRequest object can be invoked **to send HTTP headers with the request**.
Example:

```
//Tells server that this call is made for ajax
purposes.
xmlhttp.setRequestHeader('X-Requested-With',
'XMLHttpRequest');
```

c) send(payload)
To send an HTTP request, the send method of the XMLHttpRequest must be invoked. This method accepts a single parameter containing the content to be sent with the request.
The content is necessary in POST requests. For GET methods, imply pass null as parameter.
Example:

```
xmlhttp.send(null); //Request with no data in request body;
Mostly used in GET requests.
xmlhttp.send( {"id":"23423"} ); //Request with data in request
body; Mostly used in POST/ PUT requests.
```

4) abort()
This method **aborts the request if the readyState of the XMLHttpRequest object has not yet become 4** (request complete). The abort method ensures that the callback method does not get invoked in an asynchronous request.
Syntax:

```
//Abort the processing
 xmlhttp.abort();
```

Apart from above method, onreadystatechange event listener is very important which we will discuss in next section.

__**Synchronous and Asynchronous requests**__
XMLHttpRequest object is capable of sending synchronous and asynchronous requests, as required within webpage. The behavior is controlled by **third parameter of open method**. This parameter is set to **true for an asynchronous requests, and false for synchronous requests**.

```
 xmlhttp.open("GET", "report_data.xml", true); //Asynchrnonos
1request as third parameter is true
2xmlhttp.open("GET", "report_data.xml", false); Synchrnonos
 request as third parameter is false
```

The **default value of this parameter is "true"** if it is not provided.

Asynchronous Ajax Requests do not block the webpage and user can continue to interact with other elements on the page, while the request is processed on server. You should always use asynchronous Ajax Requests because a synchronous Ajax Request makes the UI (browser) unresponsive. It means user will not be able to interact with the webpage, until the request is complete.

Synchronous requests should be used in rare cases with utmost care. For example, synchronous Ajax Request should be used if you're embedding a new JavaScript file on the client using ajax and then referencing types and/or objects from that JavaScript file. Then the fetching of this new JS file should be included through using a synchronous Ajax Request.

Example synchronous request

```
var request = new XMLHttpRequest();
request.open('GET', '/bar/foo.txt', false);   //"false" makes
the request synchronous
request.send(null);

if(request.status === 200)
{
    //request successful; handle the response
}
else
{
    //Request failed; Show error message
}
```

Example asynchronous request

```
var request = new XMLHttpRequest();
request.open('GET', '/bar/foo.txt', true);   //"true" makes the
request asynchronous

request.onreadystatechange = function() {
    if (request.readyState == 4) {
        if(request.status == 200)
        {
            //request succeed
        }
        else
        {
            //request failed
        }
    }
};
```

```
request.send(null)
```

In above example, onreadystatechange is a event listener registered with XMLHttpRequest request. onreadystatechange stores a function that will process the response returned from the server. It will be called for all important events in request's life cycle. Every time an step is completed in request processing, the value of readyState will be changed and set to some other value.
0 : request not initialized
1 : server connection established
2 : request received
3 : processing request
4 : request finished and response is ready to be handled

Handling returned response from server

To get the response from a server, responseText or responseXML property of the XMLHttpRequest object is used. If the response from the server is XML, and you want to parse it as an XML object, use the responseXML property. If the response from the server is not XML, use the responseText property.
responseText : Get the response from server as a string
responseXML : Get the response from server as XML

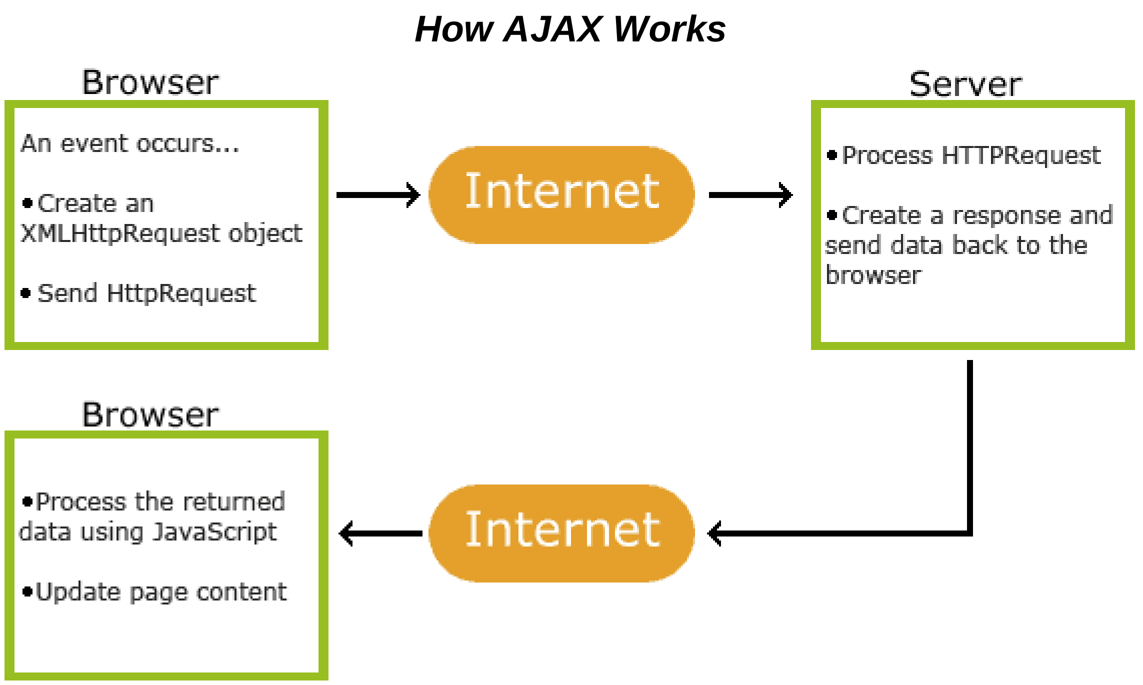
Example code:

```
if(xmlhttp.readyState == 4) {
     if(xmlhttp.status == 200)
     {
          document.getElementById("message").innerHTML =
xmlhttp.responseText;
     }
     else{
          alert('Something is wrong !!');
     }
}
```

2. Explain technologies are being used in AJAX?

- AJAX stands for **A**synchronous **Ja**vaScript and **X**ML. AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script.
- Ajax uses XHTML for content, CSS for presentation, along with Document Object Model and JavaScript for dynamic content display.
- Conventional web applications transmit information to and from the sever using synchronous requests. It means you fill out a form, hit submit, and get directed to a new page with new information from the server.
- With AJAX, when you hit submit, JavaScript will make a request to the server, interpret the results, and update the current screen. In the purest sense, the user would never know that anything was even transmitted to the server.
- XML is commonly used as the format for receiving server data, although any format, including plain text, can be used.
- AJAX is a web browser technology independent of web server software.
- A user can continue to use the application while the client program requests information from the server in the background.

- Intuitive and natural user interaction. Clicking is not required, mouse movement is a sufficient event trigger.
- Data-driven as opposed to page-driven.


How AJAX Works
Browser
An event occurs...
• Create an XMLHttpRequest object
• Send HttpRequest
Internet
Server
• Process HTTPRequest
• Create a response and send data back to the browser
Browser
• Process the returned data using JavaScript
• Update page content
Internet

AJAX is Based on Open Standards

- AJAX is based on the following open standards:
  - Browser-based presentation using HTML and Cascading Style Sheets (CSS).
  - Data is stored in XML format and fetched from the server.
  - Behind-the-scenes data fetches using XMLHttpRequest objects in the browser.
  - JavaScript to make everything happen.

AJAX cannot work independently. <u>It is used in combination with other technologies to create interactive webpages.</u>

JavaScript

- Loosely typed scripting language.
- JavaScript function is called when an event occurs in a page.
- Glue for the whole AJAX operation.

DOM

- API for accessing and manipulating structured documents.
- Represents the structure of XML and HTML documents.

CSS

- Allows for a clear separation of the presentation style from the content and may be changed programmatically by JavaScript.

XMLHttpRequest

- JavaScript object that performs asynchronous interaction with the server.

Steps of AJAX Operation

1. A client event occurs.
2. An XMLHttpRequest object is created.
3. The XMLHttpRequest object is configured.
4. The XMLHttpRequest object makes an asynchronous request to the Webserver.
5. The Webserver returns the result containing XML document.
6. The XMLHttpRequest object calls the callback() function and processes the result.
7. The HTML DOM is updated.

1. A Client Event Occurs

- A JavaScript function is called as the result of an event.
- Example: *validateUserId()* JavaScript function is mapped as an event handler to an *onkeyup* event on input form field whose id is set to *"userid"*
- <input type="text" size="20" id="userid" name="id" onkeyup="validateUserId();">.

2. The XMLHttpRequest Object is Created

```
var ajaxRequest;  // The variable that makes Ajax possible!
function ajaxFunction(){
   try{

      // Opera 8.0+, Firefox, Safari
      ajaxRequest = new XMLHttpRequest();
   }catch (e){

      // Internet Explorer Browsers
      try{
         ajaxRequest = new ActiveXObject("Msxml2.XMLHTTP");
      }catch (e) {

         try{
            ajaxRequest = new ActiveXObject("Microsoft.XMLHTTP");
         }catch (e){

            // Something went wrong
            alert("Your browser broke!");
            return false;
         }
      }
   }
}
```

3. The XMLHttpRequest Object is Configured

In this step, we will write a function that will be triggered by the client event and a callback function processRequest() will be registered.

```
function validateUserId() {
   ajaxFunction();

   // Here processRequest() is the callback function.
   ajaxRequest.onreadystatechange = processRequest;

   if (!target) target = document.getElementById("userid");
   var url = "validate?id=" + escape(target.value);

   ajaxRequest.open("GET", url, true);
   ajaxRequest.send(null);
}
```

4. Making Asynchronous Request to the Webserver

Source code is available in the above piece of code. Code written in bold typeface is responsible to make a request to the webserver. This is all being done using the XMLHttpRequest object *ajaxRequest*.

```
function validateUserId() {
   ajaxFunction();

   // Here processRequest() is the callback function.
   ajaxRequest.onreadystatechange = processRequest;

   if (!target) target = document.getElementById("userid");
   var url = "validate?id=" + escape(target.value);

   ajaxRequest.open("GET", url, true);
   ajaxRequest.send(null);
}
```

Assume you enter *Zara* in the userid box, then in the above request, the URL is set to "validate?id=Zara".

5. Webserver Returns the Result Containing XML Document

You can implement your server-side script in any language, however its logic should be as follows.

- Get a request from the client.
- Parse the input from the client.
- Do required processing.
- Send the output to the client.

If we assume that you are going to write a servlet, then here is the piece of code.

```
public void doGet(HttpServletRequest request, HttpServletResponse
response) throws IOException, ServletException
{
   String targetId = request.getParameter("id");

   if ((targetId != null) && !accounts.containsKey(targetId.trim()))
   {
      response.setContentType("text/xml");
      response.setHeader("Cache-Control", "no-cache");
      response.getWriter().write("true");
   }
   else
   {
      response.setContentType("text/xml");
      response.setHeader("Cache-Control", "no-cache");
      response.getWriter().write("false");
   }
}
```

6. Callback Function processRequest() is Called

The XMLHttpRequest object was configured to call the processRequest() function when there is a state change to the *readyState* of the *XMLHttpRequest* object. Now this function will receive the result from the server and will do the required processing. As in the following example, it sets a variable message on true or false based on the returned value from the Webserver.

```
function processRequest() {
   if (req.readyState == 4) {
      if (req.status == 200) {
         var message = ...;
...
}
```

7. The HTML DOM is Updated

This is the final step and in this step, your HTML page will be updated. It happens in the following way:

- JavaScript gets a reference to any element in a page using DOM API.
- The recommended way to gain a reference to an element is to call.

```
document.getElementById("userIdMessage"),
// where "userIdMessage" is the ID attribute
// of an element appearing in the HTML document
```

- JavaScript may now be used to modify the element's attributes; modify the element's style properties; or add, remove, or modify the child elements. Here is an example:

```
<script type="text/javascript">
<!--
function setMessageUsingDOM(message) {
   var userMessageElement = document.getElementById("userIdMessage");
   var messageText;

   if (message == "false") {
      userMessageElement.style.color = "red";
      messageText = "Invalid User Id";
   }
   else
   {
      userMessageElement.style.color = "green";
      messageText = "Valid User Id";
   }

   var messageBody = document.createTextNode(messageText);

   // if the messageBody element has been created simple
   // replace it otherwise append the new element
   if (userMessageElement.childNodes[0]) {
      userMessageElement.replaceChild(messageBody,
userMessageElement.childNodes[0]);
   }
   else
   {
      userMessageElement.appendChild(messageBody);
   }
}
-->
</script>
<body>
<div id="userIdMessage"><div>
</body>
```

3. Explain the concept of JSON concept with example.

JSON: **J**ava**S**cript **O**bject **N**otation.

- JSON is a syntax for storing and exchanging data.
- JSON is an easier-to-use alternative to XML.
- JSON is a lightweight data-interchange format
- JSON is language independent
- JSON is "self-describing" and easy to understand

The following JSON example defines an employees object, with an array of 3 employee records:

JSON Example

```
{"employees":[
   {"firstName":"John", "lastName":"Doe"},
   {"firstName":"Anna", "lastName":"Smith"},
   {"firstName":"Peter", "lastName":"Jones"}
]}
```

The following XML example also defines an employees object with 3 employee records:

XML Example

```
<employees>
   <employee>
      <firstName>John</firstName> <lastName>Doe</lastName>
   </employee>
   <employee>
      <firstName>Anna</firstName> <lastName>Smith</lastName>
   </employee>
   <employee>
      <firstName>Peter</firstName> <lastName>Jones</lastName>
   </employee>
</employees>
```

JSON Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JSON Object Creation in JavaScript</h2>

<p id="demo"></p>

<script>
var text = '{"name":"John Johnson","street":"Oslo West 16","phone":"555 1234567"}';

var obj = JSON.parse(text);

document.getElementById("demo").innerHTML =
obj.name + "<br>" +
obj.street + "<br>" +
obj.phone;
</script>

</body>
</html>
```

Output:

JSON Object Creation in JavaScript

```
John Johnson
Oslo West 16
555 1234567
```

A common use of JSON is to read data from a web server, and display the data in a web page.

The following example reads a menu from **myTutorials.txt**, and displays the menu in a web page:

JSON Example

```
<div id="id01"></div>

<script>
var xmlhttp = new XMLHttpRequest();
var url = "myTutorials.txt";

xmlhttp.onreadystatechange = function() {
  if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
    var myArr = JSON.parse(xmlhttp.responseText);
    myFunction(myArr);
    }
}
xmlhttp.open("GET", url, true);
xmlhttp.send();

function myFunction(arr) {
    var out = "";
    var i;
    for(i = 0; i < arr.length; i++) {
        out += '<a href="' + arr[i].url + '">' +
        arr[i].display + '</a><br>';
    }
    document.getElementById("id01").innerHTML = out;
}
</script>
```

Example Explained

1: Create an array of objects.

Use an **array literal** to declare an **array** of **objects**.
Give each object two properties: **display** and **url**.
Name the array **myArray**:

myArray

```
var myArray = [
{
"display": "JavaScript Tutorial",
"url": "http://www.w3schools.com/js/default.asp"
},
{
"display": "HTML Tutorial",
"url": "http://www.w3schools.com/html/default.asp"
},
{
"display": "CSS Tutorial",
"url": "http://www.w3schools.com/css/default.asp"
}
]
```

2: Create a JavaScript function to display the array.

Create a function **myFunction()** that loops the array objects, and display the content as HTML links:

myFunction()

```
function myFunction(arr) {
    var out = "";
    var i;
    for(i = 0; i < arr.length; i++) {
        out += '<a href="' + arr[i].url + '">' + arr[i].display + '</a><br>';
    }
```

```
   document.getElementById("id01").innerHTML = out;
}
```

Call **myFunction()** with **myArray** as argument:

Example

```
myFunction(myArray);
```

3: Create a text file

Put the **array literal** in a file named **myTutorials.txt**:

myTutorials.txt

```
[
{
"display": "JavaScript Tutorial",
"url": "http://www.w3schools.com/js/default.asp"
},
{
"display": "HTML Tutorial",
"url": "http://www.w3schools.com/html/default.asp"
},
{
"display": "CSS Tutorial",
"url": "http://www.w3schools.com/css/default.asp"
}
]
```

4: Read the text file with an XMLHttpRequest

Write an **XMLHttpRequest** to read the text file, and use **myFunction()** to display the array:

XMLHttpRequest

```
var xmlhttp = new XMLHttpRequest();
var url = "myTutorials.txt";

xmlhttp.onreadystatechange = function() {
if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
   var myArr = JSON.parse(xmlhttp.responseText);
   myFunction(myArr);
   }
}

xmlhttp.open("GET", url, true);
xmlhttp.send();
```

4. **Explain about Ajax Client Server Architecture.**

A Microsoft Ajax Web application consists of either a client-only solution or a client and server solution. A client-only solution uses Microsoft Ajax Library but does not use any ASP.NET server controls. For instance, an HTML can include script elements that reference the Microsoft Ajax Library .js files. The Microsoft Ajax Library allows Ajax applications to perform all processing on the client. A client and server solution consists of using both the Microsoft Ajax Library and ASP.NET server controls.

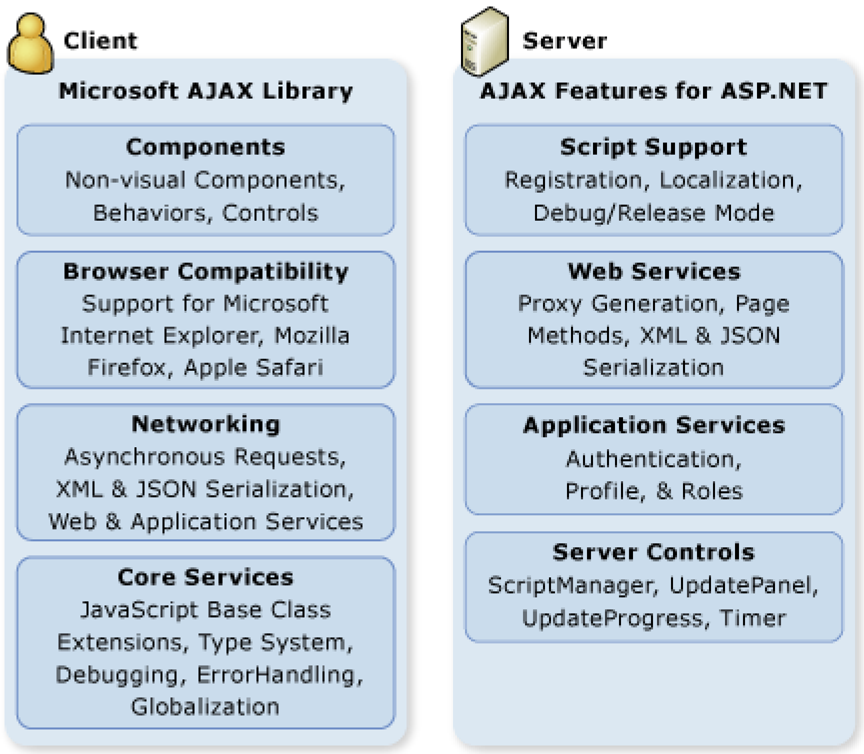

Client
Microsoft AJAX Library
Components
Non-visual Components, Behaviors, Controls
Browser Compatibility
Support for Microsoft Internet Explorer, Mozilla Firefox, Apple Safari
Networking
Asynchronous Requests, XML & JSON Serialization, Web & Application Services
Core Services
JavaScript Base Class Extensions, Type System, Debugging, ErrorHandling, Globalization
Server
AJAX Features for ASP.NET
Script Support
Registration, Localization, Debug/Release Mode
Web Services
Proxy Generation, Page Methods, XML & JSON Serialization
Application Services
Authentication, Profile, & Roles
Server Controls
ScriptManager, UpdatePanel, UpdateProgress, Timer

Figure: Microsoft Ajax client and server architecture

The illustration shows the functionality of the client-based Microsoft Ajax Library, which includes support for creating client components, browser compatibility, and networking and core services. The illustration also shows the functionality of server-based Microsoft Ajax features, which include script support, Web services, application services, and server controls. The following sections describe the illustration in more detail.

Microsoft Ajax Client Architecture

The client architecture includes libraries for component support, browser compatibility, networking, and core services.

Components

Client components enable rich behaviors in the browser without postbacks. Components fall into three categories:

- Components, which are non-visual objects that encapsulate code.
- Behaviors, which extend the behavior of existing DOM elements.
- Controls, which represent a new DOM element that has custom behavior.

The type of component that you use depends on the type of client behavior you want. For example, a watermark for an existing text box can be created by using a behavior that is attached to the text box

Browser Compatibility

The browser compatibility layer provides Microsoft Ajax scripting compatibility for the most frequently used browsers (including Microsoft Internet Explorer, Mozilla Firefox, and Apple Safari). This enables you to write the same script regardless of which supported browser you are targeting.

Networking

The networking layer handles communication between script in the browser and Web-based services and applications. It also manages asynchronous remote method calls. In many scenarios, such as partial-page updates that use the UpdatePanel control, the networking layer is used automatically and does not require that you write any code.
The networking layer also provides support for accessing server-based forms authentication, role information, and profile information in client script. This support is also available to Web applications that are not created by using ASP.NET, as long as the application has access to the Microsoft Ajax Library.

Core Services

The Ajax client-script libraries in ASP.NET consist of JavaScript (.js) files that provide features for object-oriented development. The object-oriented features included in the Microsoft Ajax client-script libraries enable a high level of consistency and modularity in client scripting. The following core services are part of the client architecture:

- Object-oriented extensions to JavaScript, such as classes, namespaces, event handling, inheritance, data types, and object serialization.
- A base class library, which includes components such as string builders and extended error handling.
- Support for JavaScript libraries that are either embedded in an assembly or are provided as standalone JavaScript (.js) files. Embedding JavaScript libraries in an assembly can make it easier to deploy applications and can help solve versioning issues. Debugging and Error Handling

The core services include the Sys.Debug class, which provides methods for displaying objects in readable form at the end of a Web page. The class also shows trace messages, enables you to use assertions, and lets you break into the debugger. An extended Error object API provides helpful exception details with support for release and debug modes.

Globalization

The Ajax server and client architecture in ASP.NET provides a model for localizing and globalizing client script. This enables you to design applications that use a single code base to provide UI for many locales (languages and cultures). For example, the Ajax architecture enables JavaScript code to format Date or Number objects automatically according to culture settings of the user's browser, without requiring a postback to the server.

Ajax Server Architecture

The server pieces that support Ajax development consist of ASP.NET Web server controls and components that manage the UI and flow of an application. The server pieces also manage serialization, validation, and control extensibility. There are also ASP.NET Web services that enable you to access ASP.NET application services for forms authentication, roles, and user profiles.

Script Support

Ajax features in ASP.NET are commonly implemented by using client script libraries that perform processing strictly on the client. You can also implement Ajax features by using server controls that support scripts sent from the server to the client.
You can also create custom client script for your ASP.NET applications. In that case, you can also use Ajax features to manage your custom script as static .js files (on disk) or as .js files embedded as resources in an assembly.
Ajax features include a model for release and debug modes. Release mode provides error checking and exception handling that is optimized for performance, with minimized script size. Debug mode provides more robust debugging features, such as type and argument checking. ASP.NET runs the debug versions when the application is in debug mode. This enables you to throw exceptions in debug scripts while

minimizing the size of release code.
Script support for Ajax in ASP.NET is used to provide two important features:

- The Microsoft Ajax Library, which is a type system and a set of JavaScript extensions that provide namespaces, inheritance, interfaces, enumerations, reflection, and additional features.
- Partial-page rendering, which updates regions of the page by using an asynchronous postback.
  Localization

The Microsoft Ajax architecture builds on the foundation of the ASP.NET 2.0 localization model. It provides additional support for localized .js files that are embedded in an assembly or that are provided on disk. ASP.NET can serve localized client scripts and resources automatically for specific languages and regions.

Web Services

With Ajax functionality in an ASP.NET Web page, you can use client script to call both ASP.NET Web services (.asmx) and Windows Communication Foundation (WCF) services (.svc). The required script references are automatically added to the page, and they in turn automatically generate the Web service proxy classes that you use from client script to call the Web service.
You can also access ASP.NET Web services without using Microsoft Ajax server controls (for example, if you are using a different Web development environment). To do so, in the page, you can manually include references to the Microsoft Ajax Library, to script files, and to the Web service itself. At run time, ASP.NET generates the proxy classes that you can use to call the services.

Application Services

Application services in ASP.NET are built-in Web services that are based on ASP.NET forms authentication, roles, and user profiles. These services can be called by client script in an Ajax-enabled Web page, by a Windows client application, or by a WCF-compatible client.

Server Controls

Ajax server controls consist of server and client code that integrate to produce rich client behavior. When you add an Ajax-enabled control to an ASP.NET Web page, the page automatically sends supporting client script to the browser for Ajax functionality. You can provide additional client code to customize the functionality of a control, but this is not required.
The following list describes the most frequently used Ajax server controls.

Control	Description
ScriptManager	Manages script resources for client components, partial-page rendering, localization, globalization, and custom user scripts. The ScriptManager control is required in order to use the UpdatePanel, UpdateProgress, and Timer controls. However, the ScriptManager control is not required when creating a client-only solution.
UpdatePanel	Enables you to refresh selected parts of the page, instead of refreshing the whole page by using a synchronous postback.
UpdateProgress	Provides status information about partial-page updates in UpdatePanel controls.
Timer	Performs postbacks at defined intervals. You can use the Timer control to post the whole page, or use it together with the UpdatePanel control to perform partial-page updates at a defined interval.

You can also create custom ASP.NET server controls that include Ajax client behaviors. Custom controls that enhance the capabilities of other ASP.NET Web controls are referred to as extender controls.

Ajax Control Toolkit

The Ajax Control Toolkit contains controls that you can use to build highly responsive and interactive

Ajax-enabled Web applications. These controls do not require knowledge of JavaScript or Ajax. They are designed using concepts that are familiar to ASP.NET Web Forms application developers. Using the Ajax Control Toolkit, you can build Ajax-enabled ASP.NET Web Forms applications and ASP.NET MVC Web applications by dragging the controls from the Visual Studio Toolbox onto a page. The Ajax Control Toolkit is an open-source project that is part of the CodePlex Foundation

5. **Develop a web application for Airline Reservation System using AJAX.**

Airline Reservation System using AJAX

This example application simulates an online reservation system that allows users to search for the best flights available between two cities. The system processes requests and provides the flight numbers, seating information and price for each leg of a three-leg flight, computing the total price at the end.
This Flight Reservation example application consists of two parts:

1. **Client-side Frontend Application**
   A client-side application which accepts end user requests from an AJAX-based graphical user interface (GUI). This application enables users to to select between one departure city and two possible destinations.
2. **Server-side Backend Application**
   A server-side application that processes end-user requests.

6. **With a simple example illustrate the steps to create a java web service. (NOV/DEC 2012)**

Writing a java web service

Currency conversion Service

- Writing a server for a service using JWSDP 1.3 tools
- Application: currency converter
  - Three operations:
    - fromDollars
    - fromEuros
    - fromYen
  - Input: value in specified currency
  - Output: object containing input value and equivalent values in other two currencies

Writing server software

1. Write service endpoint interface
   - May need to write additional classes representing data structures
2. Write class implementing the interface
3. Compile classes
4. Create configuration files and run JWSDP tools to create web service
5. Deploy web service to Tomcat

service endpoint interface

- The Web service endpoint interface is used to define the 'Web services methods'.
- A Web service endpoint interface must conform to the rules of a JAX-RPC service definition interface.
- a service endpoint interface (SEI) that defines the interface of the web service.
- Configuration files are XML files that can be changed as needed. Developers can use configuration files to change settings without recompiling applications. Administrators can use configuration files to set policies that affect how applications run on their computers.
- config.xml : Defines the URL for WSDL file location. Each Web services has a corresponding WSDL (Web service Definition Language) document.

JWSDP: Server

- **Rules for Service endpoint interface**
  - Must extend java.rmi.Remote
    - declares a set of methods that may be invoked from a remote Java Virtual Machine(JVM)
  - Every method must throw java.rmi.RemoteException
  - Parameter/return value data types are restricted
  - No public static final declarations (global constants) It must not have constant declarations
- **Allowable parameter/return value data types**
  - Java primitives (int, boolean, *etc*.)
  - Primitive wrapper classes (Integer, *etc*.)
  - String, Date, Calendar, BigDecimal, BigInteger
  - java.xml.namespace.QName, java.net.URI
  - Struct: class consisting entirely of public instance variables
  - Array of any of the above
- **Struct for currency converter app (data type for return values)**

```
package myCurCon;

public class ExchangeValues {
    public double dollars;
    public double euros;
    public double yen;
}
```

- **Service endpoint interface**

```
package myCurCon;

public interface CurCon extends java.rmi.Remote {
    public ExchangeValues fromDollars(double dollars)
        throws java.rmi.RemoteException;
    public ExchangeValues fromEuros(double euros)
        throws java.rmi.RemoteException;
    public ExchangeValues fromYen(double yen)
        throws java.rmi.RemoteException;
}
```

- Three files ExchangeValues.java, CurCon.java and CurConImpl.java written for the web service
- Class CurConImpl contains methods implements sevice endpoint interface, for *example:*

```
public ExchangeValues fromDollars(double dollars)
    throws java.rmi.RemoteException
{
    ExchangeValues ev = new ExchangeValues();
    ev.dollars = dollars;
    ev.euros = dollars * dollar2euro;
    ev.yen = dollars * dollar2yen;
    return ev;
}
```

Packaging server software

- Configuration file input to wscompile to create server

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <service
    name="HistoricCurrencyConverter"
    targetNamespace="http://tempuri.org/wsdl"
    typeNamespace="http://tempuri.org/types"
    packageName="myCurCon">
    <interface name="myCurCon.CurCon" />
  </service>
</configuration>
```

- Configuration file for web service

```
<?xml version="1.0" encoding="UTF-8"?>
<webServices
  xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/dd"
  version="1.0"
  targetNamespaceBase="http://tempuri.org/wsdl"
  typeNamespaceBase="http://tempuri.org/types"
```

- Configuration file for web service

```
  urlPatternBase="/converter">

  <endpoint
    name="CurrConverter"
    displayName="Currency Converter"
    description=
      "Converts between dollars, euros, and yen."
    interface="myCurCon.CurCon"
    model="/WEB-INF/model.xml.gz"
    implementation="myCurCon.CurConImpl"/>

  <endpointMapping
    endpointName="CurrConverter"
    urlPattern="/currency" />

</webServices>
```

Context path

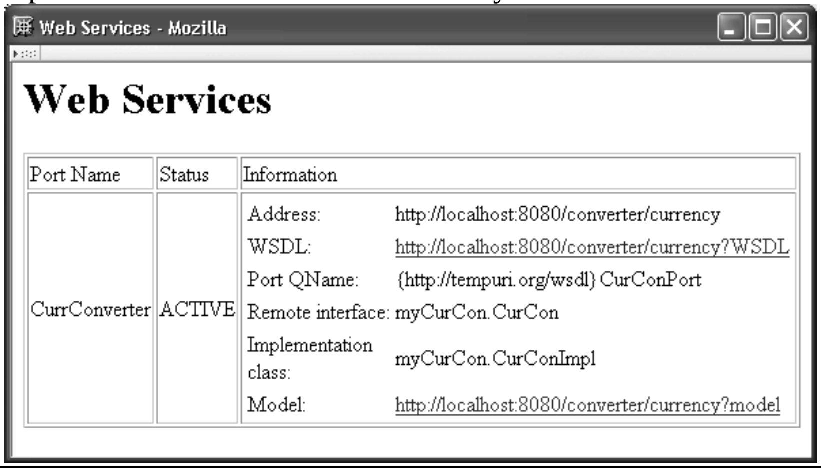
Like servlet in web.xml

Like servlet-mapping in web.xml

- Also need a minimal web.xml

```
<web-app
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">

  <display-name>Historic Currency Converter</display-name>
  <description>
    This web service converts between three currencies using their
    exchange rates as of a fixed date.
  </description>
</web-app>
```

- Run jar and wsdeploy to create a Web Archive (WAR) file converter.war
    - Name must match urlPatternBase value
- jaxrpc-ri.xml: Defines the various end points for referencing a Web service.
- *wscompile*: The wscompile tool generates stubs, and WSDL files used in JAX-RPC clients and services. The tool reads as input a configuration file and either a WSDL file or an RMI interface that defines the service.
- *wsdeploy*: Reads a WAR file (something like Jar file) and the jaxrpc-ri.xml file and then generates another WAR file that is ready for deployment
- Write service endpoint interface
    - May need to write additional classes representing data structures
- Write class implementing the interface
- Compile classes
- Create configuration files and run JWSDP tools to create web service
- Deploy web service to Tomcat
- Just copy converter.war to Tomcat webapps directory
    - May need to use Manager app to deploy
    - Enter converter.war in "WAR or Directory URL" text box
- Testing success:
    - Visit http://localhost:8080/converter/currency



Web Services - Mozilla

Web Services

Port Name	Status	Information
CurrConverter	ACTIVE	Address: http://localhost:8080/converter/currency WSDL: http://localhost:8080/converter/currency?WSDL Port QName: {http://tempuri.org/wsdl} CurConPort Remote interface: myCurCon.CurCon Implementation class: myCurCon.CurConImpl Model: http://localhost:8080/converter/currency?model

7. Show the relationship between SOAP, UDDI, WSIL and WSDL

The following standards play key roles in Web services:

- Universal Description, Discovery and Integration (UDDI),
- Web Services Description Language (WSDL),
- Web Services Inspection Language (WSIL), SOAP, and
- Web Services Interoperability (WS-I).

The relationship between these standards is described in the following figure.

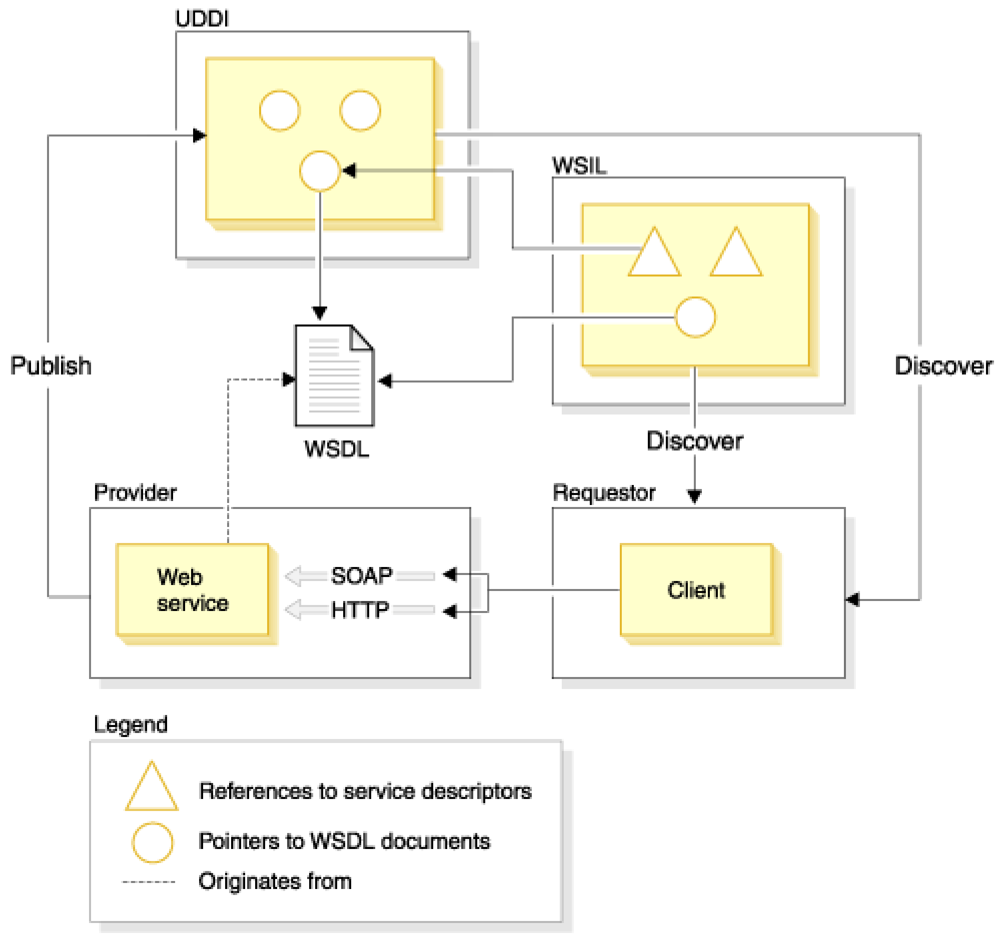


Figure: Relationships between SOAP, UDDI, WSIL and WSDL.

The UDDI specification defines open, platform-independent standards that enable businesses to share information in a global business registry, discover services on the registry, and define how they interact over the Internet.

WSIL (Web Services Inspection Language)

- XML-based open specification that defines a distributed service discovery method that supplies references to service descriptions at the service provider's point-of-offering, by specifying how to inspect a Web site for available Web services.
- A WSIL document defines the locations on a Web site where you can look for Web service descriptions.
- Since WSIL focuses on distributed service discovery, the WSIL specification complements UDDI by facilitating the discovery of services that are available on Web sites that may not be listed yet in a UDDI registry.

WSDL

- XML-based open specification that describes the interfaces to and instances of Web services on the network.

- It is extensible, so endpoints can be described regardless of the message formats or network protocols that are used to communicate.
- Businesses can make the WSDL documents for their Web services available through UDDI, WSIL, or by broadcasting the URLs to their WSDL via email or Web sites.

SOAP

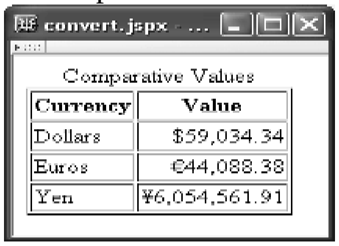
- XML-based standard for messaging over HTTP and other Internet protocols.
- It is a lightweight protocol for the exchange of information in a decentralized, distributed environment.
- It is based on XML and consists of three parts:
  - An envelope that defines a framework for describing what is in a message and how to process it.
  - A set of encoding rules for expressing instances of application-defined data types.
  - A convention for representing remote procedure calls and responses.
- SOAP enables the binding and usage of discovered Web services by defining a message path for routing messages.
- SOAP may be used to query UDDI for Web services.

A service provider hosts a Web service and makes it accessible using protocols such as SOAP/HTTP or SOAP/JMS. The Web service is described by a WSDL document that is stored on the provider's server or in a special repository. The WSDL document may be referenced by the UDDI business registry and WSIL documents. These contain pointers to the Web service's WSDL files.

8. **Explain the creation of a java web service Client in detail with examples. (MAY/JUNE 2012)**

Writing a Java Web service Client

- Goal: write a JSP-based client
  - Input: currency and value
  - Output: table of equivalent values
- Use wscompile tool to develop client



Comparative Values

Currency	Value
Dollars	$59,034.34
Euros	€44,088.38
Yen	¥6,054,561.91

- Input xml document for wscompile tool
- Configuration file input to wscompile to create client
- Child element wsdl specifies the URL of a WSDL document

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
    xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <wsdl
    location=
    "http://localhost:8080/converter/currency?WSDL"
    packageName="myCurConClient" />
</configuration>
```

- Wscompile generate proxy object
- Proxy object is a java object called on by a client software in order to access the web service

JWSDP: Client

- **Directory structure (wscompile generates content of classes and src)**
- **Run wscompile**

```
webapps
  [[ other web application document base directories ]]
  ConverterClient
    WEB-INF
      classes
        myCurConClient
      src
        myCurConClient
```

- **Run wscompile**

Wscompile –gen –keep –d classes –s src config.xml

- Wscompile tool creates a class implementing the interface
- Interface is shared between webservice server and clients via the wsdl document.
- On server side the class implementing the interface is written
- On client side the interface is automatically generated by wscompile tool

Structs will be represented as JavaBeans classes, regardless of how they are defined on the server

```
public class ExchangeValues {
    protected double dollars;
    protected double euros;
    protected double yen;
    ...
    public double getDollars() {
        return dollars;
    }

    public void setDollars(double dollars) {
        this.dollars = dollars;
    }
    ...
```

- Bean obtaining and calling proxy object:
- JSP document convert.jspx calls on javaBeans class to perform currency conversion and displays result in HTML table
- Document is placed in ConverterClient directory

```
public ExchangeValues getExValues() {
    ExchangeValues ev = null;
    CurCon curCon =
        (new HistoricCurrencyConverter_Impl()).getCurConPort();
    try {
        if (currency.equals("euros")) {
            ev = curCon.fromEuros(value);
        }
        else if (currency.equals("yen")) {
            ev = curCon.fromYen(value);
        }
        else {
            ev = curCon.fromDollars(value);
        }
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    return ev;
}
```

- **JSP document using the bean:**

```
<c:set var="exvals" value="${client.exValues}" />
...
    <tr>
      <td>Euros</td>
      <td style="text-align:right">
        <fmt:formatNumber
          type="currency" currencySymbol="&#8364;">
          ${exvals.euros}
        </fmt:formatNumber>
      </td>
    </tr>
```

9. Describe Messaging protocol in web services with its functionalities.

SOAP is one of the more popular standards, and is one of the most significant standards in communicating web services over the network. XML provides a means for communicating over the Web using an XML document that both requests and responds to information between two disparate systems. SOAP allows the sender and the receiver of XML documents to support a common data transfer protocol for effective networked communication.

SOAP Building Blocks

A SOAP message is an ordinary XML document containing the following elements:

- A required Envelope element that identifies the XML document as a SOAP message
- An optional Header element that contains header information
- A required Body element that contains call and response information
- An optional Fault element that provides information about errors that occurred while processing the message

All the elements above are declared in the default namespace for the SOAP envelope:
http://www.w3.org/2001/12/soap-envelope
and the default namespace for SOAP encoding and data types is:
http://www.w3.org/2001/12/soap-encoding

Syntax Rules

Here are some important syntax rules:

- A SOAP message MUST be encoded using XML
- A SOAP message MUST use the SOAP Envelope namespace
- A SOAP message MUST use the SOAP Encoding namespace
- A SOAP message must NOT contain a DTD reference
- A SOAP message must NOT contain XML Processing Instructions

Skeleton SOAP Message

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Header>
...
...
</soap:Header>
<soap:Body>
...
...
<soap:Fault>
...
...
</soap:Fault>
</soap:Body>
</soap:Envelope>
```

SOAP ElementS:

SOAP Envelope Element

The mandatory SOAP Envelope element is the root element of a SOAP message.
The required SOAP Envelope element is the root element of a SOAP message. It defines the XML document as a SOAP message.
Note the use of the xmlns:soap namespace. It should always have the value of:
http://www.w3.org/2001/12/soap-envelope
and it defines the Envelope as a SOAP Envelope:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
...
Message information goes here
...
</soap:Envelope>
```

The xmlns:soap Namespace

A SOAP message must always have an Envelope element associated with the "http://www.w3.org/2001/12/soap-envelope" namespace.
If a different namespace is used, the application must generate an error and discard the message.

The encodingStyle Attribute

The SOAP encodingStyle attribute is used to define the data types used in the document. This attribute may appear on any SOAP element, and it will apply to that element's contents and all child elements. A

SOAP message has no default encoding.
Syntax
soap:encodingStyle="URI"

SOAP Header Element
The optional SOAP Header element contains header information.

The optional SOAP Header element contains application specific information (like authentication, payment, etc) about the SOAP message. If the Header element is present, it must be the first child element of the Envelope element.
Note: All immediate child elements of the Header element must be namespace-qualified.

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Header>
<m:Trans
xmlns:m="http://www.w3schools.com/transaction/"
soap:mustUnderstand="1">234</m:Trans>
</soap:Header>
...
...
</soap:Envelope>
```

The example above contains a header with a "Trans" element, a "mustUnderstand" attribute value of "1", and a value of 234.
SOAP defines three attributes in the default namespace ("http://www.w3.org/2001/12/soap-envelope"). These attributes are: actor, mustUnderstand, and encodingStyle. The attributes defined in the SOAP Header defines how a recipient should process the SOAP message.
The actor Attribute
A SOAP message may travel from a sender to a receiver by passing different endpoints along the message path. Not all parts of the SOAP message may be intended for the ultimate endpoint of the SOAP message but, instead, may be intended for one or more of the endpoints on the message path.
The SOAP actor attribute may be used to address the Header element to a particular endpoint.

Syntax soap:actor="URI"

Example
<?xml version="1.0"?>
The mustUnderstand Attribute
The SOAP mustUnderstand attribute can be used to indicate whether a header entry is mandatory or optional for the recipient to process.
If you add "mustUnderstand="1" to a child element of the Header element it indicates that the receiver processing the Header must recognize the element. If the receiver does not recognize the element it must fail when processing the Header.

Syntax soap:mustUnderstand="0|1"

SOAP Body Element

The mandatory SOAP Body element contains the actual SOAP message.
The required SOAP Body element contains the actual SOAP message intended for the ultimate endpoint of the message.
Immediate child elements of the SOAP Body element may be namespace-qualified. SOAP defines one element inside the Body element in the default namespace ("http://www.w3.org/2001/12/soap-envelope"). This is the SOAP Fault element, which is used to indicate error messages.

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body>
<m:GetPrice xmlns:m="http://www.w3schools.com/prices">
<m:Item>Apples</m:Item>
</m:GetPrice>
</soap:Body>
</soap:Envelope>
```

The example above requests the price of apples. Note that the m:GetPrice and the Item elements above are application-specific elements. They are not a part of the SOAP standard.

A SOAP response could look something like this: <?xml version="1.0"?>

```
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body>
<m:GetPriceResponse xmlns:m="http://www.w3schools.com/prices">
<m:Price>1.90</m:Price>
</m:GetPriceResponse>
</soap:Body>
</soap:Envelope>
```

SOAP Fault Element

The optional SOAP Fault element is used to hold error and status information for a SOAP message.
An error message from a SOAP message is carried inside a Fault element.
If a Fault element is present, it must appear as a child element of the Body element. A Fault element can only appear once in a SOAP message.
The SOAP Fault element has the following sub elements:

Sub Element	**Description**
<faultcode>	A code for identifying the fault
<faultstring>	A human readable explanation of the fault
<faultactor>	Information about who caused the fault to happen
<detail>	Holds application specific error information related to the Body element

SOAP Fault Codes

The faultcode values defined below must be used in the faultcode element when describing faults: **Error**	**Description**
VersionMismatch	Found an invalid namespace for the SOAP Envelope element
MustUnderstand	An immediate child element of the Header element, with the mustUnderstand attribute set to "1", was not understood
Client	The message was incorrectly formed or contained incorrect information
Server	There was a problem with the server so the message could not proceed

10. Explain the anatomy of UDDI.

UDDI is

- A federated and open database
- A standard SOAP interface for querying and submitting information
- A repository that includes custom metadata for querying and finding Web services

UDDI enables companies to register their electronic services—everything from an e-mail address for technical support to XML-based Web services for purchasing. They can do this from a Web page or by programmatically using the UDDI interface.

There are two large pieces to UDDI, and one smaller piece. The larger pieces are:

- The data format
- The API for querying and submitting

More than two dozen data structures exist in the UDDI specification; therefore, covering them all would turn this book into a reference. What we should look at are the major data structures, and there are five very important ones, each with its own sub-elements:

- businessEntity— Contains information about a specific business, such as Microsoft.
- businessService— Contains information about a specific electronic service offered by a specific company.
- tModel— Categorizes a type. A service may contain multiple types.
- bindingTemplate— Collects tModels and other specific information about a service.
- categoryBag— Collects name–value pairs for a general categorization. businessEntity.

The following sections discuss businessEntity, businessService, and tModel in greater detail.

businessEntity

The businessEntity type holds information about specific businesses. Of course, there is no requirement that these documents precisely correspond to a single business entity. They could correspond to corporate entities, such as subsidiaries or corporate divisions.

This structure contains the following main pieces of information:

- businessServices— The service that this entity hosts
- categoryBag— A generic bag of names and values for properties that this entity has
- identifierBag— Similar to the category bag, a set of name–value pairs for generic information about the business entity
- contacts— A list of people to contact
- discoveryURLs— The location of documents that contain more information

businessService

The businessService document holds information about a specific service for a particular service. It contains the following data:

- serviceKey— A GUID that uniquely identifies the service
- name— A friendly, human-readable name for the service
- description— A human-readable description of the service and what it offers
- bindingTemplates— A set of properties that define the taxonomy of the service
- categoryBag— A generic name–value pair that helps to define the categories to which the service belongs

an example of a businessService document:

```
<businessService
    businessKey="some GUID here" serviceKey="some GUID here">
    <name>Keith's Service</name>
    <description>This service doesn't do anything</description>
    <bindingTemplate>
        ...
    </bindingTemplate>
</businessService>
```

tModel

The tModel is the hardest to understand and most misunderstood data structure in the UDDI Schema.
it's a generic keyed reference to something.
Arguably, this incredible flexibility makes tModel harder to use. But it enables us to describe all kinds of things, and then link them to all kinds of other things. (Notice that I have to use the vague word thing because of this abstraction and flexibility.)
With UDDI, tModels can have many different purposes, but one stands out in my mind: to reference Web service features in a transparent and generic fashion. I can create separate tModels for services that implement transactions, that implement reliable messaging, and that are described by WSDLs. I can then attach all of the tModels that apply to any of my individual Web services. Even better, I can search for services based on those tModels!
The tModel structure contains the following information:

- tModelKey (technical model)
- name— The friendly, human-readable name of the service
- authorizedName
- operator
- description(s)— A description of the tModel
- Idbag— A set of identifying name–value pairs
- category bag— A series of categories to which this tModel belongs, expressed as a set of name–value pairs

The following Listing shows a sample of what a tModel can look like.

Listing: A tModel Structure in the UDDI Schema

```
<tModel
    xmlns="urn:uddi-org:api"
    tModelKey="UUID:1111111-1111-1111-1111-1111111">
      <name>KeithBa.Com:PurchaseOrders</name>
      <description xml:lang="en">
            Purchase orders services
      </description>
      <overviewDoc>
          <overviewURL>http://keithba.com/po.wsdl</overviewURL>
      </overviewDoc>
      <categoryBag>
          <keyedReference
            tModelKey="UUID:C1ACF26D-9672-4404-9D70-39B756E62AB4"
            keyName="types"
            keyValue="wsdlSpec"/>
      </categoryBag>
  </tModel>
```

Programmer's API

In addition to defining a series of data structures, UDDI defines how to interact with those data structures—in other words, how to use them with SOAP. There are two major sets of APIs within UDDI:

- Inquiry Operations— For searching for information
- Publisher Operations— For saving, editing, and deleting information

Inquiry

The piece of UDDI used the most is the set of inquiry operations, often called the inquiry API. Inquiry operations with UDDI take two basic forms: browsing operations and retrieval operations.

Browsing operations are used to find something. You use them as broad-based queries to figure out what you want. All of these operations start with the pattern "find_XXX", where XXX is something specific, such as find_business or find_service.

There are only four operations in this category of UDDI operations:

- find_binding
- find_business
- find_service
- find_tModel

Once you know what you want, you can use the drill-down information retrieval operations to get all of the details you need about a specific business or service. These operations require specific UUIDs (Universally Unique Identifiers) that you probably will get via the browsing "find" operations. They all follow the pattern of "get_XXX," where XXX is the specific information you need. Using Microsoft's UDDI SDK, you can easily use these find operations:

```
Inquire.Url = "http://uddi.rte.microsoft.com/inquire";
FindBusiness findBusiness = new FindBusiness();
findBusiness.Names.Add("KeithBa");
BusinessList list = fb.Send();
```

There are five retrieval operations:

- get_bindingDetail
- get_businessDetail
- get_businessDetailExt
- get_serviceDetail
- get_tModelDetail

And again, the UDDI SDK from Microsoft makes it easy to call these:

```
Inquire.Url = "http://uddi.rte.microsoft.com/inquire";

FindBusiness findBusiness = new FindBusiness();
findBusiness.Names.Add("KeithBa, inc.");
```

```
BusinessList list = findBusiness.Send();

if (list.BusinessInfos.Count > 0)
{
     GetBusinessDetail gb = new GetBusinessDetail();
     gb.BusinessKeys.Add(bizList.BusinessInfos[0].BusinessKey);
     BusinessDetail bizDetail = gb.Send();
     if (bizDetail.BusinessEntities.Count > 0)
    {
        //do something interesting
    }
}
```

WSDL and UDDI

UDDI offers a way to store abstract WSDL documents (WSDLs that don't point to a specific service, but instead can be implemented by any number of services), as well. Actually, it provides for a specific tModel structure to which each businessService can point.

The basic idea is that it is possible to create WSDL documents that are abstract. Technically, almost any WSDL that is missing a service element and the child port pointing to a specific address is abstract. In practice, the UDDI binding is for abstract WSDLs that are described down through the binding section. Of course, nothing prevents you from using the true point of abstraction in a WSDL: the portType section.

Once you've defined this abstract WSDL (e.g., as part of a standards organization), you can then create a tModel in UDDI that references this WSDL. The important part to remember is that there must be a keyed-Reference to the tModelKey that represents abstract WSDLs.

Also, the overiewURL element should point to the WSDL file.

Listing: A tModel That References a WSDL Document

```
<tModel tModelKey="UUID:1111111-1111-1111-111111">
    <Name>Standard WSDL for AutoParts</Name>
    <OverviewDoc>
      <overviewURL>http://autoparts.org/autoparts.wsdl</overviewURL>
    </OverviewDoc>
    <categoryBag>
        <keyedReference
            tModelKey="uudi:C1ACF26D-9672-4404-9D70-39B756E62AB4"
            keyName="uddi-org:types"
            keyValue="wsdlSpec"
        />
    </categoryBag>
</tModel>
```

Now, we can reference this tModel from other tModels or (more likely) from businessService entries that represent specific implementations of this WSDL.

11. **Explain the anatomy of WSDL.**

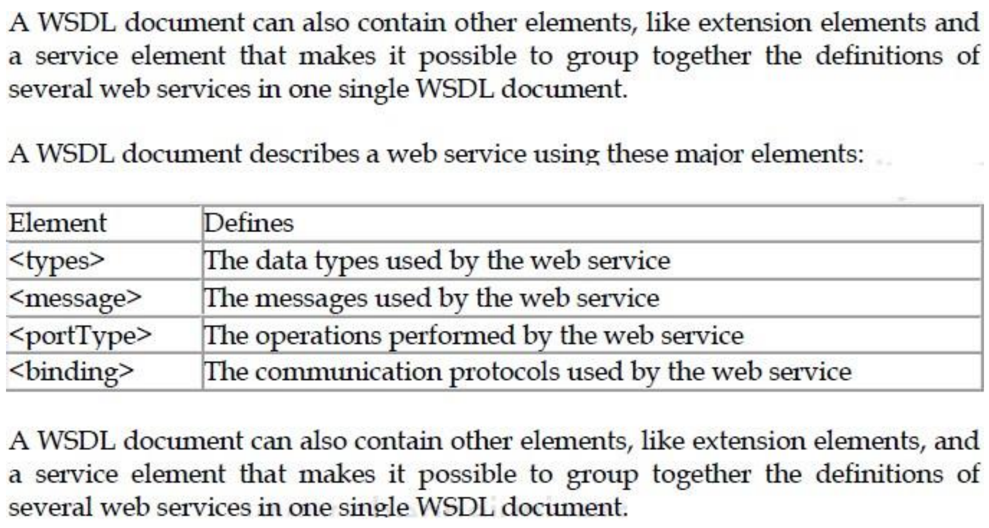
<u>**WSDL**</u>

- **Web Service Description Language.**
- **An XML language used to *describe* and *locate* web services.**
    - **Written in XML.**

- **Describe functionality of a web service**
- **Specify how to access the service (binding protocol, message format, and etc.)**

Main Structure of WSDL

```
<definition namespace = "http/… ">
        <type> xschema types </type>
        <message> … </message>
        <port> a set of operations </port>
        <binding> communication protocols </binding>
        <service> a list of binding and ports </service>
<definition>
```


A WSDL document can also contain other elements, like extension elements and a service element that makes it possible to group together the definitions of several web services in one single WSDL document.

A WSDL document describes a web service using these major elements:

Element	Defines
<types>	The data types used by the web service
<message>	The messages used by the web service
<portType>	The operations performed by the web service
<binding>	The communication protocols used by the web service

A WSDL document can also contain other elements, like extension elements, and a service element that makes it possible to group together the definitions of several web services in one single WSDL document.

Types

- <types> define types used in message declaration
- XML Schema, DTD, and etc.
- XML Schema must be supported by any vendor of WSDL conformant products.

WSDL Messages

- The **<message>** element defines the data elements of an operation.
- Each messages can consist of one or more parts. The parts can be compared to the parameters of a function call in a traditional programming language.

WSDL Ports

- The **<portType>** element is the most important WSDL element.
- It defines **a web service**, the **operations** that can be performed, and the **messages** that are involved.
- The <**port**> defines the connection point to a web service, an instance of <**portType**>.

It can be compared to a function library (or a module, or a class) in a traditional programming language. Each operation can be compared to a function in a traditional programming language

Operation Types

- The request-response type is the most common operation type, but WSDL defines four types:
  - **One-way**: The operation can receive a message but will not return a response
  - **Request-response**:The operation can receive a request and will return a response
  - **Solicit-response**:The operation can send a request and will wait for a response
  - **Notification**:The operation can send a message but will not wait for a response

Binding

- Binding defines how message are transmitted, and the location of the service.

WSDL Example

This is a simplified fraction of a WSDL document:

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>
```

In this example the **<portType>** element defines "glossaryTerms" as the name of a **port**, and "getTerm" as the name of an **operation**.
The "getTerm" operation has an **input message** called "getTermRequest" and an **output message** called "getTermResponse".
The **<message>** elements define the **parts** of each message and the associated data types.

12. Describe the major elements of SOAP. (NOV/DEC 2011, MAY/JUNE 2014) (APR/MAY 2013)

SOAP is a simple XML based protocol to let applications exchange information over HTTP.
Or more simply: SOAP is a protocol for accessing a Web Service.

SOAP Building Blocks

A SOAP message is an ordinary XML document containing the following elements:

- A required Envelope element that identifies the XML document as a SOAP message
- An optional Header element that contains header information
- A required Body element that contains call and response information
- An optional Fault element that provides information about errors that occurred while processing the message

All the elements above are declared in the default namespace for the SOAP envelope:
http://www.w3.org/2001/12/soap-envelope
and the default namespace for SOAP encoding and data types is:
http://www.w3.org/2001/12/soap-encoding

Syntax Rules

Here are some important syntax rules:

- A SOAP message MUST be encoded using XML
- A SOAP message MUST use the SOAP Envelope namespace
- A SOAP message MUST use the SOAP Encoding namespace
- A SOAP message must NOT contain a DTD reference
- A SOAP message must NOT contain XML Processing Instructions

Skeleton SOAP Message

```
<?xml version="1.0"?>
<soap:Envelope
```

```
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Header>
...
...
</soap:Header>
<soap:Body>
...
...
<soap:Fault>
...
...
</soap:Fault>
</soap:Body>
</soap:Envelope>
```

SOAP ElementS:

SOAP Envelope Element

The mandatory SOAP Envelope element is the root element of a SOAP message.

The required SOAP Envelope element is the root element of a SOAP message. It defines the XML document as a SOAP message.

Note the use of the xmlns:soap namespace. It should always have the value of:
http://www.w3.org/2001/12/soap-envelope
and it defines the Envelope as a SOAP Envelope:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
...
Message information goes here
...
</soap:Envelope>
```

The xmlns:soap Namespace

A SOAP message must always have an Envelope element associated with the "http://www.w3.org/2001/12/soap-envelope" namespace.

If a different namespace is used, the application must generate an error and discard the message.

The encodingStyle Attribute

The SOAP encodingStyle attribute is used to define the data types used in the document. This attribute may appear on any SOAP element, and it will apply to that element's contents and all child elements. A SOAP message has no default encoding.

Syntax

```
soap:encodingStyle="URI"
```

SOAP Header Element

The optional SOAP Header element contains header information.

The optional SOAP Header element contains application specific information (like authentication,

payment, etc) about the SOAP message. If the Header element is present, it must be the first child element of the Envelope element.
Note: All immediate child elements of the Header element must be namespace-qualified.

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Header>
<m:Trans
xmlns:m="http://www.w3schools.com/transaction/"
soap:mustUnderstand="1">234</m:Trans>
</soap:Header>
...
...
</soap:Envelope>
```

The example above contains a header with a "Trans" element, a "mustUnderstand" attribute value of "1", and a value of 234.
SOAP defines three attributes in the default namespace ("http://www.w3.org/2001/12/soap-envelope"). These attributes are: actor, mustUnderstand, and encodingStyle. The attributes defined in the SOAP Header defines how a recipient should process the SOAP message.

The actor Attribute

A SOAP message may travel from a sender to a receiver by passing different endpoints along the message path. Not all parts of the SOAP message may be intended for the ultimate endpoint of the SOAP message but, instead, may be intended for one or more of the endpoints on the message path.
The SOAP actor attribute may be used to address the Header element to a particular endpoint.

Syntax soap:actor="URI"

Example

```
<?xml version="1.0"?>
```

The mustUnderstand Attribute

The SOAP mustUnderstand attribute can be used to indicate whether a header entry is mandatory or optional for the recipient to process.
If you add "mustUnderstand="1" to a child element of the Header element it indicates that the receiver processing the Header must recognize the element. If the receiver does not recognize the element it must fail when processing the Header.

Syntax soap:mustUnderstand="0|1"

SOAP Body Element

The mandatory SOAP Body element contains the actual SOAP message.
The required SOAP Body element contains the actual SOAP message intended for the ultimate endpoint of the message.
Immediate child elements of the SOAP Body element may be namespace-qualified. SOAP defines one element inside the Body element in the default namespace ("http://www.w3.org/2001/12/soap-envelope"). This is the SOAP Fault element, which is used to indicate error messages.

```
<?xml version="1.0"?>
<soap:Envelope
```

```
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body>
<m:GetPrice xmlns:m="http://www.w3schools.com/prices">
<m:Item>Apples</m:Item>
</m:GetPrice>
</soap:Body>
</soap:Envelope>
```

The example above requests the price of apples. Note that the m:GetPrice and the Item elements above are application-specific elements. They are not a part of the SOAP standard.

A SOAP response could look something like this: <?xml version="1.0"?>

```
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body>
<m:GetPriceResponse xmlns:m="http://www.w3schools.com/prices">
<m:Price>1.90</m:Price>
</m:GetPriceResponse>
</soap:Body>
</soap:Envelope>
```

SOAP Fault Element

The optional SOAP Fault element is used to hold error and status information for a SOAP message.
An error message from a SOAP message is carried inside a Fault element.
If a Fault element is present, it must appear as a child element of the Body element. A Fault element can only appear once in a SOAP message.
The SOAP Fault element has the following sub elements:

Sub Element	**Description**
<faultcode>	A code for identifying the fault
<faultstring>	A human readable explanation of the fault
<faultactor>	Information about who caused the fault to happen
<detail>	Holds application specific error information related to the Body element

SOAP Fault Codes

The faultcode values defined below must be used in the faultcode element when describing faults: **Error**	**Description**
VersionMismatch	Found an invalid namespace for the SOAP Envelope element
MustUnderstand	An immediate child element of the Header element, with the mustUnderstand attribute set to "1", was

	not understood
Client	The message was incorrectly formed or contained incorrect information
Server	There was a problem with the server so the message could not proceed