

Part-II, 2nd Elective
COMPUTER APPLICATIONS

Sastri/ B.A 2nd Year
Course/Paper.2

**OBJECT ORIENTED PROGRAMMING WITH JAVA
&
WEB DESIGNING**



CENTER OF DISTANCE & ONLINE EDUCATION
(Formerly Directorate of Distance Education)

NATIONAL SANSKRIT UNIVERSITY :: TIRUPATI-517 507 (A.P)
(Erstwhile Rashtriya Sanskrit Vidyapeetha, Tirupati)

UNIT – I FUNDAMENTAL OF OBJECT ORIENTED PROGRAMMING

Structure

- 1.0 Objectives
 - 1.1 Introduction
 - 1.1.1. Object Oriented paradigm
 - 1.1.2. Basic concepts of Object Oriented programming
 - 1.1.3. Benefits of OOPs
 - 1.1.4. Applications of OOPs
 - 1.2 Overview of Java Language
 - 1.2.1. Simple Java Program
 - 1.2.2. Java program structure
 - 1.2.3. Java tokens
 - 1.2.4. Java statements
 - 1.2.5. Implementing a Java programming
 - 1.2.6. Java Virtual Machine (JVM)
 - 1.2.7. Command Line Arguments
 - 1.3 Variables and Data types
 - 1.3.1. Constants, Variables, Data types
 - 1.3.2. Declaration of Variables and giving values to variables
 - 1.3.3. Scope of variables
 - 1.3.4. Symbolic constants
 - 1.3.5. Type casting.
 - 1.4 Summary
 - 1.5 SAQ
-

1.0 Objective

The main objective of this unit is to make clear about basic of Objective Oriented Programming, its benefits, applications. OOP for Java programming language, introduction to JVM, variables and data types used in Java Language.

1.1 Introduction

OOPs stand for Object Oriented Programming, generally used as programming Language. Some of the programming language that supports OOPs say C++, Java, Eiffel, small talk and soon. Object Oriented is not only used in programming but also used as Object Oriented Software Engineering (OOSE), Object Oriented Analysis (OOA), Object Oriented Testing (OOT), Object Oriented Designing (OOD), Object Oriented Data Base Management System (OODBMS) and so on.

In this unit we concentrate how OOP concept implemented in Java programming language. Java is a high-level programming language originally developed by Sun Microsystems and released in 1995. Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX. With the advancement of Java and its widespread popularity, multiple configurations were built to suit various types of platforms. For example J2EE for Enterprise Applications, J2ME for Mobile Applications.

The main advantage of Java is guaranteed to **WIRTE ONCE, RUN ANYWHERE**.

1.1.1. Object Oriented Paradigm

Object Oriented Programming (OOP) is a programming paradigm based upon objects and classes that aims to incorporate the advantages of modularity and reusability. Objects, which are instance of classes and Class is a single unit which consists of variables, constants and member function.

The important features of object-oriented programming are

- Bottom-up approach in program design
- Programs organized around objects, grouped in classes
- Focus on data with methods to operate upon object's data
- Interaction between objects through functions
- Reusability of design through creation of new classes by adding features to existing classes

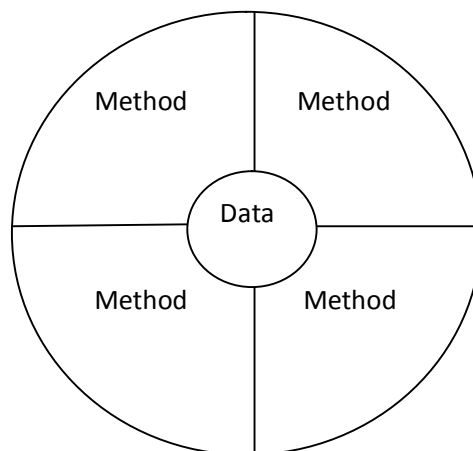


Fig1. Object = Data + Methods

1.1.2. Basic concepts of OOPs

OOPs incorporate the concepts of Objects, Classes, Data Abstraction and Encapsulation, Inheritance, Polymorphism, Dynamic Binding and Message Communication. Let us discuss each of them in detail.

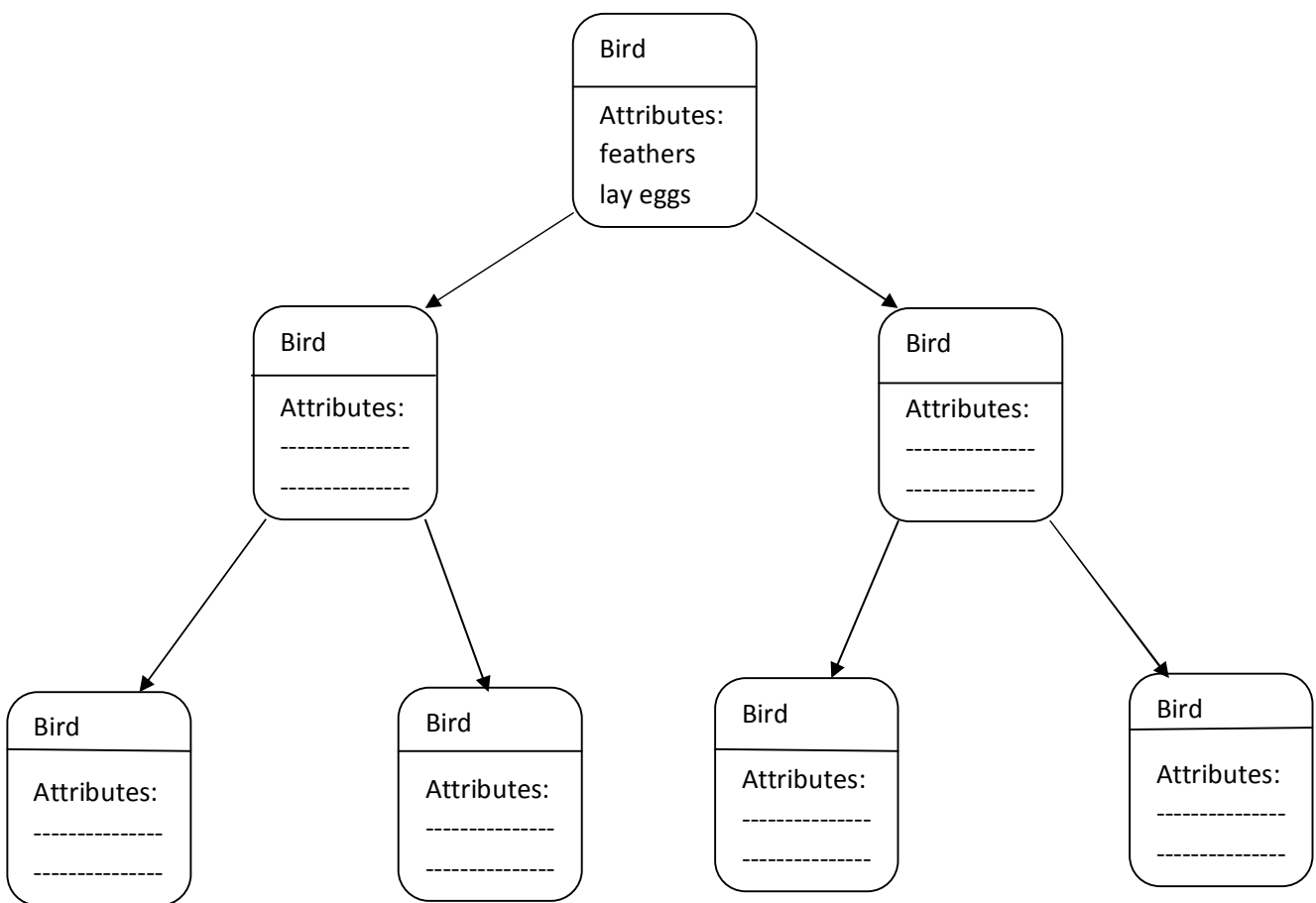
- a) **Objects:** Objects are the basic runtime entities in an Object Oriented Systems. They represent a person, place, a bank account, a table or any item that the program may handle. When a program is executed, the objects interact by sending messages to one another. For example 'customer' and 'account' are two objects in a banking program, then the customer object may send a message to the account object request for balance. Each object contains data and code to manipulate the data.

Let us consider an example of dog, then its state is – name, breed, color, and the behavior is barking, wagging the tail, running etc. software objects also have a state and a behavior. A software object's state is stored in fields and behavior is shown via methods. We can say that software development, methods operate on the internal state of an object and the object-to-object communication is done via methods. **“Object is defined as instance of class”**.

- b) **Classes:** *Collection of objects* is called class. It is a logical entity. A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space. The class is a group of similar entities. It is only a logical component and not the physical entity. For example, if you had a class called “Expensive Cars” it could have objects like Mercedes, BMW, Toyota, etc. Its properties (data) can be price or speed of these cars. While the methods may be performed with these cars are driving, reverse, braking etc.
- c) **Data Abstraction:** An abstraction is an act of representing essential features without including background details. It is a technique of creating a new data type that is suited for a specific application. For example, while driving a car, you do not have to be concerned with its internal working. Here you just need to concern about parts like steering wheel, Gears, accelerator, etc.
- d) **Encapsulation:** Encapsulation is an OOP technique of wrapping the data and code. In this OOPS concept, the variables of a class are always hidden from other classes. It can

only be accessed using the methods of their current class. For example - in school, a student cannot exist without a class.

- e) **Inheritance:** Inheritance is the process by which objects of one class acquire the properties of another class. It provides code reusability. This means that we can add additional features to an existing class without modifying it. Heritance supports the concept of hierarchical classification. It is used to achieve runtime polymorphism. For example, the bird robin is part of the class flying bird, which is again a part of the class bird. Consider the example given below



Inheritance property

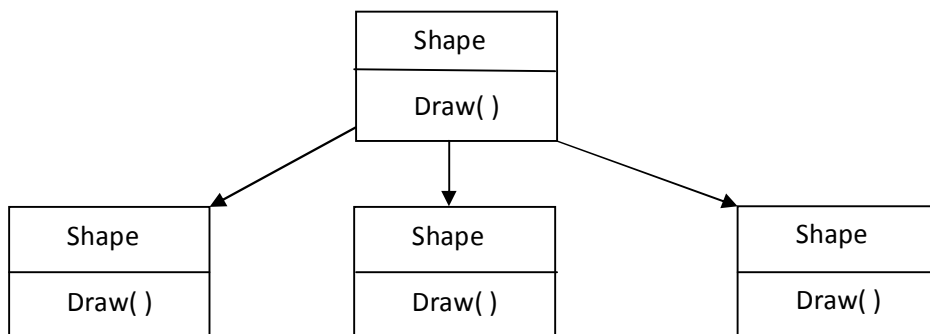
Polymorphism: Polymorphism refers to the ability of a variable, object or function to take on multiple forms. For example, in English, the verb run has a different meaning if you use it with a laptop, a foot race, and business. Here, we understand the meaning of run based on the other words used along with it. The same also applied to Polymorphism. Polymorphism plays an important role in allowing objects having different internal structure share the same

external interface. This means that a general class of operations may be accessed in same manner even though specific actions associated with each operation may differ. Polymorphism extensively used in implementing inheritance.

In Java, we use method overloading and method overriding to achieve polymorphism.

Method overloading: Method Overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different.

Method overriding: Method Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes.



Polymorphism

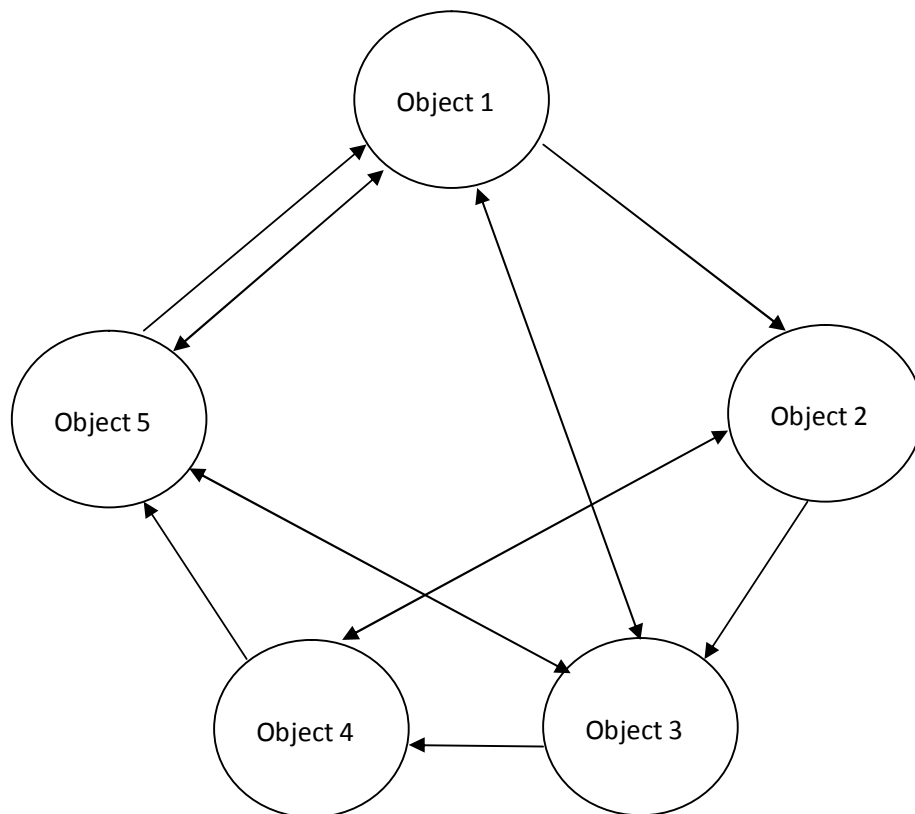
f) **Dynamic Binding:** Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at runtime. It is associated with polymorphism and inheritance. A procedure associated with a polymorphic reference depends on the dynamic type of that reference.

Let us consider the above “draw” example. By inheritance, every object will have this procedure the algorithm is, however, unique to each object and so the draw procedure will be redefined in each of that defines the object. At runtime, the code matching the object under current reference will be called.

g) **Message Communication:** An object oriented program consists of a set of objects that communicate with each other. The process of programming in an object oriented language, therefore, involves the following basic steps:

- Creating classes that define objects and their behavior.
- Creating objects from class definitions.
- Establishing communication among objects

Objects communicate with one another by sending and receiving information much the same way the people pass messages to one another as given below. The concept of message passing making easier to talk about building systems that directly model or simulate their real world counterparts.



Network of objects communicating between them

A message for an object is a request for executing of a procedure, and therefore will invoke the method (procedure) in the receiving object that generates the required result.

Employee . salary (name) ;

Here in the above example employee.salary(name) ----- employee is object, salary is message and name is information.

1.1.3 Benefits of OOPs

OOPs offers several benefits to both program designer and the user. Object orientation contribution to the solution of many problems associated with the development and quality of software products. The new technology promises greater programmer productivity, better quality of software and less maintenance cost. The principal advantages are:

1. Eliminate redundant code and extend the use of existing classes.
2. Programs can be build from the standard working modules that communicate with one another and start writing the code from scratch. This shows to saving of development time and higher productivity.
3. The data handling rules helps the programmer to make secure programs that cannot be invaded by code in other parts of the program.
4. It has multiple objects to coexist without any interference.
5. It has map objects in the problem domain to those objects in the program.
6. Partition of work is easy in a project based on objects.
7. The data-centered design approaches capture more details of a model in an implementable form.
8. Object oriented systems can be easily upgraded from small to large systems.
9. It is possible to easily manage software complexity.

1.1.4. Applications of Object Oriented Programming

Applications of OOP are beginning to gain importance in many areas. The most popular application of Object Oriented Programming, has been in the area of user interface design such as windows. There are hundreds of windowing systems developed using OOP techniques. Real business systems are often much more complex and contain many more objects with complicated attributes and methods. OOP is useful in this type of applications because it can simplify a complex problem. The promising areas for application of OOP includes:

- **Real-time systems:** Real time systems Real time systems inherit complexities that makes difficult to build them. Object-oriented techniques make it easier to handle those complexities. These techniques present ways of dealing with these complexities by

providing an integrated framework which includes schedulability analysis and behavioral specifications.

- **Simulation and modeling:** It's difficult to model complex systems due to the varying specification of variables. These are prevalent in medicine and in other areas of natural science, such as ecology, zoology, and agronomic systems. Simulating complex systems requires modelling and understanding interactions explicitly. Object-oriented Programming provides an alternative approach for simplifying these complex modelling systems.
- **Object oriented databases:** These databases try to maintain a direct correspondence between the real-world and database objects in order to let the object retain their identity and integrity. They can then be identified and operated upon.
- **Hypertext, hypermedia and expert text:** OOP also helps in laying out a framework for Hypertext. Basically, hypertext is similar to regular text as it can be stored, searched, and edited easily. The only difference is that hypertext is text with pointers to other text as well. Hypermedia, on the other hand, is a superset of hypertext. Documents having hypermedia, not only contain links to other pieces of text and information, but also to numerous other forms of media, ranging from images to sound.
- **AI and Expert systems:** These are computer applications which are developed to solve complex problems pertaining to a specific domain, which is at a level far beyond the reach of a human brain.

It has the following characteristics:

- Reliable
 - Highly responsive
 - Understandable
 - High-performance
- **Neural networks and parallel programming:** neural networks are developed in a particular time interval to disperse the load of various networks. OOP simplifies the entire process by simplifying the approximation and prediction ability of networks.
 - **Decision support and office automation systems:** These include formal as well as informal electronic systems primarily concerned with information sharing and

communication to and from people inside as well as outside the organization. Some examples are:

- Email
 - Word processing
 - Web calendars
 - Desktop publishing
- **CIM/CAD/CAD System:** OOP can also be used in manufacturing and design applications as it allows people to reduce the effort involved. For instance, it can be used while designing blueprints, flowcharts, etc. OOP makes it possible for the designers and engineers to produce these flowcharts and blueprints accurately.

1.2 Over view of Java Language

Java is a programming language and computing platform first released by Sun Microsystems in 1995. Java has emerged as the object-oriented programming language of choice. Some of the important concepts of Java include:

- A Java virtual machine (JVM), which provides the fundamental basis for platform independence.
- Automated storage management techniques, such as garbage collection.
- Language syntax that is similar to that of the C language.

The result is a language that is object-oriented and efficient for application programming.

Java is a general purpose, Object Oriented Programming Language. We can develop two types of Java programs:

- Stand-alone applications
- Web applets

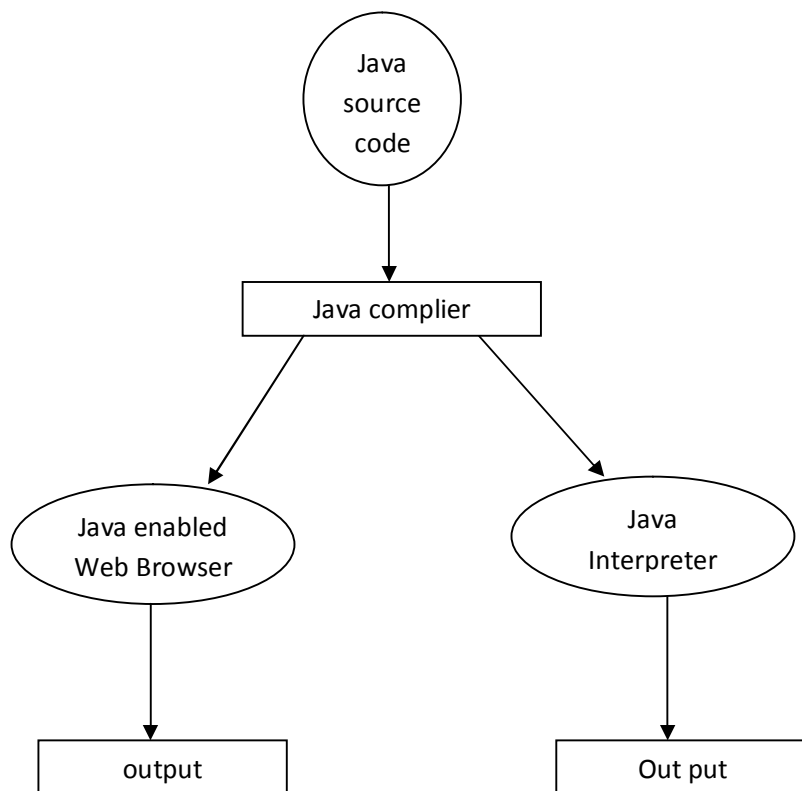
They are implemented as shown in the below figure. Stand-alone applications are programs written in Java to carry out certain tasks on a stand-alone local computer. In fact, Java can be used to develop programs for all kinds of application, which earlier, were developed using languages like C and C++. Executing a stand-alone Java program involves two steps:

- Compiling source code into bytecode using **javac** compiler.
- Executing the bytecode program using **java** interpreter.

Applets are small Java programs developed for Internet applications. An applet located on a distant computer (Server) can be downloaded via Internet and executed on a local

computer(Client) using a Java capable browser. Applets are embedded in an HTML(Hypertext Markup Language) document and run inside a Web page, creating and running applets are more complex than creating an application.

Stand alone programs can read and write files and perform certain operations that applets cannot do. An applet can only run within a Web browser.



Overview of Java Language

1.2.1. Simple Java Program

The best way to learn a new language is to write a few simple example programs and execute them. Let us start writing simple “hello”Java program

Example 1:

```
Class simple{  
  
public static void main(String args[]) {  
  
System.out.println(“Hello Java”);
```

```
}  
  
}
```

Let us discuss the program line by line in detail.

Class declaration

The first line “**class simple**“ declares a class, which is an object oriented construct. As stated earlier, java is a true object oriented language and therefore, everything must be placed inside a class. **Class** is a key word and declares that a new class definition follows. “**Simple**” is a Java identifier that specifies the name of the class to be defined.

Opening braces

Every class definition in Java begins with an opening brace “{” and ends with a matching closing brace “}” appearing in the last line in the example.

The main line

The third line

```
public static void main(String args[] )
```

defines a method named **main**. Conceptually, this is similar to the **main() function in c/c++**. Every Java application program must include the **main()** method. This is the starting point for the interpreter to begin the execution of the program. A Java application can have any number of classes but only one of them must include a **main** method to initiate the execution.

Public: the key word **public** is an access specifier that declares the **main** method as unprotected and therefore making it accessible to all other classes. This is similar to the c++ **public** modifier.

Static: next appears the keyword **static** which declares this method as one that belongs to the entire class and not a part of any objects of the class. The **main** must always be declared as **static** since the interpreter uses this method before any objects are created. More about static methods and variables.

Void: the type modifier **void** states that the **main** method does not return any value.

String[] args : Java main method accepts a single argument of type String array. This is also called as java command line arguments.

System.out.println A Java statement that prints the argument passed, into the **System.out** which is generally stdout.

- **System** – is a **final class** in java.lang package. As per javadoc, “...Among the facilities provided by the Systemclass are standard input, standard output, and error output streams; access to externally defined properties and environment variables; a means of loading files and libraries; and a utility method for quickly copying a portion of an array...”
- **out** – is a **static member field of System class and is of type PrintStream**. Its access specifiers are public final. This gets instantiated during startup and gets mapped with standard output console of the host. This stream is open by itself immediately after its instantiation and ready to accept data.
- **println** – is a **method of PrintStream class**. println prints the argument passed to the standard console and a newline. There are multiple println methods with different arguments (overloading). Every println makes a call to printmethod and adds a newline. print calls write() and the story goes on like that.

In the above Example 1. System.out.println(“Hello Java”); it prints “**Hello Java**” to the screen. The println always appends a newline character to the end of the string. This means that any subsequent output will start on a new line. Every Java statement must be ended with Semicolon.

NOTE: Once java program is written it should be run and compiled. Let us learn how to run a java program using command prompt

We use Java compiler **javac** to compile Java program and the Java interpreter **java** to run the Java program

Steps to achieve our goal:

- i. Create a folder
- ii. Create a java class and write a java program
- iii. Open command prompt
- iv. Run the created Java program using command prompt

Let us see above steps in detail

i. create a folder

c:\java

ii. Create a java class and write a java program

Using Notepad or another text editor, create a Java file **simple.java** with the following text:

```
public class simple{  
    public static void main(String[] args){  
        System.out.println("Hello Java");  
    }  
}
```

Save your file as **simple.java** in C:\java

iii. Open command prompt

Open Command Prompt (Open Run (Windows+R) and type **cmd**)

iv Run the created Java program using command prompt

```
C:\users\admin> cd\
```

```
C:\java> dir
```

This makes **C:\java** the current directory.

```
C:\SoftwareTestingMaterial> set path=%path%;C:\Program Files\Java\jdk1.8.0_101\bin
```

(NOTE : use the JDK folder for the version installed on your system). This tells the system where to find JDK programs.

```
C:\java> javac simple.java
```

This runs **javac.exe**, the compiler. You should see nothing but the next system prompt.

```
C:\java> java simple
```

This runs the Java interpreter. You should see the program output

OUTPUT: Hello Java

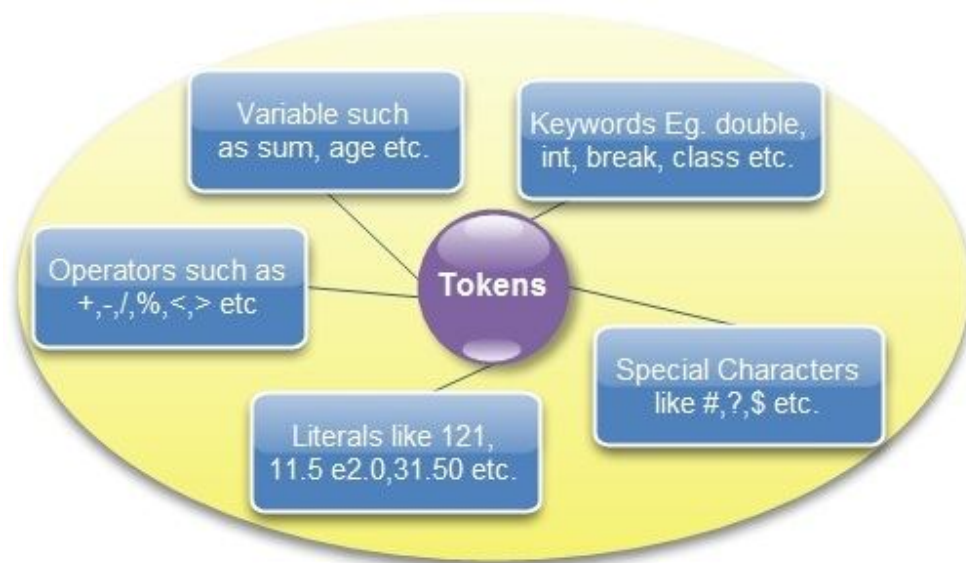
Note:

i. Java is case-sensitive! Check your Java text. Check the spelling and capitalization in the file name and the class name and the **simple** command.

ii. In “**Step iv**”, we set the **jdk** path. It is possible to make the path setting permanent but you have to be very careful because your system might crash in case of any mistake. **Proceed with extreme caution!**

1.2.3. Java tokens

Java Tokens are the smallest individual building block or smallest unit of a Java program; the Java compiler uses it for constructing expressions and statements. Java program is a collection of different types of tokens, comments, and white spaces. When we write a program, we need different important things. We require language tokens, white spaces, and formats.



There are various tokens used in Java:

- i. Reserved Keywords
- ii. Identifiers
- iii. Literals
- iv. Operators

v. Separators

White space is also considered as a token.

i. Reserved Keywords: Key words are words that have already been defined for Java compiler. They have special meaning for the compiler. Java Keywords must be in your information because you can not use them as a variable, class or a method name. You can't use keyword as identifier in your Java programs, its reserved words in Java library and used to perform an internal operation.

Abstract	Assert	boolean	break
Byte	Case	catch	char
Class	Const	continue	default
Do	Double	else	enum
Extends	Final	finally	float
For	Goto	if	implements
Import	instanceof	int	interface
Long	Native	new	package
Private	protected	public	return
Short	Static	strictfp	super
Switch	synchronized	this	throw

Throws	transient	try	void
Volatile	While	true	false
Null			

- *true*, *false* and *null* are not reserved words but cannot be used as identifiers, because it is literals of built-in types.

ii. Identifier: A Java identifier is the symbolic name that a programmer gives to various programming elements such as a variables method, class, array, etc.

iii. Literals: A literal is a constant value that can be classified as integer literals, string literals, and boolean literals.

iv. Operators: Java operators are symbols that are used to perform mathematical or logical manipulations. Java is rich with built-in operators. Operators are tokens that perform some calculations when they are applied to variables. The following are the different operators with suitable examples with Java program.

- **Arithmetic operator:** The way we calculate mathematical calculations, in the same way, Java provides arithmetic operators for mathematical operations. It provides operators for all necessary mathematical calculations. There are various arithmetic operators used in Java:

Operator	Meaning	Work
+	Addition	To add two operands.
-	Subtraction	To subtract two operands.
*	Multiplication	To multiply two operands.

/	Division	To divide two operands.
%	Modulus	To get the area of the division of two operands.

- **Unary arithmetic operator:** In Java, unary arithmetic operators are used to increasing or decreasing the value of an operand. Increment operator adds 1 to the value of a variable, whereas the decrement operator decreases a value. Increment and decrement unary operator works as follows:

Syntax:

```
val++;
```

```
val--;
```

These two operators have two forms: Postfix and Prefix. Both do increment or decrement in appropriate variables. These two operators can be placed before or after of variables. When it is placed before the variable, it is called prefix. And when it is placed after, it is called postfix.

Following example table, demonstrates the work of Increment and decrement operators with postfix and prefix:

Example	Description
val = a++;	Store the value of "a" in "val" then increments.
val = a--;	Store the value of "a" in "val" then decrements.
val = ++a;	Increments "a" then store the new value of "a" in "val".
val = --a;	Decrement "a" then store the new value of "a" in "val".

Example: Program to Show Unary operator

```
public class unaryop {  
  
    public static void main(String[] args) {  
  
        int r = 6;  
  
        System.out.println("r=: " + r++);  
  
        System.out.println("r=: " + r);  
  
        int x = 6;  
  
        System.out.println("x=: " + x--);  
  
        System.out.println("x=: " + x);  
  
        int y = 6;  
  
        System.out.println("y=: " + ++y);  
  
        int p = 6;  
  
        System.out.println("p=: " + --p);  
  
    }  
  
}
```

Output:

```
r=: 6  
  
r=: 7  
  
x=: 6  
  
x=: 5  
  
y=: 7  
  
p=: 5
```

- **Relational operator:** The Java Relational operators compare between operands and determine the relationship between them.

There are six types of relational operators in Java, these are:

Operator	Meaning
==	Is equal to
!=	Is not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Example : Program to Show Relational operators

```
public class relatiop {
    public static void main(String[] args) {
        //Variables Definition and Initialization
        int num1 = 12, num2 = 4;
        //is equal to
        System.out.println("num1 == num2 = " + (num1 == num2) );

        //is not equal to
```

```
System.out.println("num1 != num2 = " + (num1 != num2) );  
  
//Greater than  
  
System.out.println("num1 > num2 = " + (num1 > num2) );  
  
//Less than  
  
System.out.println("num1 < num2 = " + (num1 < num2) );  
  
//Greater than or equal to  
  
System.out.println("num1 >= num2 = " + (num1 >= num2) );  
  
//Less than or equal to  
  
System.out.println("num1 <= num2 = " + (num1 <= num2) );  
  
}  
  
}
```

Output:

```
num1 == num2 = false  
  
num1 != num2 = true  
  
num1 > num2 = true  
  
num1 < num2 = false  
  
num1 >= num2 = true  
  
num1 <= num2 = false
```

- **Logical operators:** The Java Logical Operators work on the Boolean operand. It's also called Boolean logical operators. It operates on two Boolean values, which return Boolean values as a result.

Operator	Meaning	Work
&&	Logical AND	If both operands are true then only "logical AND operator" evaluate true.
	Logical OR	The logical OR operator is only evaluated as true when one of its operands evaluates true. If either or both expressions evaluate to true, then the result is true.
!	Logical Not	Logical NOT is a Unary Operator, it operates on single operands. It reverses the value of operands, if the value is true, then it gives false, and if it is false, then it gives true.

Example: Program to Show Logical operator

```
public class logicalop {
    public static void main(String[] args) {
        //Variables Definition and Initialization
        boolean bool1 = true, bool2 = false;
        //Logical AND
        System.out.println("bool1 && bool2 = " + (bool1 && bool2));
        //Logical OR
        System.out.println("bool1 || bool2 = " + (bool1 | bool2) );
        //Logical Not System.out.println("!(bool1 && bool2) = " + !(bool1 && bool2)); }
    }
}
```

Output:

bool1 && bool2 = false

bool1 || bool2 = true

```
!(bool1 && bool2) = true
```

- **Bitwise operators:** The Java Bitwise Operators allow access and modification of a particular bit inside a section of the data. It can be applied to integer types and bytes, and cannot be applied to float and double.

Operator	Meaning	Work
&	Binary AND Operator	There are two types of AND operators in Java: the logical && and the binary &. Binary & operator work very much the same as logical && operators works, except it works with two bits instead of two expressions. The "Binary AND operator" returns 1 if both operands are equal to 1.
	Binary OR Operator	Like "AND operators ", Java has two different "OR" operators: the logical and the binary . Binary Operator work similar to logical operators works, except it, works with two bits instead of two expressions. The "Binary OR operator" returns 1 if one of its operands evaluates as 1. if either or both operands evaluate to 1, the result is 1.
^	Binary XOR Operator	It stands for "exclusive OR" and means "one or the other", but not both. The "Binary XOR operator" returns 1 if and only if exactly one of its operands is 1. If both operands are 1, or both are 0, then the result is 0.
~	Binary Complement Operator	
<<	Binary Left Shift Operator	
>>	Binary Right Shift Operator	
>>>	Shift right zero fill operator	

Example: Program to Show Bitwise operator

```
public class bitwiseop {  
  
    public static void main(String[] args) {  
  
        //Variables Definition and Initialization  
  
        int num1 = 30, num2 = 6, num3 =0;  
  
        //Bitwise AND  
  
        System.out.println("num1 & num2 = " + (num1 & num2));  
  
        //Bitwise OR  
  
        System.out.println("num1 | num2 = " + (num1 | num2) );  
  
        //Bitwise XOR  
  
        System.out.println("num1 ^ num2 = " + (num1 ^ num2) );  
  
        //Binary Complement Operator  
  
        System.out.println("~num1 = " + ~num1 );  
  
        //Binary Left Shift Operator  
  
        num3 = num1 << 2;  
  
        System.out.println("num1 << 1 = " + num3 );  
  
        //Binary Right Shift Operator  
  
        num3 = num1 >> 2;  
  
        System.out.println("num1 >> 1 = " + num3 );  
  
        //Shift right zero fill operator  
  
        num3 = num1 >>> 2;  
  
        System.out.println("num1 >>> 1 = " + num3 );  
    }  
}
```

```
}  
  
}
```

Output:

```
num1 & num2 = 6
```

```
num1 | num2 = 30
```

```
num1 ^ num2 = 24
```

```
~num1 = -31
```

```
num1 << 1 = 120
```

```
num1 >> 1 = 7
```

```
num1 >>> 1 = 7
```

- **Assignment operators:** The Java Assignment Operators are used when you want to assign a value to the expression. The assignment operator denoted by the single equal sign =

In a Java assignment statement, any expression can be on the right side and the left side must be a variable name. For example, this does not mean that "a" is equal to "b", instead, it means assigning the value of 'b' to 'a'. It is as follows:

Syntax:

```
variable = expression;
```

Example:

```
int a = 6;
```

```
float b = 6.8F;
```

Java also has the facility of chain assignment operators, where we can specify a single value for multiple variables.

Example: Program to Show Assignment operator

```
public class ChainAssign {  
  
    public static void main(String args[]) {  
  
        int a, b, c;  
  
        a = b = c = 100; // set a, b, and c to 100  
  
        System.out.println("a = " + a);  
  
        System.out.println("b = " + b);  
  
        System.out.println("c = " + c);  
  
    }  
}
```

Output:

```
a = 100  
  
b = 100  
  
c = 100
```

- **Conditional operator:** The Java Conditional Operator selects one of two expressions for evaluation, which is based on the value of the first operands. It is also called ternary operator because it takes three arguments. The conditional operator is used to handling simple situations in a line.

Syntax:

```
expression1 ? expression2:expression3;
```

The above syntax means that if the value given in Expression1 is true, then Expression2 will be evaluated; otherwise, expression3 will be evaluated.

Example:

```
val == 0 ? you are right:you are not right;
```

Example: Program to Show Conditional Operator Works

```
public class condiop {  
  
    public static void main(String[] args) {  
  
        String out;  
  
        int a = 6, b = 12;  
  
        out = a==b ? "Yes":"No";  
  
        System.out.println("Ans: "+out);  
  
    }  
  
}
```

Output:

```
Ans: No
```

In the above example, the condition given in expression1 is false because the value of a is not equal to the value of b.

- **Instanceof operator:** The Java instanceof Operator is used to determining whether this object belongs to this particular (class or subclass or interface) or not. This operator gives the boolean values such as true or false. If it relates to a specific class, then it returns true as output. Otherwise, it returns false as output.

Syntax:

```
object-reference instanceof type;
```

Example: Program to Show Downcasting with instanceof Operator

```
class Company {}

public class Employee extends Company {

    public void check() {

        System.out.println("Success.");

    }

    public static void view(Company c) {

        if (c instanceof Employee) {

            Employee b1 = (Employee) c;

            b1.check();

        }

    }

    public static void main(String[] args) {

        Company c = new Employee();

        Employee.view(c);

    }

}
```

Output:

```
Success.
```

- **Separators:** Separators are the lines that are used to virtual group related items together.

1.2.4. Java statement

Statements are similar to sentences in the English language. A sentence forms a complete idea which can include one or more clauses. Likewise, a *statement* in Java forms a complete

command to be executed and can include one or more expressions. Every statement in Java must be ended with semicolon(;).

In simpler terms, a Java statement is just an instruction that explains what should happen.

Types of Java Statements

Java supports three different types of statements:

- Expression statements change values of variables, call methods, and create objects. An expression with a semicolon at the end is called an expression statement. For example

```
/Increment and decrement expressions
num++;
++num;
num--;
--num;
```

```
//Assignment expressions
num = 100;
num *= 10;
```

- Declaration statement is used to declare variables. For example

```
int num;

int num1 = 100;

string str;
```

- Control-flow statements determine the order that statements are executed. Typically, Java statements parse from the top to the bottom of the program. However, with control-flow statements, that order can be interrupted to implement branching or looping so that the Java program can run particular sections of code based on certain conditions.

1.2.5. Implementing a Java program

Implementation of a Java application program involves a series of steps. They include :

- Creating the program
- Compiling the program
- Running the program

Remember that, before creating the program, the Java Development Kit (JDK) must be properly installed on our system

Creating the program

Create a program using any text editor. Assume that we have entered the following program:

```
class Test
{
    public static void main(String args[] )
    {
        System.out.println("hello world");
        System.out.println("welcome to the world of Java");
        System.out.println("let us learn Java");
    }
}
```

The program in a file must be saved with the class name Test.java ensure that the file name contains the class name properly. This file is called the source file. Note that all java source files will have the extension java.

Compiling the program

To compile the program, we must run the java compiler **javac**, with the name of the source file on the command line. If everything is OK, the **javac** compiler creates a file called **Test.class** containing the **bytecodes** of the program. Note that the compiler automatically names the bytecode file as **<classname>.class**.

Running the program

To run the program, we must run the Java interpreter **java**, with the name of the class file on the command line.

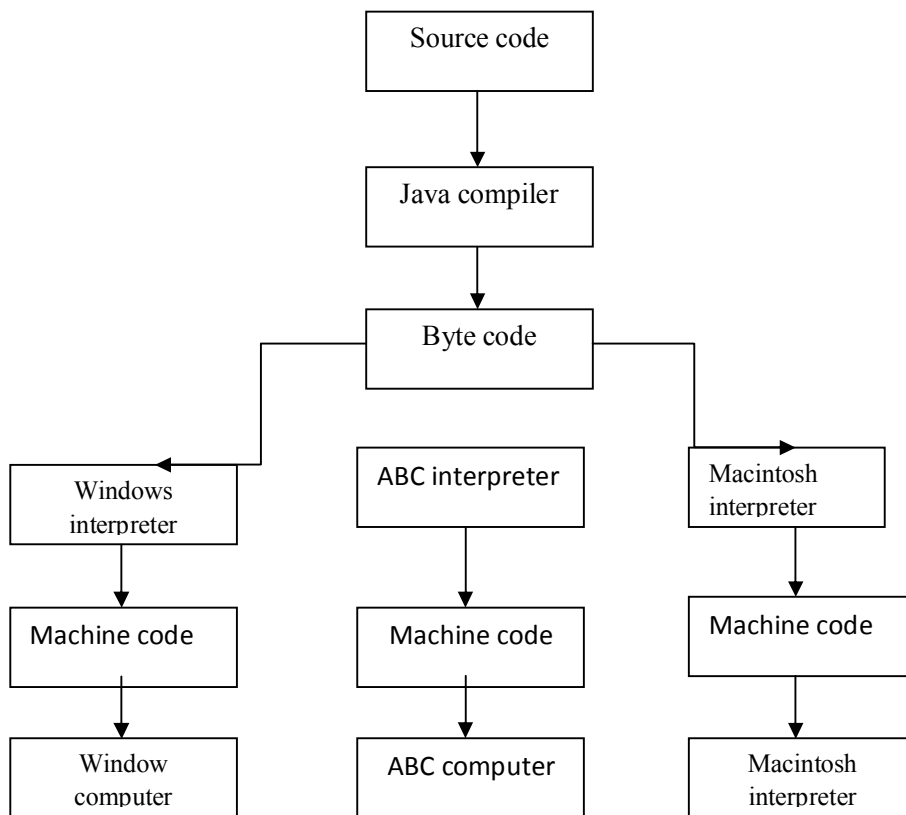
```
c:\> cd java
```

```
c:\java>java Test.java
```

```
c:\java>java Test
```

```
hello world
welcome to the world of Java
let us learn Java
```





Implementation of Java programs

1.2.6. Java Virtual Machine (JVM)

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed. JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).

What is JVM

1. **A specification** where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Oracle and other companies.
2. **An implementation** Its implementation is known as JRE (Java Runtime Environment).
3. **Runtime Instance** Whenever you write java command on the command prompt to run the java class, an instance of JVM is created.

What it does

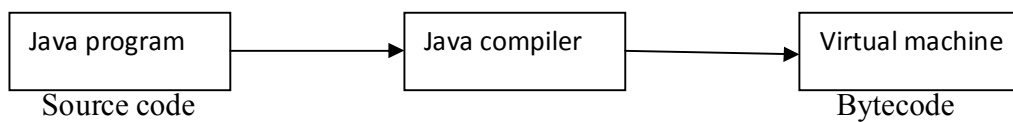
The JVM performs following operation:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

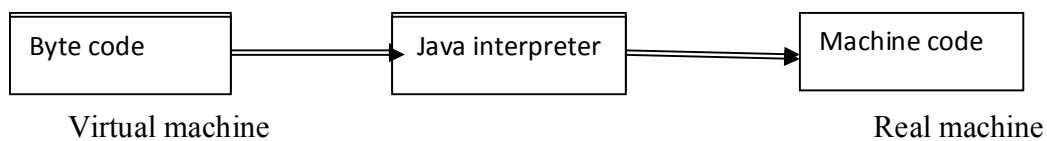
JVM provides definitions for the:

- Memory area
- Class file format
- Register set
- Garbage-collected heap
- Fatal error reporting etc.

The following illustrates the process of compiling a java program into bytecode which is also referred to as **virtual machine code**.



Process of compilation



Process of converting bytecode into machine code

1.2.7. Command Line Argument

A command-line argument is an information that directly follows the program's name on the command line when it is executed. To access the command-line arguments inside a Java program is quite easy. They are stored as strings in the String array passed to main().

Example

The following program displays all of the command-line arguments that it is called with -

```
class test
{
    Public static void main(String[] args)
    {
        for(int i=0,i<args.length;i++)
        {
            System.out.println(args[i]);
        }
    }
}
```

When the program is executed output will be

10

20

30

1.3 Variables and Data types

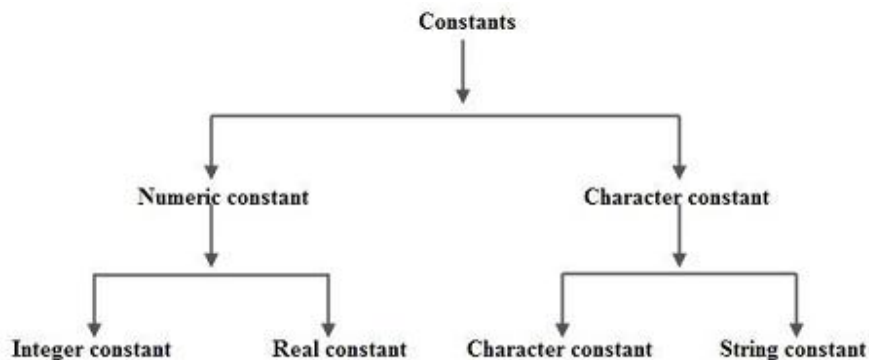
Variables: Variables are the names you give to computer memory locations which are used to store values in a computer program.

For example, assume you want to store two values 10 and 20 in your program and at a later stage, you want to use these two values. Let's see in detail in 1.3.1 section given below.

Data types: a very simple but very important concept available in almost all the programming languages which is called **data types**. As its name indicates, a data type represents a type of the data which you can process using your computer program. It can be numeric, alphanumeric, decimal, etc.

1.3.1. Constants, Variables and Data types

Constants: Constants in Java refer to fixed values that do not change during the execution of a program. Java supports several types of constants such as integer constants, real constants, single character constants, string constants and backslash character constants.



Integer Constants: An integer constant is a sequence of digits from 0 to 9 without decimal points or fractional part or any other symbols. There are 3 types of integers namely decimal integer, octal integers and hexadecimal integer.

Decimal Integers consists of a set of digits 0 to 9 preceded by an optional + or - sign. Spaces, commas and non digit characters are not permitted between digits. Example for valid decimal integer constants are **123 -234 0 are a decimal integer constant**

Octal Integers constant consists of any combination of digits from 0 through 7 with a O at the beginning. Some examples of octal integers are

Hexadecimal integer constant is preceded by OX or Ox, they may contain alphabets from A to F or a to f. The alphabets A to F refers to 10 to 15 in decimal digits. Example of valid hexadecimal integers are **Ox12 x 0x2 are Hexa-Decimal integer constant**

Real Constants

Real Constants consists of a fractional part in their representation. Integer constants are inadequate to represent quantities that vary continuously. These quantities are represented by numbers containing fractional parts like 26.082. Example of real constants are

0.0065 -0.206 5.06

Real Numbers can also be represented by exponential notation. The general form for exponential notation is mantissa exponent. The mantissa is either a real number expressed in decimal notation or an integer. The exponent is an integer number with an optional plus or minus sign.

Single Character Constants

A Single Character constant represent a single character which is enclosed in a pair of quotation symbols.

Example for character constants are **'5' 'c' ';' ''**

All character constants have an equivalent integer value which are called ASCII Values.

String Constants

A string constant is a set of characters enclosed in double quotation marks. The characters in a string constant sequence may be a alphabet, number, special character and blank space. Example of string constants are "VISHAL" "1234" "God Bless" "!.....?"

Backslash Character Constants [Escape Sequences]

Backslash character constants are special characters used in output functions. Although they contain two characters they represent only one character. Given below is the table of escape sequence and their meanings.

Constant	Meaning
'\a'	.Audible Alert (Bell)
'\b'	.Backspace
'\f'	.Formfeed
'\n'	.New Line
'\r'	.Carriage Return
'\t'	.Horizontal tab
'\v'	.Vertical Tab
'\''	.Single Quote
'\"'	.Double Quote
'\?'	.Question Mark
'\\'	.Back Slash
'\0'	.Null

Variables:

A variable is a container which holds the value while the java program is executed. A variable is assigned with a datatype.

Variable is a name of memory location. There are three types of variables in java: local, instance and static.

Types of Variables

There are three types of variables in java:

- local variable
- instance variable
- static variable

1) Local Variable

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

2) Instance Variable

A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static.

It is called instance variable because its value is instance specific and is not shared among instances.

3) Static variable

A variable which is declared as static is called static variable. It cannot be local. You can create a single copy of static variable and share among all the instances of the class. Memory allocation for static variable happens only once when the class is loaded in the memory.

1. **class** A{
2. **int** data=50;//instance variable
3. **static int** m=100;//static variable
4. **void** method(){
5. **int** n=90;//local variable
6. }

Data types

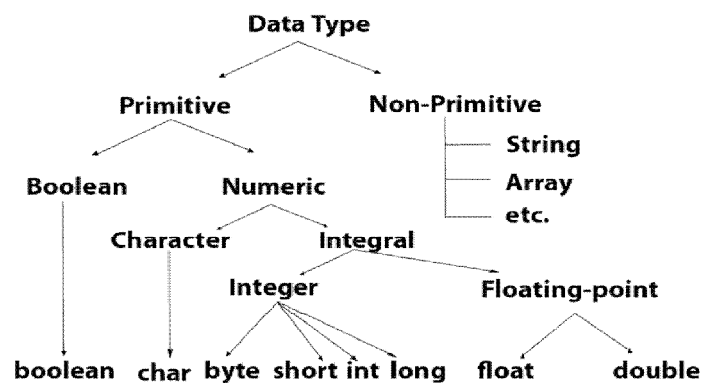
Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.

There are 8 types of primitive data types:

- boolean data type
- byte data type
- char data type
- short data type
- int data type
- long data type
- float data type
- double data type



Data Type	Default Value	Default size
Boolean	false	1 bit
Char	'\u0000'	2 byte

Byte	0	1 byte
Short	0	2 byte
Int	0	4 byte
Long	0L	8 byte
Float	0.0f	4 byte
Double	0.0d	8 byte

Boolean Data Type

The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.

The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.

Example: Boolean one = false

Byte Data Type

The byte data type is an example of primitive data type. It is an 8-bit signed two's complement integer. Its value-range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0.

The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type.

Example: byte a = 10, byte b = -20

Short Data Type

The short data type is a 16-bit signed two's complement integer. Its value-range lies between -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.

The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer.

Example: short s = 10000, short r = -5000

Int Data Type

The int data type is a 32-bit signed two's complement integer. Its value-range lies between -2,147,483,648 (-2^{31}) to 2,147,483,647 ($2^{31} - 1$) (inclusive). Its minimum value is -2,147,483,648 and maximum value is 2,147,483,647. Its default value is 0.

The int data type is generally used as a default data type for integral values unless if there is no problem about memory.

Example: int a = 100000, int b = -200000

Long Data Type

The long data type is a 64-bit two's complement integer. Its value-range lies between -9,223,372,036,854,775,808 (-2^{63}) to 9,223,372,036,854,775,807 ($2^{63} - 1$) (inclusive). Its minimum value is -9,223,372,036,854,775,808 and maximum value is 9,223,372,036,854,775,807. Its default value is 0. The long data type is used when you need a range of values more than those provided by int.

Example: long a = 100000L, long b = -200000L

Float Data Type

The float data type is a single-precision 32-bit IEEE 754 floating point. Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F.

Example: float f1 = 234.5f

Double Data Type

The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.

Example: double d1 = 12.3

Char Data Type

The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive). The char data type is used to store characters.

Example: char letterA = 'A'

1.3.2. Declaration of variables and giving values to variables

A variable provides us with named storage that our programs can manipulate. Each variable in Java has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

You must declare all variables before they can be used. Following is the basic form of a variable declaration

data type variable[= value][,variable [= value]...];

Here *data type* is one of Java's datatypes and *variable* is the name of the variable. To declare more than one variable of the specified type, you can use a comma-separated list.

Following are valid examples of variable declaration and initialization in Java –

Example

```
int a,b,c;           //declares three ints, a,b,c.
int a=10,b=20;      // example of initialization.
byte B=34;          // initializes a byte type variable B.
double pi=3.14759;  // declares and assigns a value of PI.
char a ='a'         // the char variable a is initialized with value 'a'.
```

1.3.3. Scope of variables

There are three kinds of variables in Java which we have studied in 1.3.1. here we will study in detail.

- Local variables
- Instance variables
- Class/Static variables

Local Variables

- Local variables are declared in methods, constructors, or blocks.
- Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.

- Access modifiers cannot be used for local variables.
- Local variables are visible only within the declared method, constructor, or block.
- Local variables are implemented at stack level internally.
- There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.

Example

Here, *age* is a local variable. This is defined inside *pupAge()* method and its scope is limited to only this method.

Public class Test

```
{
    public void pupAge()
    {
        Int age=0;
        age = age + 7;
        System.out.println(" puppy age is : " + age);
    }
    public static void main(String args[] )
    {
        Test test = new Test( );
        Test.pupAge();
    }
}
```

output

puppy age is : 7

in the above program it will show error while compiling it (say : variable number might not have been initialized)

Instance Variables

- Instance variables are declared in a class, but outside a method, constructor or any block.
- When a space is allocated for an object in the heap, a slot for each instance variable value is created.
- Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
- Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.
- Instance variables can be declared in class level before or after use.
- Access modifiers can be given for instance variables.
- The instance variables are visible for all methods, constructors and block in the class. Normally, it is recommended to make these variables private (access level). However, visibility for subclasses can be given for these variables with the use of access modifiers.
- Instance variables have default values. For numbers, the default value is 0, for Booleans it is false, and for object references it is null. Values can be assigned during the declaration or within the constructor.
- Instance variables can be accessed directly by calling the variable name inside the class. However, within static methods (when instance variables are given accessibility), they should be called using the fully qualified name. *ObjectReference.VariableName*.

Example

```
Import java.io.*;

public class Employee{

// this instance variable is visible for any child class.

public String name;

//salary variable is visible in Employee class only.

private double salary;

// the name variable is assigned in the constructor.
```

```

public void setSalary(double empSal) {
    salary = empSal;
}

//this method prints the employee details
public void printEmp() {
    System.out.println("name : ' + name);
}

public static void main( String args[ ] ) {
    Employee empOne = new Employee(" prasanna");
    empOne.setSalary(100000);
    empOne.printEmp();
}
}

```

output

```

name : prasanna
salary : 100000.0

```

Class/Static Variables

- Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
- There would only be one copy of each class variable per class, regardless of how many objects are created from it.
- Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final, and static. Constant variables never change from their initial value.
- Static variables are stored in the static memory. It is rare to use static variables other than declared final and used as either public or private constants.

- Static variables are created when the program starts and destroyed when the program stops.
- Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class.
- Default values are same as instance variables. For numbers, the default value is 0; for Booleans, it is false; and for object references, it is null. Values can be assigned during the declaration or within the constructor. Additionally, values can be assigned in special static initializer blocks.
- Static variables can be accessed by calling with the class name *ClassName.VariableName*.
- When declaring class variables as public static final, then variable names (constants) are all in upper case. If the static variables are not public and final, the naming syntax is the same as instance and local variables

1.3.5. Symbolic constants

Constants may appear repeatedly in number of places in the program. Constant values are assigned to some names at the beginning of the program, then the subsequent use of these names in the program has the effect of caving their defined values to be automatically substituted in appropriate points. The constant is declared as follows:

Syntax : final type symbolicname= value;

Eg final float PI =3.14159;
final int STRENGTH =100;

Rules :-

symbolic names take the some form as variable names. But they one written in capitals to distance from variable names. This is only convention not a rule. After declaration of symbolic constants they shouldn't be assigned any other value with in the program by using an assignment statement. Consider the below example

STRENGTH = 200 is illegal

Symbolic constants are declared for types there are not done in C & C++ where symbolic constants are defined using the define statement. They can't be declared inside a method. They should be used only as class data members in the beginning of the class.

1.3.6. Type casting

Assigning a value of one type to a variable of another type is known as **Type Casting**.

Example:

```
int x = 10;
byte y = (byte) x;
```

In Java, type casting is classified into two types,

- Widening Casting(Implicit)



- Narrowing Casting(Explicitly done)



Widening or Automatic type conversion

Automatic Type casting takes place when,

- the two types are compatible
- the target type is larger than the source type

Example

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

```

    {
        int I = 100; //no explicit type casting required
        long l = I; // no explicit type casting required
        float f = 1;
        System.out.println(" int value" + i);
        System.out.println( " Long value" + 1);
        System.out.println( "Float value" + f );
    }
}

```

Output

```

Int value 100
Long value 100
Float value 100.0

```

Narrowing or Explicit type conversion

When you are assigning a larger type value to a variable of smaller type, then you need to perform explicit type casting.

Example :

```

public class Test
{
    public static void main(String[ ] args)
    {
        double d = 100.04;
        long l = (long)d; //explicit type casting required
        int i = (int) l; //explicit type casting required
        System.out.println(" Double value" + d);
        System.out.println( " Long value" + 1);
        System.out.println( "Int value" + i );
    }
}

```

Output

```

Double value 100.04
Long value 100
Int value 100

```

1.4. Summary

1. **Java** is an object-oriented, cross platform, multi-purpose **programming** language produced by Sun Microsystems.
2. First released in 1995, it was developed to be a machine independent web technology.
3. It was based on C and C++ syntax to make it easy for programmers from those communities to learn.
4. A class is a template that describes the data and behavior associated with an instance of that class.
5. An object is an instance of a class.
6. Inheritance: A class can be derived from another class. In this case this class is called a *subclass*. Another common phrase is that *a class extends another class*
7. *Variables* allow the Java program to store values during the runtime of the program.
8. Instance variable is associated with an instance of the class (also called object). Access works over these objects.
9. A method is a block of code with parameters and a return value. It can be called on the object.
10. There are four access levels: *public*, *protected*, *default* and *private*.

1.5. SAQ

1. Explain OOP paradigm?
2. Write a short note on application of OOPs
3. “Java is said to be JVM”, support your answer.
4. Explain variable, data types in java?
5. Write a short note on type casting in Java?
6. Explain Java tokens?
7. Explain OOPs?

UNIT II – Decision Making and branching

Structure

- 2.0 Objectives
 - 2.1 Introduction
 - 2.2 Decision making with if statement
 - 2.2.1. if statement
 - 2.2.2. if... else statement
 - 2.2.3. nesting of if...else statements
 - 2.2.4. the else if ladder
 - 2.2.5. the switch statement
 - 2.2.6. the ? : operator
 - 2.3. Decision making and looping
 - 2.3.1. the while statement
 - 2.3.2. the do statement
 - 2.3.3. the for statement
 - 2.3.4. jumps in loops
 - 2.4. Summary
 - 2.5. SAQ
-

2.0 Objective

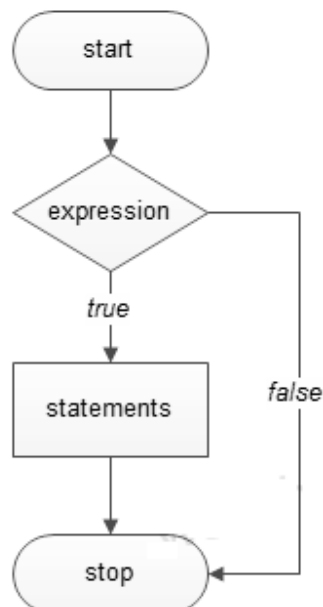
This Unit discusses about Decision making and looping are the concepts which works as a control statement in Java. Control statements in Java are the instructions which can control the flow of execution of a program. In control statement, there are some conditions which are specified by a programmer. These conditions are evaluated by the system and a particular block of statements are executed.

2.1. Introduction

Decision making in programming is similar to decision making in real life. In programming also we face some situations where we want a certain block of code to be executed when some condition is fulfilled. A programming language uses control statements to control the flow of execution of program based on certain conditions. These are used to cause the flow of execution to advance and branch based on changes to the state of a program.

2.1. Decision making with if statement

Java decision-making statements allow you to make a decision, based upon the result of a condition. All the programs in Java have set of statements, which are executed sequentially in the order in which they appear. It happens when jumping of statements or repetition of certain calculations is not necessary. However, there may arise some situations where programmers have to change the order of execution of statements based on certain conditions which involve kind of decision-making statements. In this chapter, you will learn about how the control flow statements work. The following flowchart shows the decision-making technique in Java



Java has such decision-making capabilities within its program by the use of following the decision making statements:

If statement

If...else statement

Nesting of if...else

Else if ladder

Switch statement

The ? : operator

2.2.1. if statement

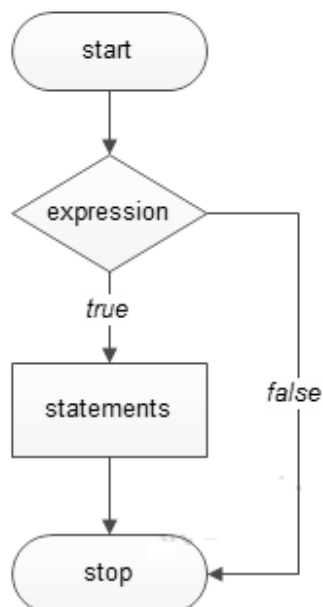
If statements in Java is used to control the program flow based on some condition, it's used to execute some statement code block if the expression evaluated to true; otherwise, it will get skipped. This statement is the simplest way to modify the control flow of the program. The following is the basic format of “if statement”.

Syntax:

```
If(expression)
    {
        Statement 1;
        Statement 2;
        ....
    }
```

'Statement n' can be a statement or a set of statements, and if the test expression evaluated to true, the statement block will get executed, or it will get skipped.

Flowchart of if Statement



Example

```
public class sample
{
    public static void main(String args[] )
    {
        int a=20, b=30;
        if(b>a)
            System.out.println("b is greater");
    }
}
```

Output

b is greater

2.2.2. if...else statement

If else statements in Java is also used to control the program flow based on some condition, only the difference is: it's used to execute some statement code block if the expression is evaluated to true, otherwise executes else statement code block.

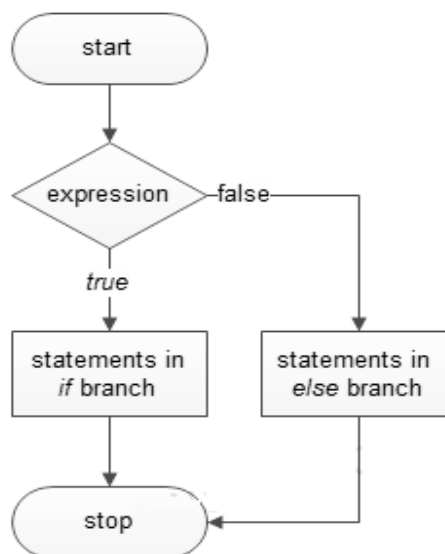
The basic format of if...else statement is as follows:

Syntax:

```
if (test_expression)
```

```
{
    //execute your code
}
else
{
    //execute your code
}
```

Flowchart of if...else



Example

```
public class sample
{
    public static void main(String args[] )
    {
        int a=20, b=30;
        if(b > a)
        {
            system.out.println("b is greater");
        }
        else
        {
            System.out.println("a is greater");
        }
    }
}
```

Output

a is greater

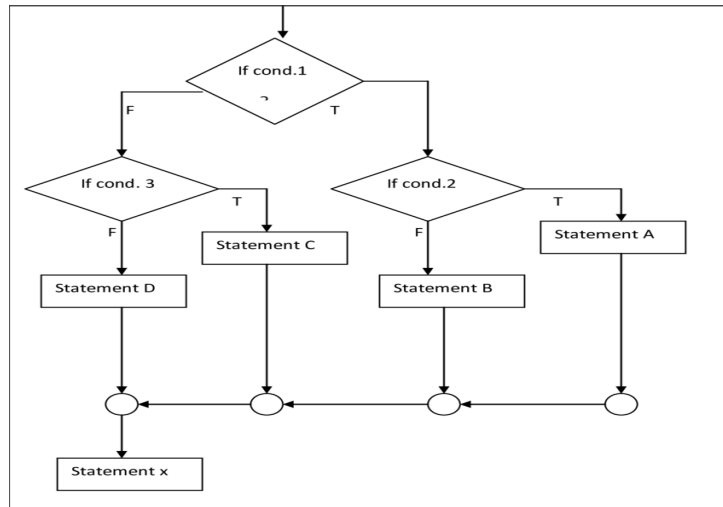
2.2.3. nesting of if...else

A nested if is an if statement that is the target of another if or else. Nested if statements means an if statement inside an if statement. Yes, java allows us to nest if statements within if statements. i.e, we can place an if statement inside another if statement.

Syntax

```
if (condition)
{
    //body of parent if
    if ( condition)
    {
        //body of nested if
    }
}
```

Flow chart of nested if



Example

// Java program to illustrate nested-if statement

```
class NestedIfDemo
```

```
{
```

```
    public static void main(String args[])
```

```
    {
```

```
        int i = 10;
```

```
        if (i == 10)
```

```
        {
```

```
            // First if statement
```

```
            if (i < 15)
```

```
                System.out.println("i is smaller than 15");
```

```
            // Nested - if statement
```

```
            // Will only be executed if statement above
```

```
            // it is true
```

```
            if (i < 12)
```

```
                System.out.println("i is smaller than 12 too");
```

```
            else
```

```
                System.out.println("i is greater than 15");
```

```
        }
```

```
    }
```

```
}
```

Output:

```
i is smaller than 15
```

```
i is smaller than 12 too
```

2.2.4. the else... if ladder

A nested if is an if statement that is the target of another if or else. Nested if statements means an if statement inside an if statement. Yes, java allows us to nest if statements within if statements. i.e, we can place an if statement inside another if statement.

Syntax:

```
if( condition 1)
{
    //executes when condition1 is true
else if(condition2)
    {
        //executes when condition2 is true
    }
else
{
    //execute your code
}
```

Example

```
public class sample
{
    public static void main(String args[] )
    {
        int a=20, b=30;
        if(b > a)
        {
            system.out.println("b is greater");
        }
        else if ( a> b)
        {
            System.out.println("a is greater");
        }
        else
        {
            System.out.println( "both are equal");
        }
    }
}
```

Output

Both are equal

2.2.5. The switch statement

Java switch statement is used when you have multiple possibilities for the if statement.

Syntax

```
Switch( variable)
{
    case 1:
        //execute your code
        break;

    case 2:
        //execute your code
        break
    :
    :

    case n:
        //execute your code
        break;
    default:
        //execute your code
        break;
}
```

After the end of each block it is necessary to insert a break statement because if the programmers do not use the break statement, all consecutive blocks of codes will get executed from each case onwards after matching the case block.

Example of a Java Program to Demonstrate Switch Statement

Example

```
public class sample
{
    public static void main( String args[] )
    {
        int a= 5;
        switch ( a)
        {
            Case 1:
                System.out.println(“ you chose one”);
                break;
```



```

case 1:
    System.out.println(" you chose one");
    break;
case 2:
    System.out.println(" you chose two");
    break;
case 3:
    System.out.println(" you chose three");
    break;
case 4:
    System.out.println(" you chose four");
    break;
case 5:
    System.out.println(" you chose five");
    break;
default:
    System.out.println(" Invalid choice.Enter a no between 1 and 5");
    break;
}
}
}

```

output
You chose five

When none of the cases is evaluated to *true*, the default case will be executed, and *break statement* is not required for default statement.

2.2.6. the ? : operator

When none of the cases is evaluated to *true*, the default case will be executed, and *break statement* is not required for default statement.

`expr1 ? expr2 : expr3`

where `expr1` is a boolean expression and `expr2` and `expr3` are the expressions of any type other than void. The `expr2` and `expr3` must be of the same type.

If `expr1` has value `true`, the operator returns a result `expr2` .
If `expr1` has value `false`, the operator returns a result `expr3` . During the calculation of the first argument . if it has the true value , then calculates the second (middle) argument returned as a result . However , if the calculated result of the first argument is false, then it is

given the third (last) argument the returned as a result. Here is an example of the use of the operator " ? " :

```
//Program to Find greatest of three numbers using Conditional Operator
import java.util.Scanner; //program uses Scanner class
public class ConditionalOperator
{
    public static void main(String[] args)
    {
        int a,b,c,result;
        //create Scanner object to obtain input from keyboard
        Scanner input=new Scanner(System.in);
        System.out.print("Enter the Three Number : "); //prompt for input
        a=input.nextInt(); //Read First number
        b=input.nextInt(); //Read Second number
        c=input.nextInt(); //Read third number
        result = (a>b)? ((a>c)?a:c) : ((b>c)?b:c);
        System.out.println( result + " is Greatest");
    }
}
```

2.3. Decision making and looping

Looping statements allow the programmer to **execute some group of statement repetitively** multiple times.

Control goes inside the body of the loop if the condition is true otherwise goes outside of looping block. The main difference between decision making statements and looping statement is decision making statement execute once alike looping statements executes several times.

There are two types of loop

1. Entry control loop (while loop and for loop)
2. Exit control loop (do ... while loop)

Entry control loops are the loop in which **condition is tasted at the beginning** of the loop.

Exit control loops are the loop which **condition is tasted at the end** of the loop.

Do ... while is an exit control loop. For and while are entry control loop.

Note: Exit control loops are executed at least once in their lifetime.

Advantages of looping:

- Reduce the memory consumption.
- Reduce the length of the code.
- No need to write same code again to execute several times

Following are the **types of looping** statements

- while loop
- do ... while loop
- for loop
- nested loops

2.3.1. The while statement

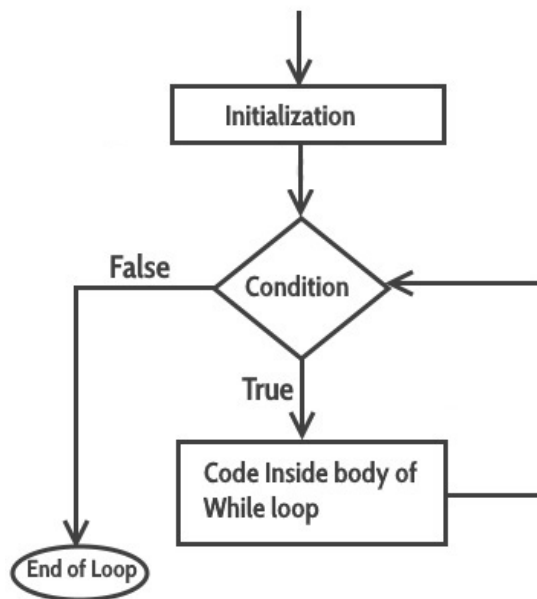
While loop is an entry control looping statement that allows code to be executed until the boolean condition is true.

You can also call while loop as a repeating if statement.

Syntax :

```
// assignment of counter variable
while ( condition )
{
    // body of loop
    // increment or decrement of variable
}
```

Flow chart for while statement



Example

Class whileloop

```

{
    public static void main(String args[])
    {
        int i = 0;
        while( i < 10)
        {
            //this loop will be executed 10 times
            System.out.println(i);
            i++; //increment of counter variable
        }
    }
}
  
```

Output:

0
1
2

3
4
5
6
7
8
9

In above example, while loop is executed 10 times. Initially counter variable i is assigned by value 0 then the loop is executed 10 times and each time counter variable i is incremented by 1.

2.3.2. the do statement

Do ... while is an exit control looping statement which is also used to execute a block of instructions several times. This loop is executed at least once in its lifetime whether the condition is true or false.

Syntax

// assignment of counter variable

do (condition)

{

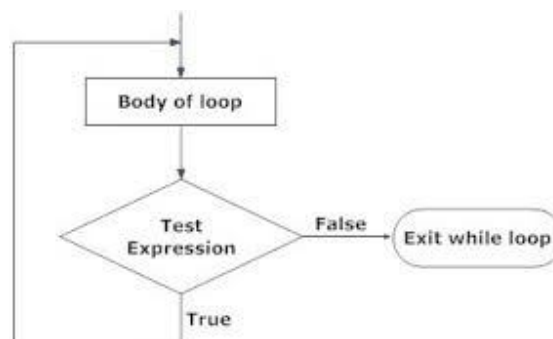
 // body of statement

 // increment or decrement of counter variable

}

While (condition); // statement must be ended with semicolon here

Flow chart



Class whileloop

```
{
    public static void main(String args[])
    {
        int i = 0;
    do
    {
        //this loop will be executed 10 times
        System.out.println(i);
        i++; //increment of counter variable
    }
    While ( i < 10 );
    }
}
```

Output

0
1
2
3
4
5
6
7
8
9

2.3.3. The for statement

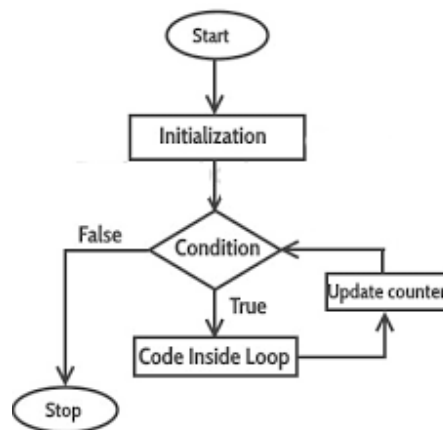
For loop is an entry control loop which works same as the other two loop but its syntax is different. In other two loop counter variable is declared outside the body of the loop. Here, the counter variable is declared within for loop itself but the counter variable is declared and assigned by initial value only once. Increment and condition checks will be done multiple times.

Syntax:

for (initialization; condition ; increment/decrement)

```
{  
    // assignment of counter variable  
    // body of statement  
    // increment or decrement of counter variable  
}
```

Flow chart



Example:

Class forloop

```
{  
    public static void main(String args[])  
    {  
        for(int i = 0; i<10;i++)  
        {  
            //this loop will be executed 10 times  
            System.out.println(i);  
        }  
    }  
}
```

```
}
```

Output

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

2.3.4. Jump in loop

Loops perform a set of operations repeatedly until the control variable fails to satisfy the test condition. The number of times a loop is repeated is decided in advance and the condition is written to achieve this. Sometime, when executing a loop it becomes desirable to skip a part of the loop or to leave the loop as soon as certain condition occurs. For example, consider the case of searching for a particular name in a list containing, like 100 names. A program loop written for reading and testing the names a 100 times must be terminated as soon as the desired name is found. Java permits a jump from one statement to the end or beginning of a loop as well as a jump out of a loop.

Jumping out of a loop

In the previous topics we saw that loop can accomplished by using the break statement. Even we come across the break in switch statement. This statement can also be used within while, do or for loops too. Consider the below examples with explains jumping in loops and jump out of a loop.

```
While( condition)  
{  
    .....  
    .....
```



```
if(condition)
    break;
    .....
    .....
}
```

Skipping a part of a loop

During the loop operations, it may be necessary to skip a part of the body of the loop under certain conditions. For example, in processing of applications for some job, we might like to exclude the processing of data of applicants belonging to a certain category. On reading the category code of an applicant, a test is made to see whether his application should be considered or not. If its is not to be considered, the part of the program loop that processes the application details is skipped and the execution continues with the next loop operation.

2.4 Summary

1. An 'If' statement decides whether to execute a statement or which statement has to execute first between the two.
2. In Java, the control statements are divided into three categories which are selection statements, iteration statements, and jump statements.
3. List of Different control statements in C Programming: Do check it out here. The if, else, switch, case and default are used for selection purposes. The do, while and for are used for iterative purposes.
4. The goto, break, continue and return are used for jumping purposes.
5. The main types of control statements are FOR, WHILE, IF and CASE.
6. The FOR statement is used to execute one or more statements repeatedly, while incrementing or decrementing a variable with each repetition, until a condition is met.

2.5. SAQ

1. Write a short note on condition statement in Java?
2. Explain briefly control statements in Java?
3. List the difference between do..while and while statements in Java?

Unit – III Classes, Objects and Methods

3.0. Structure

3.1. Objective

3.2. Class, Objects and Methods

3.2.1. Defining a class

3.2.2. Fields declaration

3.2.3. Methods declaration

3.2.4. Creating objects

3.2.5. Accessing class members

3.2.6. Constructors

3.2.7. Methods Overloading

3.2.8. Static members

3.2.9. Nesting of methods

3.2.10. Inheritance

3.2.11. Overriding methods

3.2.12. Final variables, methods and Final classes

3.2.13. Abstract methods and classes, Visibility control

3.3. Arrays

3.3.1. One dimensional Array

3.3.2. Two dimensional arrays

3.4. Strings

3.5. Vectors

3.6. Wrapper classes

3.7. Enumerated types

3.8. Summary

3.9. SAQ

3.1. Objective

Java is a true Object-Oriented language and therefore the underlying structure of all Java programs concentrates on classes. Anything that is represented in a Java program must be encapsulated in a class that defines the state and behavior of the basic program components known as Objects. Classes create objects and objects use methods to communicate between them. This makes the sense of Object Oriented Programming. In this unit we concentrate on what are Classes, objects, methods and arrays in detail.

3.2. Class, Objects and Methods

3.2.1. Defining a class

A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:

1. **Modifiers** : A class can be public or has default access.
2. **Class name**: The name should begin with a initial letter.
3. **Superclass (if any)**: The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
4. **Interfaces(if any)**: A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
5. **Body**: The class body surrounded by braces, { }.

3.2.2. Field Declaration

Data is encapsulated in a class by placing data fields inside the body of the class definition. The variables are called instance variables because they are created whenever an object of the class instantiated. We can declare the instance variables exactly the same way as we declare local variables. Instance variables are also known as member variables. We can also declare the instance variables exactly the same way as we declare local variables. Consider the following example for field declaration

Class Area

```

{
    int length;
    int breadth;
}

```

In the above example Area is the name of the class which consists of two variables(length and breadth) of integer type.

Note: the variables declared above does not have any storage space in the memory.

3.2.3. Methods declaration

A method is a collection of statements that perform some specific task and return the result to the caller. A method can perform some specific task without returning anything. Methods allow us to **reuse** the code without retyping the code. In Java, every method must be part of some class which is different from languages like C, C++, and Python. Methods are **time savers** and help us to **reuse** the code without retyping the code.

In general, method declarations has six components:

- **Modifier-**: Defines **access type** of the method i.e. from where it can be accessed in your application. In Java, there 4 type of the access specifiers.
 - public: accessible in all class in your application.
 - protected: accessible within the class in which it is defined and in its **subclass(es)**
 - private: accessible only within the class in which it is defined.
 - default (declared/defined without using any modifier) : accessible within same class and package within which its class is defined.
- **The return type** : The data type of the value returned by the method or void if does not return a value.
- **Method Name** : the rules for field names apply to method names as well, but the convention is a little different.
- **Parameter list** : Comma separated list of the input parameters are defined, preceded with their data type, within the enclosed parenthesis. If there are no parameters, you must use empty parentheses ().

- **Exception list** : The exceptions you expect by the method can throw, you can specify these exception(s).
- **Method body** : it is enclosed between braces. The code you need to be executed to perform your intended operations.

Consider the following simple example

```

Class rectangle
{
int length;
int breadth;
void getdata(int x, int y) //method declaration
    {
    length=x;
    breadth=y;
    }
int rectarea() // declaration of another method
{
int area = length * breadth;
return(area);
}
}

```

3.2.4. Creating Objects

It is a basic unit of Object Oriented Programming and represents the real life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of :

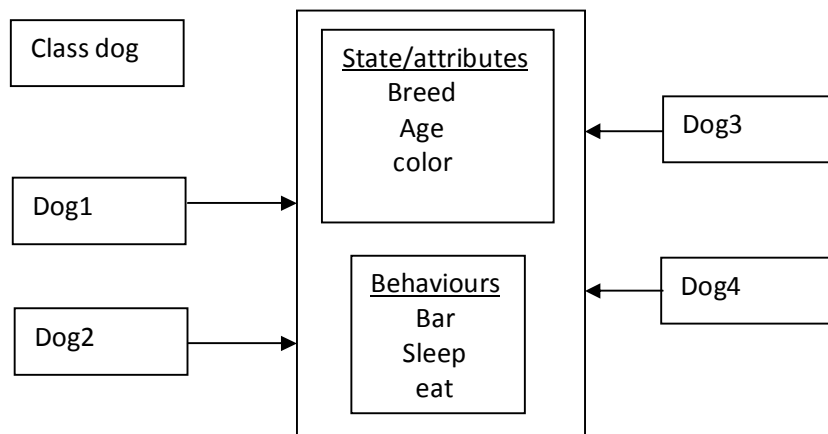
1. **State/attribute** : It is represented by attributes of an object. It also reflects the properties of an object.
2. **Behavior** : It is represented by methods of an object. It also reflects the response of an object with other objects.
3. **Identity** : It gives a unique name to an object and enables one object to interact with other objects.

Example of an object : dog

<u>Identity</u>	<u>State/ Attribute</u>	<u>Behaviors</u>
Name of the dog	Breed	Bark
	Age	Sleep
	Color	eat

Objects correspond to things found in the real world. For example, a graphics program may have objects such as “circle”, “square”, “menu”. An online shopping system might have objects such as “shopping cart”, “customer”, and “product”.

Declaring Objects (Also called instantiating a class). When an object of a class is created, the class is said to be **instantiated**. All the instances share the attributes and the behavior of the class. But the values of those attributes, i.e. the state are unique for each object. A single class may have any number of instances.



Objects in Java are created using the “new” operator. The “new” operator created an object of the specified class and returns a reference to that object. Consider the following example for creating an object of type Rectangle.

```
Rectangle rect1;           // declare the object
Rect1 = new Rectangle();  //instantiate the object
```

The first statement declares a variable to hold the object reference and the second one actually assigns the object reference to the variable. The variable “rect1” is now an object of the Rectangel class. In the above example Rectangle() is the default constructor of the class. We create any number of objects of Rectangle. For example

```
Rectangle rect1 = new Rectangle();
Rectangle rect2 = new Rectangle();
```

It is important to understand that each object has its own copy of the instance variables of its class. This means that any changes to the variables of one object have not effect on the variables of another is also possible to create two or more references to the same object.

Initializing an object

The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The new operator also invokes the class constructor.

```
// Class Declaration
```

```
public class Dog
{
    // Instance Variables
    String name;
    String breed;
    int age;
    String color;

    // Constructor Declaration of Class
    public Dog(String name, String breed, int age, String color)
    {
        this.name = name;
        this.breed = breed;
        this.age = age;
        this.color = color;
    }

    // method 1
    public String getName()
    {
        return name;
    }

    // method 2
    public String getBreed()
    {
        return breed;
    }

    // method 3
    public int getAge()
    {
        return age;
    }
}
```

```

// method 4
public String getColor()
{
    return color;
}

@Override
public String toString()
{
    return("Hi my name is "+ this.getName()+"\nMy breed,age and color are " +
        this.getBreed()+";" + this.getAge()+ ";" + this.getColor());
}

public static void main(String[] args)
{
    Dog tuffy = new Dog("tuffy","papillon", 5, "white");
    System.out.println(tuffy.toString());
}
}

```

Output

Hi my name is tuffy.

My breed, age and color are paillon,5,white

This class contains a single constructor. We can recognize a constructor because its declaration uses the same name as the class and it has no return type. The Java compiler differentiates the constructors based on the number and the type of the arguments. The constructor in the *Dog* class takes four arguments. The following statement provides “tuffy”, “papillon”, 5, “white” as values for those arguments:

```
Dog tuffy= new Dog(“tuffy”,“papillin”,5,“white”);
```

3.2.5. Accessing class members

In the previous topic we have gone through objects, each containing its own set of variables, how to assign values to the variable in the program. All variables must be assigned values before they are used. Since we are outside the class, we cannot access the instance variables and the methods directly. In order to access variable we must use the concerned object and the dot operator as shown below

Syntax

```
Objectname.variablename = value;  
Objectname.methodname(parameter-list);
```

Objectname is the name of the object, variablename is the name of the instance variable inside the object that we wish to access, methodname is the method that we wish to call, and parameter-list is a common separated list of “actual values” (or expressions) that must match in type and number with the parameter list of the methodname declare in the class.

For example instance variables of the class “Rectangle” may be accessed and assigned values as follows

```
Rect1.lenght =15  
Rect1.breadth =90  
Rect2.lenght =24  
Rect2.breadth =25
```

Note that the two objects rect1 and rect2 store different values.

Suppose if we are using the method “getdata”, we can call the “getdata” on any “Rectangle” object to set the values of both length and breadth. The following is the segment to achieve this.

```
Rectangle rect1 = new Rectangle( ); // creating an object  
rect1.getdata( 25, 23); // calling the method using the object
```

the above code created “rect1” object and then passes in the values 25 and 23 for x and y parameters of the method “getdata”. This method then assigns these values to length and breadth variables respectively. Consider the following method again

```
void getdata( int x, int y)  
{  
    Length = x;  
    Breadth = y;  
}
```

Now the object “rect1” contains values for its variables, we can compute the are of the rectangle represented by “rect1”. The above can also be done in other ways which is as follows.

- In the first approach to acces the instance variables using the dot operator and compute the “area”

```
int area1 = rect1.lenght * rect1.breadth;
```

- In the second approach is to call the method rectarea declared inside the class.

```
Int area1 = rect1.rectarea(); // calling the method
```

Let us consider the following example

Class Rectangle

```
{
    int length, breadth;           // declaration of variables
    void getdata( int x, int y)    //definition of method
    {
        length = x;
        breadth = y;
    }
    int rectarea()                 // definition of another method
    {
        int area = lenth * breadth;
        return( area);
    }
}
Class AreaRect                     // class with main method
{
    public static void main(String args[ ])
    {
        int area1,area2;
        Rectangle rect1 = new Rectangle(); // creating objects
        Rectangle rect2 = new Rectangle();
        rect1.length = 25; // accessing variables
        rect2.breadth = 23;
        rect2.getdata (12,15);
        area2 = rect2.rectarea();
        System.out.println("area value for the first area=" + area1);
        System.out.println("area value for the second area=" + area2);
    }
}
```

Output

Area value for the first area = 575

Area value for the second area = 180

3.2.6. Constructors

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory. It is a special type of method which is used to initialize the object. Every time an object

is created using the new() keyword, at least one constructor is called. It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default. There are two types of constructors in Java: no-arg constructor, and parameterized constructor.

Note: It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Rules for creating Java constructor

There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

Note: we can use access modifiers while declaring a constructor. It controls the object creation. In other words, we can have private, protected, public or default constructor in Java.

Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

• Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor

1. <class_name>(){}

Consider the following example for constructor in which we creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
//Java Program to create and call a default constructor
class Bike1 {
//creating a default constructor
Bike1(){
```

```

System.out.println("Bike is created");
}
//main method
public static void main(String args[])
{
//calling a default constructor
Bike1 b=new Bike1();
}
}

```

Output:

Bike is created

Rule: if there is no constructor in a class, compiler automatically creates a default constructor.

Example of default constructor that displays the default value

```

class Student3 {

int id;

String name;

void display(){System.out.println(id+" "+name);}

public static void main(String args[]){

Student3 s1=new Student3();

Student3 s2=new Student3();

s1.display();

s2.display();

}

}

```

Output:

0 null

0 null

Explanation: In the above class, you are not creating any constructor so compiler provides you a default constructor. Here 0 and null values are provided by default constructor.

- **Java Parameterized Constructor**

A constructor which has a specific number of parameters is called a parameterized constructor. Why use the parameterized constructor? The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Example of parameterized constructor

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.

Example for parameterized 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

3.2.7. Method overloading

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**. If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

Advantage of method overloading

- Method overloading increases the readability of the program.

Note: Method overloading is not possible by changing the return type of the method only.

Different ways to overload the method

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

1) **Method Overloading:** changing no. of arguments

In this example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers. We are creating static methods so that we don't need to create instance for calling methods.

```
class Adder{  
    static int add(int a,int b){return a+b;}  
    static int add(int a,int b,int c){return a+b+c;}  
}  
  
class TestOverloading1 {  
    public static void main(String[] args){  
        System.out.println(Adder.add(11,11));  
        System.out.println(Adder.add(11,11,11));  
    }  
}
```

Output:

22

33

2) Method Overloading: changing data type of arguments

In this example, we have created two methods that differs in data type. The first add method receives two integer arguments and second add method receives two double arguments.

```
class Adder{  
    static int add(int a, int b){return a+b;}  
    static double add(double a, double b){return a+b;}  
}  
  
class TestOverloading2 {  
    public static void main(String[] args){
```

```
System.out.println(Adder.add(11,11));  
  
System.out.println(Adder.add(12.3,12.6));  
}}
```

Output:

22

24.9

3.2.8. Static Members

apply **static** keyword with any **method**, it is known as **static method**. A **static method** belongs to the class rather than the object of a class. A **static method** can be invoked without the need for creating an instance of a class. A **static method** can access **static** data member and can change the value of it.

Static Method in Java belongs to the class and not its instances. ... Usually, **static methods** are utility **methods** that we want to expose to be used by other classes without the need of creating an instance.

Static method in Java is a method which belongs to the class and not to the object. A static method can access only static data.

- It is a method which belongs to the class and not to the object(instance)
- A static method can access only static data. It can not access non-static data (instance variables)
- A static method can call only other static methods and can not call a non-static method from it.
- A static method can be accessed directly by the class name and doesn't need any object
- A static method cannot refer to "this" or "super" keywords in anyway

Syntax :

Class-name.method-name

Consider the following java program to demonstrate the use of a static method

```
class Student{
    int rollno;
    String name;
    static String college = "ITS";
    static void change(){
        college = "BBDIT";
    }
    Student(int r, String n){
        rollno = r;
        name = n;
    }
    void display(){System.out.println(rollno+" "+name+" "+college);}
}

public class TestStaticMethod{
    public static void main(String args[]){
        Student.change();//calling change method
        //creating objects
        Student s1 = new Student(111,"Karan");
        Student s2 = new Student(222,"Aryan");
        Student s3 = new Student(333,"Sonoo");
        //calling display method
        s1.display();
    }
}
```

```

s2.display();

s3.display();

}

}

```

Out put

111 karan BBDIT

222 Aryan BBDIT

333 Sono BBDIT

3.2.9. Nesting of Methods

When a **method in java** calls another **method** in the same class, it is called **Nesting of methods**

Consider the following example to understand nesting of methods

```

import java.util.Scanner;
public class Nesting_methods
{
    Int perimeter(int l, int b)
    {
        int pr = 12 * ( l + b);
        return pr;
    }
    int area(int l, int b)
    {
        int pr = perimeter( l , b);
        System.out.println("perimeter:" + pr);
        int ar = 6 * l * b;
        return ar;
    }
    int volume( int l, int b, int h)
    {
        int ar= area(l, b);
        System.out.println("area=" + ar);
        int vol;
        vol = l * b * h;
        return vol;
    }
    Public static void main(String[ ] args)
    {

```

```

        Scanner s= new Scanner(System.in);
        System.out.println("enter length of cuboid:");
int l = s.nextInt();
        System.out.println("enter breadth of cuboid:");
int b = s.nextInt();
        System.out.println("enter height of cuboid:");
int h = s.nextInt();
        Nesting_methods obj = new Nesting_methods( );
int vol = obj.volume(l, b, h);
        System.out.println("Volume=" + vol);
    }
}

```

3.2.10. Inheritance

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

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 the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.

Why use inheritance in java

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

Terms used in Inheritance

- **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

- **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

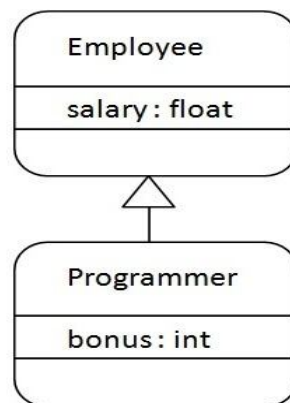
The syntax of Java Inheritance

```
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. The meaning of "extends" is to increase the functionality.

In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.



As displayed in the above figure, Programmer is the subclass and Employee is the superclass. The relationship between the 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);
}
}
```

Output

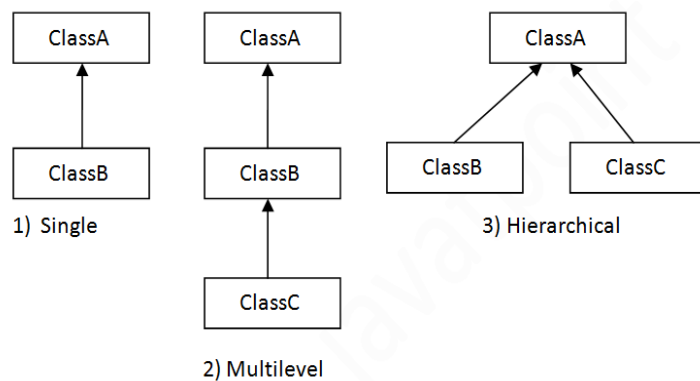
Programmer salary is : 40000.0

Bonus of programmer is : 10000.0

In the above example, Programmer object can access the field of own class as well as of Employee class i.e. code reusability.

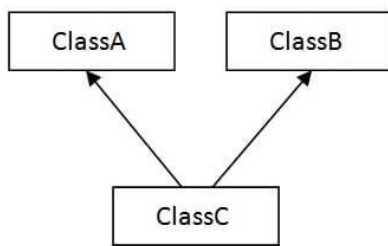
Types of inheritance in java

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.

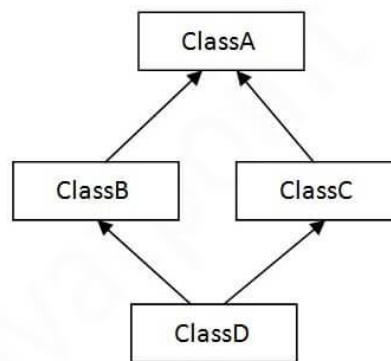


Note: Multiple inheritance is not supported in java through class.

When one class inherits multiple classes, it is known as multiple inheritance. For Example:



4) Multiple



5) Hybrid

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 the same method and you call it from child class object, there will be ambiguity to call the 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.

```

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?
}
}
  
```

3.2.11. Overriding methods

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 a subclass provides the specific

implementation of the method that has been declared by one of its parent class, it is known as method overriding.

Usage of Java Method Overriding

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

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

Consider the following example

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

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

Out put

Vehicle is running

Example of method overriding

In this example, we have defined the run method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the method are the same, and there is IS-A relationship between the classes, so there is method overriding.

```
class Vehicle{
    void run(){System.out.println("Vehicle is running");}
}
class Bike2 extends Vehicle{
    void run(){System.out.println("Bike is running safely");}
```

```

public static void main(String args[]){
    Bike2 obj = new Bike2();
    obj.run();
}
}

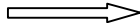
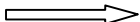

```

Output

Bike is running safely

3.2.12. Final variables, methods and classes

final keyword is used in different contexts. First of all, *final* is a non-access modifier applicable **only to a variable, a method or a class**. Following are different contexts where final is used.

Final variable		to create constant variables
Final methods		prevent method overriding
Final classes		prevent inheritance

- **Final variables**

When a variable is declared with *final* keyword, its value can't be modified, essentially, a constant. This also means that you must initialize a final variable. If the final variable is a reference, this means that the variable cannot be re-bound to reference another object, but internal state of the object pointed by that reference variable can be changed i.e. you can add or remove elements from final array or final collection. It is good practice to represent final variables in all uppercase, using underscore to separate words

```

// a final variable
final int THRESHOLD = 5;
// a blank final variable
final int THRESHOLD;
// a final static variable PI
static final double PI = 3.141459265;
// a blank final static variable
Static final double PI;

```


Initializing a final variable

We must initialize a final variable, otherwise compiler will throw compile-time error. A final variable can only be initialized once, either via an initializer or an assignment statement. There are three ways to initialize a final variable:

1. You can initialize a final variable when it is declared. This approach is the most common. A final variable is called **blank final variable**, if it is **not** initialized while declaration. Below are the two ways to initialize a blank final variable.
2. A blank final variable can be initialized inside instance-initializer block or inside constructor. If you have more than one constructor in your class then it must be initialized in all of them, otherwise compile time error will be thrown.
3. A blank final static variable can be initialized inside static block.

Consider the following example for working with final variable

```
//Java program to demonstrate different
// ways of initializing a final variable

class Gfg
{
    // a final variable
    // direct initialize
    final int THRESHOLD = 5;

    // a blank final variable
    final int CAPACITY;

    // another blank final variable
    final int MINIMUM;

    // a final static variable PI
    // direct initialize
    static final double PI = 3.141592653589793;

    // a blank final static variable
    static final double EULERCONSTANT;

    // instance initializer block for
    // initializing CAPACITY
    {
        CAPACITY = 25;
    }
}
```

```

// static initializer block for
// initializing EULERCONSTANT
static {
    EULERCONSTANT = 2.3;
}

// constructor for initializing MINIMUM
// Note that if there are more than one
// constructor, you must initialize MINIMUM
// in them also
public GFG()
{
    MINIMUM = -1;
}
}

```

When to use a final variable:

The only difference between a normal variable and a final variable is that we can re-assign value to a normal variable but we cannot change the value of a final variable once assigned. Hence final variables must be used only for the values that we want to remain constant throughout the execution of program.

Reference final variable: When a final variable is a reference to an object, then this final variable is called reference final variable. For example, a final StringBuffer variable looks like

```
final StringBuffer sb;
```

As you know that a final variable cannot be re-assign. But in case of a reference final variable, internal state of the object pointed by that reference variable can be changed. Note that this is not re-assigning. This property of *final* is called *non-transitivity*.

```

// Java program to demonstrate
// reference final variable

class Gfg
{
    public static void main(String[] args)
    {
        // a final reference variable sb
        final StringBuilder sb = new StringBuilder("Geeks");
    }
}

```

```

System.out.println(sb);

// changing internal state of object
// reference by final reference variable sb
sb.append("ForGeeks");

System.out.println(sb);
}
}

```

Output:

Geeks

GeeksForGeeks

The *non-transitivity* property also applies to arrays, because arrays are objects in java. Arrays with final keyword are also called final arrays.

- **Final classes**

When a class is declared with *final* keyword, it is called a final class. A final class cannot be extended(inherited). There are two uses of a final class :

1. One is definitely to prevent inheritance, as final classes cannot be extended. For example, all Wrapper Classes like Integer,Float etc. are final classes. We cannot extend them.
2. The other use of final with classes is to create an immutable class like the predefined String class. you cannot make a class immutable without making it final.

- **Final methods**

When a method is declared with *final* keyword, it is called a final method. A final method cannot be overridden. The Object class does this—a number of its methods are final. We must declare methods with final keyword for which we required to follow the same implementation throughout all the derived classes. The following fragment illustrates final keyword with a method:

```

class A
{
    final void m1()
    {
        System.out.println("this is a final method");
    }
}

```

```

    }
}
class B
{
    void m1()
    {
        //compile-error! Can't override.
        System.out.println("illegal!");
    }
}

```

3.2.13. Abstract methods and classes, Visibility control

An *abstract class* is a class that is declared abstract—it may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be subclassed. An *abstract method* is a method that is declared without an implementation (without braces, and followed by a semicolon), like this:

```
abstract void moveTo(double deltaX, double deltaY);
```

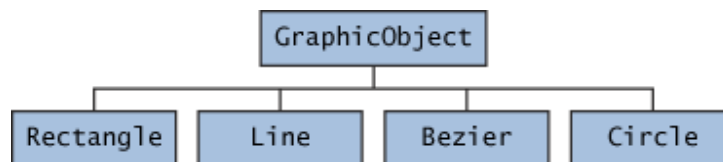
If a class includes abstract methods, then the class itself *must* be declared abstract, as in:

```
public abstract class GraphicObject {
    // declare fields
    // declare nonabstract methods
    abstract void draw();
}

```

When an abstract class is subclassed, the subclass usually provides implementations for all of the abstract methods in its parent class. However, if it does not, then the subclass must also be declared abstract. Methods in an *interface* that are not declared as default or static are *implicitly* abstract, so the `abstract` modifier is not used with interface methods. (It can be used, but it is unnecessary.)

In an object-oriented drawing application, you can draw circles, rectangles, lines, Bezier curves, and many other graphic objects. These objects all have certain states (for example: position, orientation, line color, fill color) and behaviors (for example: moveTo, rotate, resize, draw) in common. Some of these states and behaviors are the same for all graphic objects (for example: position, fill color, and moveTo). Others require different implementations (for example, resize or draw). All GraphicObjects must be able to draw or resize themselves; they just differ in how they do it. This is a perfect situation for an abstract superclass. You can take advantage of the similarities and declare all the graphic objects to inherit from the same abstract parent object (for example, GraphicObject) as shown in the following figure.



Classes Rectangle, Line, Bezier, and Circle Inherit from GraphicObject

First, you declare an abstract class, GraphicObject, to provide member variables and methods that are wholly shared by all subclasses, such as the current position and the moveTo method. GraphicObject also declares abstract methods for methods, such as draw or resize, that need to be implemented by all subclasses but must be implemented in different ways. The GraphicObject class can look something like this:

```
abstract class GraphicObject {
    int x, y;
    ...
    void moveTo(int newX, int newY) {
        ...
    }
    abstract void draw();
    abstract void resize();
}
```

Each nonabstract subclass of GraphicObject, such as Circle and Rectangle, must provide implementations for the draw and resize methods:

```
class Circle extends GraphicObject {
    void draw() {
        ...
    }
    void resize() {
        ...
    }
}
class Rectangle extends GraphicObject {
    void draw() {
        ...
    }
    void resize() {
        ...
    }
}
```

Visibility control

The visibility modifiers are also known as access modifiers. Java provides three types of visibility modifiers: public, private and protected.

Public modifier

The members, methods and classes that are declared public can be accessed from anywhere. This modifier doesn't put any restriction on the access.

public access modifier example in java

Lets take the same example that we have seen above but this time the method addTwoNumbers() has public modifier and class Test is able to access this method without even extending the Addition class. This is because public modifier has visibility everywhere.

Addition.java

```
package abcpackage;
public class Addition
{
    public int addTwoNumbers(int a, int b)
    {
        Return a + b;
    }
}
```

Test.java

```
package xyzpackage;
import abcpackage;
class Test
{
    public static void main(String args[])
    {
        Addition obj = new Addition();
        System.out.println(obj.addTwoNumbers(100, 1);
    }
}
```

Output

101

Private access modifier

The scope of private modifier is limited to the class only.

1. Private Data members and methods are only accessible within the class
2. Class and Interface cannot be declared as private
3. If a class has private constructor then you cannot create the object of that class from outside of the class.

Private access modifier example in java

This example throws compilation error because we are trying to access the private data member and method of class ABC in the class Example. The private data member and method are only accessible within the class.

```
class ABC
{
private double num = 100;
private int square(int a){
return a * a;
}
}

public class Example
{
public static void main(String args[])
{
    ABC obj = new ABC();
    System.out.println(obj.num);
    System.out.println(obj.square(10));
}
}
```

Out put

Compile – time error

Protected access modifier

Protected data member and method are only accessible by the classes of the same package and the subclasses present in any package. You can also say that the protected access modifier is similar to default access modifier with one exception that it has visibility in sub classes.

Classes cannot be declared protected. This access modifier is generally used in a parent child relationship.

Protected access modifier example in Java

In this example the class Test which is present in another package is able to call the addTwoNumbers() method, which is declared protected. This is because the Test class extends class Addition and the protected modifier allows the access of protected members in subclasses (in any packages).

Addition.java

```
package abcpackage;
public class Addition
{
    protected int adTwoNumbers(int a, int b)
    {
        return a + b;
    }
}
```

Test.java

```
package xyzpackage;
import abcpackage;
class Test Extends Addition
{
    public static void main(String args[])
    {
        Addition obj = new Addition();
        Test obj = new Test();
        System.out.println(obj.addTwoNumbers(100, 211);
    }
}
```

Out put

311

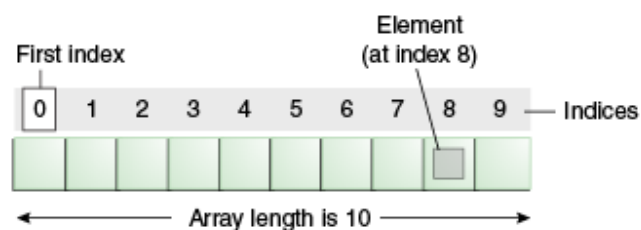
The scope of access modifiers in tabular form

	Class	Package	Subclass(same package)	Subclass(different package)	Outside class
Public	yes	yes	yes	yes	Yes
Protected	Yes	yes	yes	yes	No
Default	yes	yes	yes	Yes	No
Private	yes	No	No	No	No

3.3. Arrays

An array is a collection of similar type of elements which have a contiguous memory location. Java array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array. Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on

Unlike C/C++, we can get the length of the array using the length member. In C/C++, we need to use the sizeof operator. In Java, array is an object of a dynamically generated class. Java array inherits the Object class, and implements the Serializable as well as Cloneable interfaces. We can store primitive values or objects in an array in Java. Like C/C++, we can also create single dimensional or multidimensional arrays in Java. Moreover, Java provides the feature of anonymous arrays which is not available in C/C++.



Advantages

- **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
- **Random access:** We can get any data located at an index position.

Disadvantages

- **Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.

Types of arrays in Java

There are two type of arrays in Java

- Single Dimensional Array
- Multidimensional Array

3.3.1. One Dimensional Arrays

In regular terms, it is the length of something. Similarly, as far as an **array** is concerned, **one dimension** means it has only **one** value per location or index. **One-dimensional array in Java** programming is an **array** with a bunch of values having been declared with a **single** index.

Syntax to declare an array

```
Datatype[ ] arrayname;
```

Creation/Instantiation of an Array

```
Arrayname = new datatype[size];
```

Example

```
int num[ ] = {2,2,5,2,5,6};
```

1. Consider the below example for declare, instantiate, initialize of array

```
class Testarray{  
public static void main(String args[]){  
  
int a[]=new int[5];//declaration and instantiation  
a[0]=10;//initialization  
a[1]=20;  
a[2]=70;  
a[3]=40;  
a[4]=50;  
  
//printing array  
for(int i=0;i<a.length;i++)//length is the property of array  
System.out.println(a[i]);  
  
}}
```

Out put

10

20

70

40

50

2. Java program to illustrate the use of declaration instantiation and initialization of java array in a single line

```

class Testarray1 {
public static void main(String args[]){

int a[]={33,3,4,5};//declaration, instantiation and initialization

//printing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);

}
}

```

Out put

33

3

4

5

3.3.2. Two Dimensional Array

The elements of a **2D array** are arranged in rows and columns, and the new operator for **2D arrays** specifies both the number of rows and the number of columns. For **example**, `int[][] A; A = new int[3][4];` This creates a **2D array** of `int` that has 12 elements arranged in 3 rows and 4 columns.

Syntax

```

datatype[ ] [ ] arrayname;
arrayname = new datatype[size][size];

```

Example

```

int [ ] [ ] a;
a= new int[3] [4];

```

consider example program for two dimensional array

```

class Testarray3 {
public static void main(String args[]){

//declaring and initializing 2D array

```

```
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
```

```
//printing 2D array
for(int i=0;i<3;i++){
  for(int j=0;j<3;j++){
    System.out.print(arr[i][j]+" ");
  }
  System.out.println();
}
}}
```

Out put

```
1 2 3
2 4 5
```

```
4 4 5
```

3.4. Strings

String manipulation is the most common part of many Java programs. Strings represent sequence of characters. The easiest way to represent a sequence of characters in Java is by using a character array.

Example

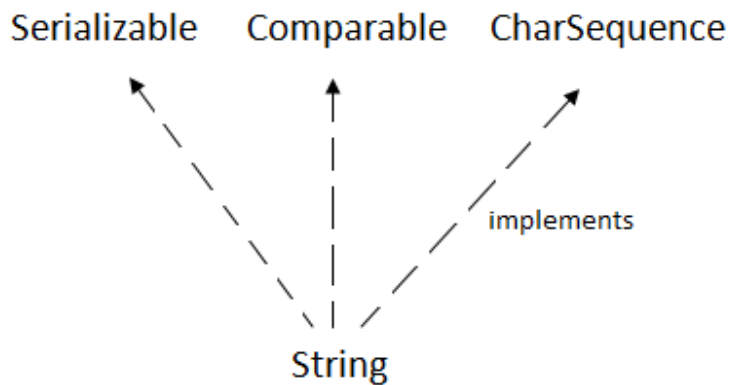
```
Char [ ] ch = {'j','a','v','a'};
String s = new String(ch);
```

Is same as

```
String s="java";
```

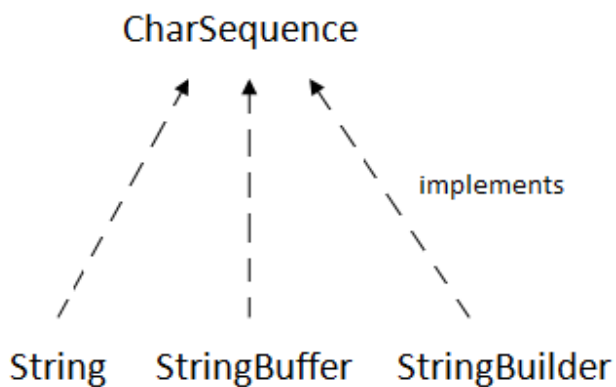
Java String class provides a lot of methods to perform operations on string such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

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



CharSequence Interface

The `CharSequence` interface is used to represent the sequence of characters. `String`, `StringBuffer` and `StringBuilder` classes implement it. It means, we can create strings in java by using these three classes



The Java `String` is immutable which means it cannot be changed. Whenever we change any string, a new instance is created. For mutable strings, you can use `StringBuffer` and `StringBuilder` classes.

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:

```
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 the string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

```
String s1 = "welcome";
```

```
String s2 = "welcome";//it doesn't create a new instance
```

In the above example, only one object will be created. Firstly, JVM will not find any string object with the value "Welcome" in string constant pool, that is why it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create a new object but will return the reference to the same instance.

2. New keyword

```
String s = new String("welcome");// creates two objects and one reference variable
```

In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).

Consider the following example for string in Java

```
public class StringExample{
public static void main(String args[]){
String s1="java";
char ch[]={'s','t','r','i','n','g','s'};
String s2=new String(ch);
String s3=new String("example");
System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
}}
```


Out put

java

strings

examples

Java String class methods

The java.lang.String class provides many useful methods to perform operations on sequence of char values.

No.	Method	Description
1	<u>char charAt(int index)</u>	returns char value for the particular index
2	<u>int length()</u>	returns string length
3	<u>static String format(String format, Object... args)</u>	returns a formatted string.
4	<u>static String format(Locale l, String format, Object... args)</u>	returns formatted string with given locale.
5	<u>String substring(int beginIndex)</u>	returns substring for given begin index.
6	<u>String substring(int beginIndex, int endIndex)</u>	returns substring for given begin index and end index.
7	<u>boolean contains(CharSequence s)</u>	returns true or false after matching the sequence of char value.
8	<u>static String join(CharSequence delimiter, CharSequence... elements)</u>	returns a joined string.
9	<u>static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)</u>	returns a joined string.
10	<u>boolean equals(Object another)</u>	checks the equality of string with the

		given object.
11	<u>boolean isEmpty()</u>	checks if string is empty.
12	<u>String concat(String str)</u>	concatenates the specified string.
13	<u>String replace(char old, char new)</u>	replaces all occurrences of the specified char value.
14	<u>String replace(CharSequence old, CharSequence new)</u>	replaces all occurrences of the specified CharSequence.
15	<u>static String equalsIgnoreCase(String another)</u>	compares another string. It doesn't check case.
16	<u>String[] split(String regex)</u>	returns a split string matching regex.
17	<u>String[] split(String regex, int limit)</u>	returns a split string matching regex and limit.
18	<u>String intern()</u>	returns an interned string.
19	<u>int indexOf(int ch)</u>	returns the specified char value index.
20	<u>int indexOf(int ch, int fromIndex)</u>	returns the specified char value index starting with given index.
21	<u>int indexOf(String substring)</u>	returns the specified substring index.
22	<u>int indexOf(String substring, int fromIndex)</u>	returns the specified substring index starting with given index.
23	<u>String toLowerCase()</u>	returns a string in lowercase.
24	<u>String toLowerCase(Locale l)</u>	returns a string in lowercase using specified locale.
25	<u>String toUpperCase()</u>	returns a string in uppercase.
26	<u>String toUpperCase(Locale l)</u>	returns a string in uppercase using specified locale.

27	<u>String trim()</u>	removes beginning and ending spaces of this string.
28	<u>static String valueOf(int value)</u>	converts given type into string. It is an overloaded method.

3.5. Vectors

Java Vector class comes under the java.util package. The vector class implements a growable array of objects. Like an array, it contains the component that can be accessed using an integer index.

Vector is very useful if we don't know the size of an array in advance or we need one that can change the size over the lifetime of a program.

Vector implements a dynamic array that means it can grow or shrink as required. It is similar to the ArrayList, but with two differences-

- Vector is synchronized.
- The vector contains many legacy methods that are not the part of a collections framework

3.6. Wrapper classes

The **wrapper class in Java** provides the mechanism *to convert primitive into object and object into primitive*.

Since J2SE 5.0, **autoboxing** and **unboxing** feature convert primitives into objects and objects into primitives automatically. The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.

Use of Wrapper classes in Java

Java is an object-oriented programming language, so we need to deal with objects many times like in Collections, Serialization, Synchronization, etc. Let us see the different scenarios, where we need to use the wrapper classes.

- **Change the value in Method:** Java supports only call by value. So, if we pass a primitive value, it will not change the original value. But, if we convert the primitive value in an object, it will change the original value.
- **Serialization:** We need to convert the objects into streams to perform the serialization. If we have a primitive value, we can convert it in objects through the wrapper classes.
- **Synchronization:** Java synchronization works with objects in Multithreading.
- **java.util package:** The java.util package provides the utility classes to deal with objects.
- **Collection Framework:** Java collection framework works with objects only. All classes of the collection framework (ArrayList, LinkedList, Vector, HashSet, LinkedHashSet, TreeSet, PriorityQueue, ArrayDeque, etc.) deal with objects only.

The eight classes of the *java.lang* package are known as wrapper classes in Java. The list of eight wrapper classes are given below:

Primitive Type	Wrapper class
Boolean	<u>Boolean</u>
Char	<u>Character</u>
Byte	<u>Byte</u>
Short	<u>Short</u>
Int	<u>Integer</u>
Long	<u>Long</u>
Float	<u>Float</u>
Double	<u>Double</u>

3.7. Enumerated types

The **Enum in Java** is a data type which contains a fixed set of constants.

It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, and SATURDAY) , directions (NORTH, SOUTH, EAST, and WEST),

season (SPRING, SUMMER, WINTER, and AUTUMN or FALL), colors (RED, YELLOW, BLUE, GREEN, WHITE, and BLACK) etc. According to the Java naming conventions, we should have all constants in capital letters. So, we have enum constants in capital letters.

Java Enums can be thought of as classes which have a fixed set of constants (a variable that does not change). The Java enum constants are static and final implicitly. It is available since JDK 1.5.

Enums are used to create our own data type like classes. The **enum** data type (also known as Enumerated Data Type) is used to define an enum in Java. Unlike C/C++, enum in Java is more *powerful*. Here, we can define an enum either inside the class or outside the class.

Java Enum internally inherits the *Enum class*, so it cannot inherit any other class, but it can implement many interfaces. We can have fields, constructors, methods, and main methods in Java enum.

Points to remember for Java Enum

- Enum improves type safety
- Enum can be easily used in switch
- Enum can be traversed
- Enum can have fields, constructors and methods
- Enum may implement many interfaces but cannot extend any class because it internally extends Enum class

```
class EnumExample1 {  
  
    public enum Season { WINTER, SPRING, SUMMER, FALL }  
  
    public static void main(String[] args) {  
        for (Season s : Season.values())  
            System.out.println(s);  
  
    }  
}
```

Out put

WINTER

SPRING

SUMMER

FALL

3.8. Summary

1. Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors – wagging the tail, barking, eating. An object is an instance of a class.
2. A class can be defined as a template/blueprint that describes the behavior/state that the objects of its type support.
3. Method overloading: Method Overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different.
4. Method overriding: Method Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes.
5. Inheritance is the process by which objects of one class acquire the properties of another class. It provides code reusability.
6. Final keyword is used in different contexts. First of all, final is a non-access modifier applicable only to a variable, a method or a class.
7. An array is a collection of similar type of elements which have a contiguous memory location.
8. String manipulation is the most common part of many Java programs. Strings represent sequence of characters. The easiest way to represent a sequence of characters in Java is by using a character array.
9. The Enum in Java is a data type which contains a fixed set of constants.
10. The *java.lang* package are known as wrapper classes in Java.

3.9. SAQ

1. Distinguish between a class and an object?
2. Explain the terms i) static ii) final
3. Name and explain any three methods that the object class defines?
4. Explain string class in Java? Give suitable examples for creating objects of string class?
5. Explain arrays in Java by giving suitable examples?
6. List the difference between run time polymorphism and dynamic polymorphism.
7. Explain polymorphism in detail?

Unit – IV Interfaces

- 4.0. Structure
 - 4.1. Objective
 - 4.2. Interfaces: Multiple Inheritances
 - 4.3. Extending Interfaces
 - 4.4. Implementing Interfaces
 - 4.5. Accessing Interface variables
 - 4.6. Summary
 - 4.7. SAQ
-

4.1. Objective

In this unit we discuss about interfaces. How interfaces can be implemented for multiple inheritances. How to use extends and implements key words.

4.2. Interfaces: Multiple Inheritances

In the previous units we come across classes and how they can be inherited by other classes. Even about various form of inheritance and pointed out that java does not support multiple inheritance. That is, classes in Java cannot have more than one superclass. For instance, a definition given below is not permitted in Java.

```
Class A extends B extends C
{
    -----
    -----
}
```

However, the designer of Java could not overlook the importance of multiple inheritance. A large number of real-life applications require the use of multiple inheritance whereby we inherit methods and properties from several, distinct classes. Since C++ like implementation of multiple inheritance proves difficult and adds complexity to the language, Java provides an

alternate approach know as Interfaces to support the concept of multiple inheritance. Although a Java class cannot be a subclass of more than one superclass, it can implement more than one interface, thereby enabling us to create classes that build upon other classes without the problems created by multiple inheritance.

Declaring interfaces

An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

Syntax

```
Interface<interface_name>
{
    //declare constant fields
    // declare methods that abstract
    // by default
}
```

Interface is the key word and Inteface_name is any valid java variable. Variable are declared as following:

```
Static final type variablename = value;
```

Note that all variables are declared as constants. Methods declaration will contain only a list of methods without any body statements. Consider the following example

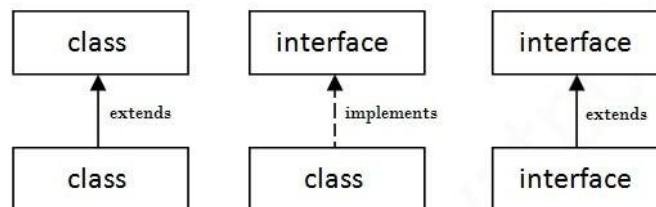
```
return-type methodname1 ( parameter_list);
```

Consider the following example with two variables and one method

```
interface Item
{
    static final int code = 1001;
    static final String name = "fan";
    void display( );
}
```

The relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface, but a **class implements an interface**.



4.3. Extends Interface

When one interface inherits from another interface, that sub-interface inherits all the methods and constants that its super interface declared. In addition, it can also declare new abstract methods and constants. To extend an interface, you use the extends keyword just as you do in the class definition. Unlike a subclass which can directly extend only one subclass, an interface can directly extend multiple interfaces.

syntax

```
[public] interface InterfaceName extends interface1[, interface2, , interfaceN]
{ //interface body }
```

Here, the name of the interface is InterfaceName. The extends clause declares that this interface extends one or more interfaces. These are known as *super interfaces* and are listed by name. Each super interface name is separated by commas.

Now let us consider a program that demonstrates how interface extends another interface

```
interface Interface1
{
    public void f1();
}
//Interface2 extending Interface1
interface Interface2 extends Interface1
{
    public void f2();
}
```

```

}
class x implements Interface2
{
    //definition of method declared in interfacel
    public void f1()
    {
        System.out.println("Contents of Method f1() in Interface1");
    }
    public void f2()
    {
        System.out.println("Contents of Method f2() in Interface2");
    }
    public void f3()
    {
        System.out.println("Contents of Method f3() of Class X");
    }
}
class ExtendingInterface
{
    public static void main(String[] args)
    {
        Interface2 v2; //Reference variable of Interface2
        v2 = new x(); //assign object of class x
        v2.f1();
        v2.f2();
        x x1=new x();
        x1.f3();
    }
}

```

Out put

contents of Method f1() in interface1

contents of Method f2() in interface2

contents of Method f3() of Class x

4.4. Implementing Interfaces

A class declares all of the interfaces that it implements in its class declaration. To declare that your class implements one or more interfaces, use the keyword `implements` followed by a comma-delimited list of the interfaces implemented by your class.

For example, consider the `Collection` interface introduced on the [previous](#) page. Now, suppose that you were writing a class that implemented a FIFO (first in, first out) queue. Because a FIFO

queue object contains other objects it makes sense for the class to implement the Collection interface. The FIFOQueue class would declare that it implements the Collection interface like this:

```
class FIFOQueue implements Collection {
    ...
    void add(Object obj) {
        ...
    }
    void delete(Object obj) {
        ...
    }
    Object find(Object obj) {
        ...
    }
    int currentCount() {
        ...
    }
}
```

By declaring that it implements the Collection interface, the FIFOQueue class guarantees that it provides implementations for the add, delete, find, and currentCount methods. By convention, the implements clause follows the extends clause if it exists.

4.5. Accessing Interface variables

An interface does not have instance variables. The members of an interface are always declared as static and final, that the variable cannot be modified by the methods in the class. Such variables will be inherited by the class that implements the interface.

Consider the following example for accessing interface variables

```
interface Data
{
    int data1 = 50;
    int data2 = 100;
}
class ShowData implements Data
{
    void interface Values()
    {
        System.out.println("data1=" + data1);
        System.out.println("data1=" + data1);
    }
    void modifyInterfaceValues()
    {
```

```

data1+=20;
data2+=40;
}
}
public class Javaapp
{
    public static void main(String[ ] args)
    {
        System.out.println("data1=" + Data.data1);
        System.out.println("data2=" + Data.data2);
        ShowData obj = new ShowData();
        obj.interfaceValues();
        obj.modifyInterfaceValues();
        obj.interfaceValues();
    }
}

```

4.6. Summary

1. A class that implements interface must implements all the methods in interface. All the methods are public and abstract. And all the fields are public, static, and final. It is used to achieve multiple inheritance.
2. A class implements an interface, thereby inheriting the abstract methods of the interface. ... Interfaces are more flexible, because a class can implement multiple interfaces. Since Java does not have multiple inheritance, using abstract classes prevents your users from using any other class hierarchy.
3. An interface is a Java programming language construct, similar to an abstract class, that allows you to specify zero or more method signatures without providing the implementation of those methods.

4.7. SAQ

1. How interfaces are appropriate in Java?
2. Explain interfaces in Java with suitable example?
3. Write a Java program that implements interfaces?
4. What is multiple inheritances and how Java supports multiple inheritance?

Unit – V Packages

5.0. Structure

5.1. Objective

5.2. Packages

5.2.1 Java API packages

5.3. Using system packages

5.4. Naming conventions

5.5. Creating packages

5.6. Accessing packages

5.7. Using a Package

5.8. Summary

5.9. SAQ

5.1. Objective

The main objective of this unit is to learn what are packages? How to create and use packages in the program without physically copying them into the program under development.

5.2. Packages

The main features of OOPs is its ability to reuse the code already created. One way of achieving this is by extending the classes and implementing the interfaces that already studied in the previous units. It is limited to reuse the classes within a program. Suppose if we need to use classes from other programs without physically copying them into the program under development, this can be accomplished in Java by using the concept of **PACKAGES**. Packages are similar to the of class libraries in other languages. The concept of reusability in Java is implemented by using packages. The following are the benefits of package

1. The classes contained in the packages of other programs can be easily reused.
2. In packages, classes can be unique compared with classes in other packages. That is, two classes in two different packages can have the same name. they may be

referred by their fully qualified name, comprising the package name and the class name.

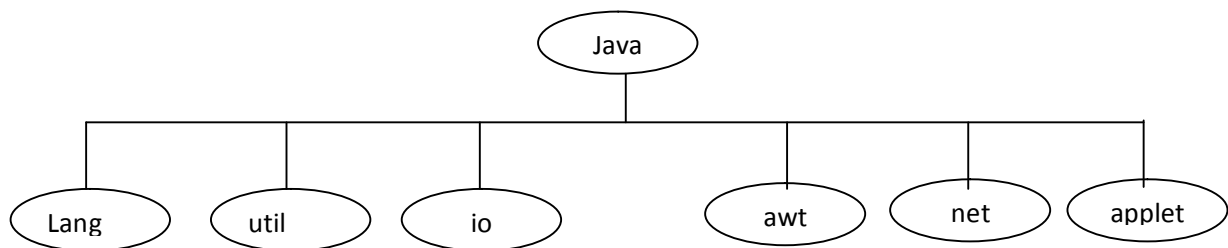
3. Packages provide a way to hide classes thus preventing other programs or packages from accessing classes that are ment for internal use only.
4. Packages also provide a way for separating design from coding. First we can design classes and decide their relationships, and then we can implement the Java code needed for the methods. It is possible to change the implementation of any method without affecting the rest of the design.

Most of the applications need to use different sets of classes, one for the internal representation of out program's data and other for external presentation purposes. We may have to build our own classes for handling our data and use existing class libraries for designing user interfaces. Java packages are therefore classified into two types

1. Java API packages
2. User defined packages

5.2.1. Java API packages

Java API provides a large number of classes grouped into different packages according to functionality. Most of the time we use the packages available with the Java API which can be see in the below figure.



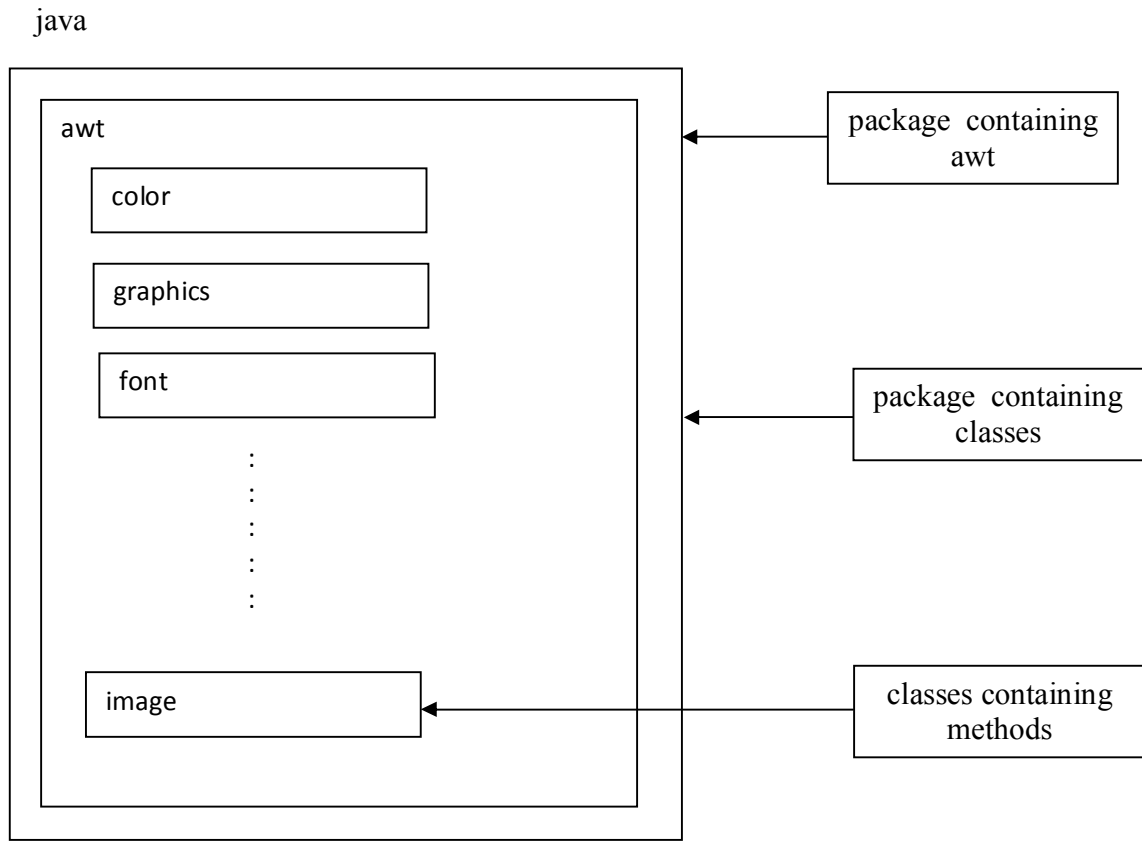
Frequently used API packages

PACKAGE NAME	EXAMPLE CLASSES	FUNCTIONALITY (PURPOSE)
java.lang	System, String, Object, Thread, Exception etc.	These classes are indispensable for every Java program. For this reason, even if this package is not imported, JVM automatically imports.
java.util	These are called as utility (service) classes and are used very frequently in coding.	
java.io	FileInputStream, FileOutputStream, FileReader, FileWriter, RandomAccessFile, BufferedReader, BufferedWriter etc.	These classes are used in all I/O operations including keyboard input.
java.net	URL, ServerSocket, Socket, DatagramPacket, DatagramSocket etc.	Useful for writing socket programming (LAN communication).
java.applet	AppletContext, Applet, AudioStub, AudioClip etc	Required for developing applets that participate on client-side in Internet (Web) programming.
java.awt	Button, Choice, TextField, Frame, List, Checkbox etc.	Essential for developing GUI applications.
java.awt.event	MouseListener, ActionListener,(ActionEvent, WindowAdapter etc.	Without these classes, it is impossible to handle events generated by GUI components
java.sql	DriverManager, Statement, Connection, ResultSet etc	Required for database access in JDBC applications.

Table: Predefined Packages Java Java API

5.3. Using System Packages

The packages are organized in a hierarchical structure as given below. This shows that the package named 'java' contains the package 'awt', which in turn contains various classes required for implementing graphical user interface.



Hierarchical representation of java.awt package

There are two ways of accessing the classes stored in a package. The first approach is to use the fully qualified class name of the class that we want to use. This is done by using the package name containing the class and then appending the class name to it using the dot operator. For example, if we want to refer to the 'color' in the awt package, then we may do so as follows:

java.awt.Color

Notice that awt is as package within the package java and the hierarchy is represented by separating the levels with dots. This approach is perhaps the best and easiest one if we need to access the class only once or when we need not have to access any other classes of the package. But in many situations, we might want to use a class in a number of places in the program or we may like to use many of the classes contained in a package. We may achieve this easily as follows:

import packagename.classname;

or

```
import packagename;
```

These are known as import statements and must appear at the top of the file, before any class declarations, import is a keyword. The first statement allows the specified class in the specified package to be imported. For example

```
import java.awt.Color;
```

Import the class Color and therefore the class name can now be directly used in the program. There is no need to use the package name to qualify the class. the second statement imports every class contained in to specified package. For example, the statement

```
import java.awt.*;
```

5.4. Naming conventions

Packages can be named using the standard java naming rules. By convention, however, packages begin with lowercase letters. This makes it easy for users to distinguish package names from class names when looking at an explicit reference to a class. we know that all class names, again by convention, begin with an uppercase letter. For example, look at the following statement:

```
Double y = java. Lang. Math. sqrt(x);
```

Package name Class name Method name

This statement uses a fully qualified class name Math to invoke the method sqrt(). Note that methods begin with lowercase letters. Consider another example

```
java. Awt.Point pts [ ];
```

this statement declares an array of 'Point' type objects using the fully qualified class name. every package name must be unique to make the best use of packages. Duplicate name will cause run-time errors. Since multiple users work on Internet, duplicate package names are unavoidable.

Java designers have recognized this problem and therefore suggested a package naming convention that ensures uniqueness. This suggests the use of domain names as prefix to the preferred package names. For example

cbe.psg.mypackage

Here cbe denotes city name and psg denotes organization name. Remember that we can create a hierarchy of packages with in packages by separating by levels with dots.

5.5. Creating packages

Let us see now how to create our own packages. We must first declare the name of the package using the package keyword followed by name. This must be first statement in java source file (except for comments and white spaces). Then we define a class, just as we normally define a class. Consider the following example

```
package firstPackage;      // package declaration
public class FirstClass    // class definition
{
    -----
    ----- ( body of class)
    -----
}
```

Here the package name is 'firstPackage'. The class 'FirstClass' is now considered a part of this package. This listing would be saved as a file called FirtsClass.java, and located in a directory named firstPackage. When the source file is compiled, java will create a .class file and store it in the same directory. Remember that the .class file must be located in a directory that has the same name as the package, and this director should be a subdirectory of the directory where classes that will import the packages located.

Let us consider another example of java package. The package keyword is used to create a package in java.

```
//save as simple.java
Package mypack;
Public class simple
```

```

{
  Public static void main(String args[ ])
{
  System.out.println("welcome to package");
}
}

```

For compile java package

If you are not using any IDE, you need to follow the syntax given below

Javac -d directory javafilename

For example

Javac -d.simple.java

The -d switch specifies the destination where to put the generated class file. You can use any directory name like /home (in case of Linux), d:/abc (in case of windows) etc. If you want to keep the package within the same directory, you can use . (dot).

How to run java package program

You need to use fully qualified name e.g. mypack.simple etc to run the class.

To compile: javac -d.simple.java

To run: java mypack.simple

Output: welcome to package

5.6. Accessing a package

In order to access a package we use the key word 'import' this approach is discussed here. The import statement when there are many references to a particular package or the package name is too long and unwieldy. The same approaches can be used to access the user-defined packages as well. The import statement can be used to search a list of packages for a particular class. the general form of import statement for searching a class is as follows:

```
import package1 [.package2] [.package3] . classname;
```

package1 is the name of the top level package, package2 is the name of the package that is inside the package1 and so on. We can have any number of packages in a package hierarchy. Finall, the explicit classname is specified and ended with semicolon(;). Consider the following example

```
import firstPackage.secondPackage.myclass;
```

Here we can also denote a single package or hierarchy of packages as given in the below example also.

```
import packagename.*;
```

The star (*) indicates that the compiler should search this entire package hierarchy when it encounters a class name. This implies that we can access all classes contained in the above package directly.

5.7. Using a package

Let us now consider some simple programs that will use classes from other packages. The below example shows a package name package1 containing a single class 'classA'

```
package package1;
public class classA
{
    Public void displayA()
    {
        System.out.println("class A");
    }
}
```

This source file should be named classA.java and stored in the subdirectory package1 and stated earlier. Now compile this java file. The resultant classA.class will be stored in the same subdirectory. Now consider the listing below:

```
import package1.classA;
class packagetest1
{
    public static void main(String args[ ])
    {
        classA objectA = new classA();
        objectA.displayA();
    }
}
```

This listing shows a simple program that imports the class 'classA' from the package 'package1'. The source file should be saved as `packagetest1.java` and then compiled. The source file and the compiled file would be saved in the directory of which `package1` was subdirectory. Now we can run the program and obtain the results. During the compilation of `packagetest1.java` the compiler checks for the file `classA.class` in the `package1` directory for information it needs, but it does not actually include the code from `classA.class` in the file `packagetest1.class`. When the `packagetest1` program is run, java looks for the file `packagetest1.class`. and loads it using something called class loader. Now the interpreter knows that it also needs the code in the file `classA.class` and loads it as well.

Practicing examples based on packages

```
//PCKG1_ClassOne.java
package pckg1;
public class PCKG1_ClassOne{
int a = 1;
private int pri_a = 2;
protected int pro_a = 3;
public int pub_a = 4;
public PCKG1_ClassOne() {
System.out.println("base class constructor called");
System.out.println("a = " + a);
System.out.println("pri_a = " + pri_a);
System.out.println("pro_a "+ pro_a);
System.out.println("pub_a "+ pub_a);
}
}
```

The above file `PCKG1_ClassOne` belongs to package `pckg1`, and contains data members with all access modifiers.

```
//PCKG1_ClassTwo.java
```

```
package pckg1;
```

```

class PCKG1_ClassTwo extends PCKG1_ClassOne {
PCKG1_ClassTwo() {
System.out.println("derived class constructor called");
System.out.println("a = " + a);
// accessible in same class only
// System.out.println("pri_a = " + pri_a);
System.out.println("pro_a "+ pro_a);
System.out.println("pub_a =" + pub_a);
}
}

```

The above file PCKG1_ClassTwo belongs to package pckg1, and extends PCKG1_ClassOne, which belongs to the same package.

//PCKG1_ClassInSamePackage

```

package pckg1;
class PCKG1_ClassInSamePackage {
PCKG1_ClassInSamePackage() {
PCKG1_ClassOne co = new PCKG1_ClassOne();
System.out.println("same package class constructor called");
System.out.println("a = " + co.a);
// accessible in same class only
// System.out.println("pri_a = " + co.pri_a);
System.out.println("pro_a "+ co.pro_a);
System.out.println("pub_a = " + co.pub_a);
}
}

```

The above file PCKG1_ClassInSamePackage belongs to package pckg1, and having an instance of PCKG1_ClassOne.

```

package PCKG1;
//Demo package PCKG1

```

```

public class DemoPackage1 {
public static void main(String ar[]) {
PCKG1_ClassOne obl = new PCKG1_ClassOne();
PCKG1_ClassTwo ob2 = new PCKG1_ClassTwo();
PCKG1_ClassInSamePackage ob3 = new PCKG1_ClassxnSamePackage();
}
}

```

The above file DemoPackageI belongs to package pckgI, and having an instance of all classes in pckg1.

```

package pckg2;
class PCKG2_ClassOne extends PCKG1.PCKG1_ClassOne {
PCKG2_ClassOne() {
System.out.println("derived class of other package constructor
called");
// accessible in same class or same package only
// System.out.println("a = " + a);
// accessible in same class only
// System.out.println("pri_a = " + pri_a);
System.out.println("pro_a = " + pro_a);
System.out.println("pub_a = " + pub_a);
}
}

```

The above file PCKG2_ClassOne belongs to package pckg2. extends PCKG I_ClassOne, which belongs to PCKG1, and it is trying to access data members of the class PCKGI_ClassOne.

//PCKG2_ClassInOtherPackage

```

package pckg2;
class PCKG2_ClassInOtherPackage {
PCKG2_ClassInOtherpackage() {
PCKG1.PCKG1_ClassOne co = new PCKG1.PCKG1_ClassOne();
}
}

```



```

System.out.println("other package constructor");
// accessible in same class or same package only
// System.out.println("a ,= " + co.a);
// accessible in same class only
// System.out.println("pri_a = " + co.pri_a);
// accessible in same class, subclass of same or other package
// System.out.println("pro_a = " + co.pro_a);
System.out.println("pub_a = " + co.pub_a);
}
}

```

The above file PCKG2_ClassInOtherPackage belongs to package pckg2, and having an instance of PCKG LClassOne of package pckg I, trying to access its some data members.

// Demo package pckg2.

```

package pckg2;
public class DemoPackage2 {
public static void main(String ar[]) {
PCKG2_ClassOne obl = new PCKG2_ClassOne();
PCKG2_ClassInOtherPackage ob2 = new PCKG2_ClassInOtherPackage();
}
}

```

5.8. Summary

1. Java package. A Java package organizes Java classes into namespaces, providing a unique namespace for each type it contains.
2. Classes in the same package can access each other's package-private and protected members.
3. A Package is a collection of related classes.
4. It helps organize your classes into a folder structure and make it easy to locate and use them. More importantly, it helps improve re-usability.

5. Each package in Java has its unique name and organizes its classes and interfaces into a separate namespace, or name group
6. Programmers can define their own packages to bundle a group of classes/interfaces, etc.
7. It is a good practice to group related classes implemented by you so that a programmer can easily determine that the classes, interfaces, enumerations, and annotations are related

5.9. SAQ

1. What are packages? Discuss its access modifiers?
2. How are packages created in Java? Give example?
3. How are sub packages are created and accessed?

Unit – VI

6.0. Structure

6.1. Objective

6.2. Basics of Web Design

6.3. Elements of Web

6.4. Web Site Architecture

6.5. Designing a web site

6.5.1. Web design process and Web site design process

6.5.2. Elements of Web site design

6.9. Web Page and Layout

6.10. Summary

6.11. SAQ

6.1. Objective

The aim of this chapter is how to design a web page.

6.2. Basics of Web Design

Web page design is a process of conceptualization, planning, modeling, and execution of electronic media content delivery via Internet in the form of technologies (such as markup languages) suitable for interpretation and display by a web browser or other web-based graphical user interfaces (GUIs).

The intent of web design is to create a web site (a collection of electronic files residing on one or more web servers) that presents content (including interactive features or interfaces) to the end user in the form of web pages once requested. Such elements as text, forms, and bit-mapped images (GIFs, JPEGs, PNGs) can be placed on the page using HTML, XHTML, or XML tags. Displaying more complex media vector graphics, animations, videos, sounds.

Web design

Beyond visuals and technology considerations the creation and organization of content in Web sites is the most important aspect of Web design. The four primary aspects of the Web design are content, technology, visuals and economics. The primary purpose of content is to inform or perhaps persuade users. The point of using technology on a Web site is to implement the function of the site. The visuals provide the form for the site. Lastly, for most sites we need to consider the economic ramifications of building the site. The amount of influence of each particular aspect of Web sites varies based on the type of site being built. For example, a personal home page generally doesn't have the economic considerations of a shopping site. An intranet for a manufacturing company may not have the visual considerations of a public Web site promoting an action movie, and so on.

The Web Design Pyramid:

It is useful to think of Web sites metaphorically as pyramids, as shown below:

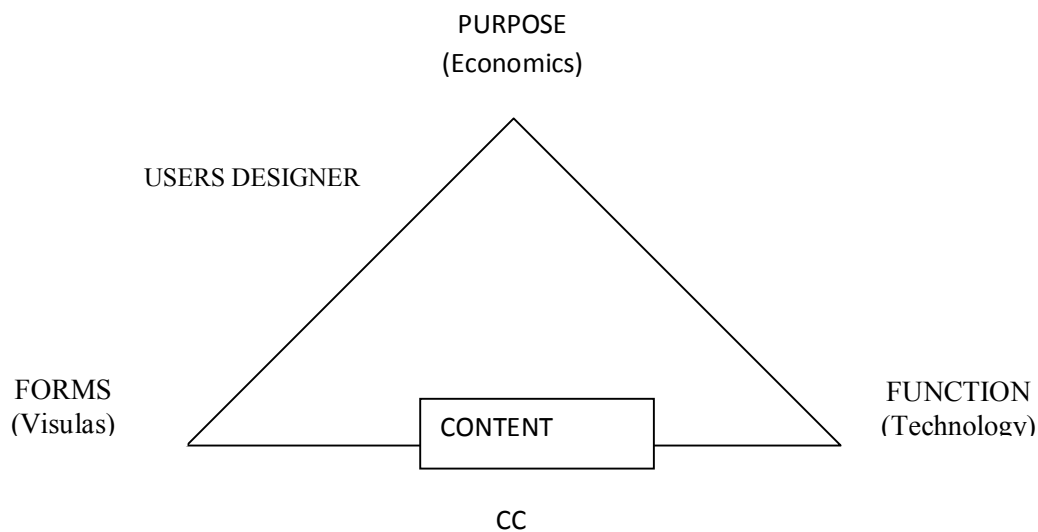


Fig: Web pyramids: the facets of Web design

Content provides the bricks we use to build the pyramid, but the foundation rests solidly on both visuals and technology, with a heavy reliance on economics to make our project worth doing. Even if we are experts able to construct a beautiful and functional Web site, our users may look

at our beautiful construction with puzzlement. Designers, or their employers, often spend more time considering their own needs and wants than those of the site's visitors.

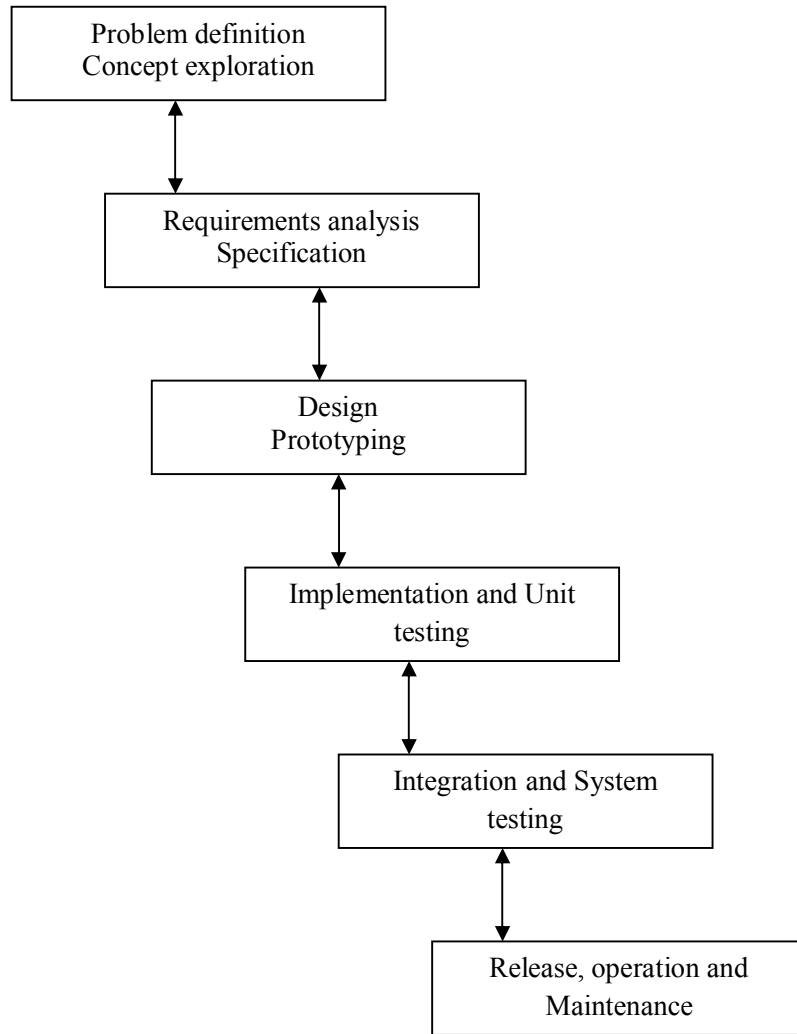
Building Web Sites:

Building a Web site can be very difficult. While some of the core technologies like HTML are easy enough to master, developers seem to make numerous mistakes. The main reasons for this are a lack of developer experience; a poorly defined process, and unrealistic schedules. Because of time constraints or inexperience, designers tend to start from one extreme or another, and then jump right to implementation without considering the preceding steps.

Web Development Process Model:

Technology and visuals provide the base of the Web design pyramid; both are necessary and must relate directly to the purpose of the site. Instead of implementing first and asking questions later, it makes more sense to discuss the purpose of the site and then determine how to accomplish any defined goals. This top-down approach to Web site development is fairly well understood. Many disciplines such as software engineering have defined a process model, the most famous being the waterfall approach, which describes the software life cycle from project planning to eventual release and maintenance. The process is split into a variety of steps that guide the developer from general requirements to specific implementation. An example of the stages in the waterfall process is as shown below:

The Waterfall Model



While building Web sites the following rules must be kept in mind:

1. YOU ARE NOT THE USER
2. USERS ARE NOT DESIGNERS
3. DESIGN FOR THE COMMON USER, BUT ACCOUNT FOR DIFFERENCES
4. A SITE'S EXECUTION MUST BE CLOSE TO FLAWLESS
5. KNOW AND RESPECT THE WEB AND INTERNET MEDIUM CONSTRAINTS
6. WEB SITES SHOULD RESPECT GUI PRINCIPLES WHERE APPROPRIATE
7. NAVIGATION IS ONLY A MEANS TO AN END RESULT
8. VISUALS WILL HEAVILY INFLUENCE THE USER'S INITIAL PERCEPTION OF A SITE'S VALUE

9. THE SITE'S TAKE-AWAY VALUE IS INFLUENCED BY VISUALS, CONTENT, TECHNOLOGY, USABILITY, AND GOAL ACCOMPLISHMENT
10. DO NOT INVENT INTERFACES TO BUILD BRAND
11. THERE IS NO FORM OF CORRECT WEB DESIGN THAT FITS EVERY SITE
12. CONTROL SHOULD BE GIVEN OR AT LEAST APPEAR TO BE GIVEN TO THE USER
13. WHAT YOU SEE IS WHAT YOU WANT (WYSIWYW)

To put in a nutshell, Web design is a multidisciplinary pursuit that consists of four primary components: content, form, function, and purpose. However, agreement as to exactly how these components mix together varies from person to person as well as project to project. While good Web design is hard to define, there is certainly an understanding of what not to do. The field has a great deal to learn from other disciplines, particularly from the intersection of graphical interface design and traditional print design. If the designer keeps the user in mind at all times, many of the most serious design errors can be avoided. Furthermore, designers should respect the restrictions of the medium, as well as any emerging conventions regardless of what users may think they want.

6.3. Elements of Web

Here are the five elements to web design:

1) Content

There's no denying that 'Content is King'. It plays a massive role in Search Engine Optimisation (SEO), and is one of the main reasons people visit your website. You really need to focus a great deal of effort into creating first class content for your website, which should include videos, relevant news/information and high-resolution imagery to make your website 'stickier'. This will ensure you keep your users on your website for longer.

2) Usability

Great usability will never be noticed by the end user, but bad usability instantly stands out. Your website must be easily navigable, intuitive, accessible and mobile-friendly.

The user should know where they are on the website at all times and be able to find where they want to go with little thought. They should also be able to access any page they need without having to view the whole site.

Your site should try to anticipate what your visitors are thinking and help them to fulfill their needs with as little effort as possible.

3) Aesthetics

In this day and age, having a visually impressive website across all devices is crucial. However, you must maintain your brand image. Your website must reflect who you are as a business, and visually connect with the audience.

The visual appeal of your website not only contributes to your brand awareness but also increase your credibility.

4) Visibility

If you had the most aesthetically pleasing and user-friendly website on the web, it would still be unsuccessful unless it could be found. Your presence and visibility through digital marketing campaigns including SEO, social media and email marketing is vital to the success of your website.

It's important that you understand how to be found, what platforms to target and how to utilise your content. Thousands of factors have an impact on where you appear within the search engines, so make sure you have a plan in place!

5) Interaction

Your website must engage with your audience, hold their attention, direct them through the stages of your website and finally encourage them to contact you.

6.4. Web site Architecture and site types

Just as there are many types of software there are many types of Web sites.

General Web Site Types

There are three general categories of Web sites.

1. Public Web sites: A public Web site, an Internet Web site, an external Web site, or simply a Web site is one that is not explicitly restricted to a particular class of users.

2. Intranet Web sites: An intranet Web site is a site that is private to a particular organization, generally run within a private network rather than on the Internet at large.

3. Extranet Web sites: An extranet site is a Web site that is available to a limited class of users, but is available via the public Internet. The design considerations will vary dramatically between the general Web site, as stated in the following table:

Design Consideration	Intranets	Extranets	Public Sites
Information About Users	High	Medium	Low
Capacity Planning	Possible	Usually possible	Difficult to impossible
Bandwidth	High	Varies	Varies greatly
Ability to Set Technology	Yes	Sometimes	Rarely

Interactive Vs Static Sites

Another way to classify sites is if they are interactive or static.

Interactive: An interactive site is one where the users of the site are able to interact directly with the content on the site or with other users of the site.

Static: A static site one where content is relatively fixed in that the user is unable to affect the look or scope of the data they view. In short, the visitor has minimal ability to interact with the site's content other than choosing the order in which to view content.

Dynamic Sites

The more frequently the site changes, the more dynamic it could be thought to be.

Dynamic site: A dynamically generated site is one where the pages of the site are generated at request or view time for the user.

The benefit of a dynamically generated site should be obvious since it presents content the way the user probably wants it.

A site that is targeted to specific users and allows them to determine exactly what they want to see is called a personalized page.

Personalized site: A personalized site is one where content is directly geared towards a particular user, and the user generally can explicitly determine the content, look, or technology contained within a page.

Site Structure

There are two structural aspects to any Web site logical structure and physical structure. A logical structure will describe documents that are related to other documents. The logical structure defines the links between documents. However, the logical location of documents within a site may not relate to the actual physical location of a document. A physical structure describes where a document actually lives. A web site's logical structure is more important to a user than its physical structure. Do not expose physical site file structure, when possible. A site's logical document structure does not have to map to directly match physical structure.

Logical Site Organization Models

There are four main logical organizational forms used in Web sites. They are Linear, grid, hierarchy, and web. Choosing the correct site organization is important in making a site usable.

Linear

A linear form is the most familiar of all site structures because traditional print media tend to follow this style of organization. For example, books are written in linear order so that one page follows another. Presenting information in a linear fashion is often very useful when discussing a step-by-step procedure.

Basic Linear

A pure linear organization facilitates an orderly progression through a body of information, as shown under:

Pure linear:

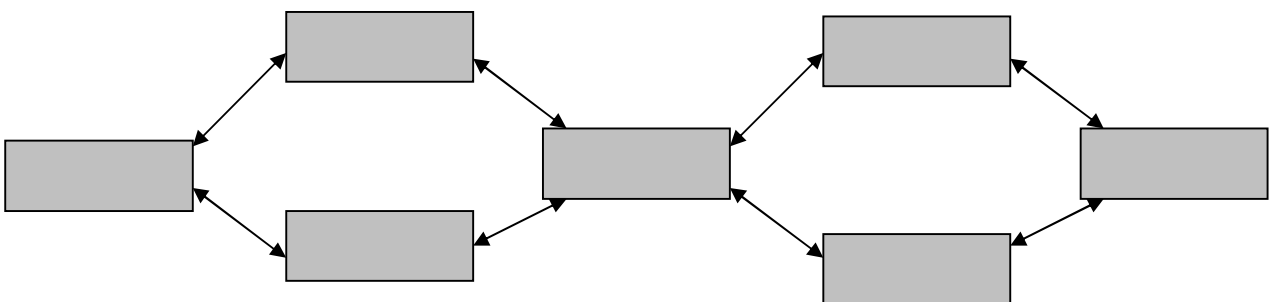


The linear style of organization provides a great deal of predictability in that the designer knows exactly where the user will go next. Because there is really no choice but to move forward or back, a user may find a linear form to be very restrictive.

Linear with alternatives

A linear with alternatives organization simulates interactivity by user back to another page within sequence as illustrated here:

Linear with alternatives

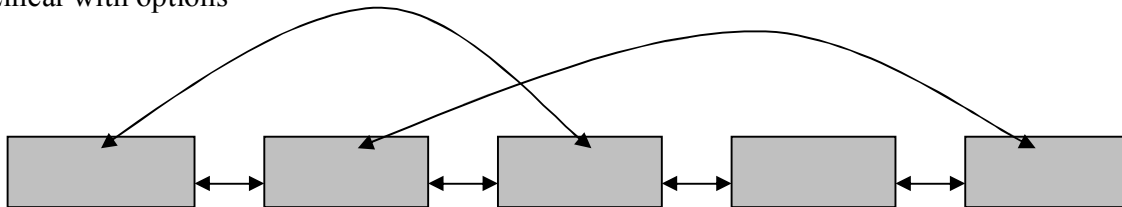


Though the pages are static and there is no dynamic generation of pages, to the user it appears that there is some interactivity. Despite its appearance of choice, the linear with alternatives structure preserves the general linear path through a document collection. Unfortunately, the multiple path possibilities make preloading of pages more difficult with this form of site.

Linear with options

A linear with options structure is good when the general path must be preserved, but slight variations must also be accommodated, such as skipping particular pages. This type of hypertext organization might be useful for an online survey where some users might skip certain inapplicable questions. The basic idea of this site structure is shown below:

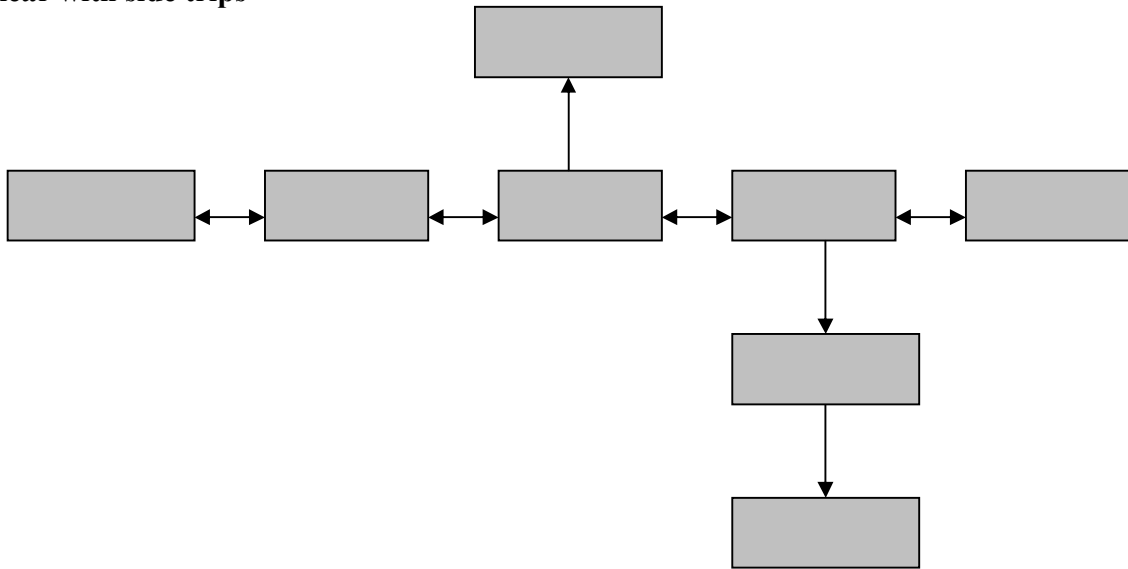
Linear with options



Linear with side trips

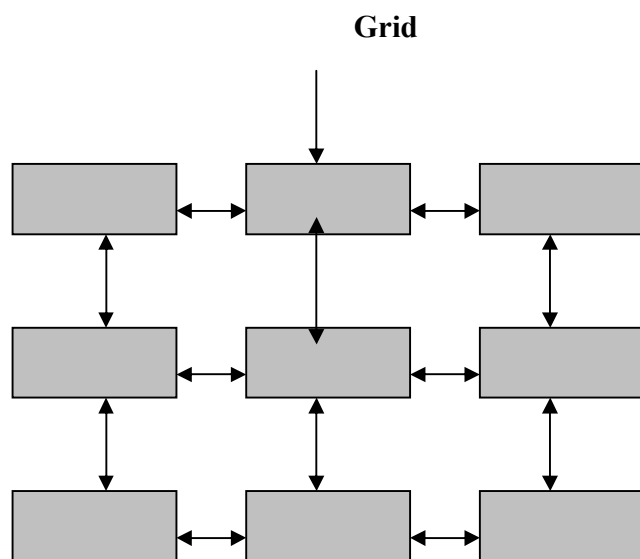
A linear with side trips site organization allows controlled diversions. Although the user might take a short side trip, the structure forces the user back to the main path, preserving the original flow. A side trip to a linear progression is like a sidebar to a magazine article. Rather than distracting the user too much from the main path, this bit of information enhances the experience. However, when many side trips are added into the linear progression, the structure begins to look like the common tree or hierarchy form. The general form of this site structure is shown below:

Linear with side trips



Grid

A grid is a dual linear structure that presents both a horizontal and a vertical relationship between items. Because a grid has a spatial organization, it is good for collections of related items. However, a pure grid structure is uncommon on the Web. When designed properly, a grid provides horizontal and vertical orientation so the user may not feel lost within the site. While a grid structure is highly regular and may be easy for a user to navigate, not many types of information are uniform enough to lend themselves well to this organization style. The grid structure is shown below:



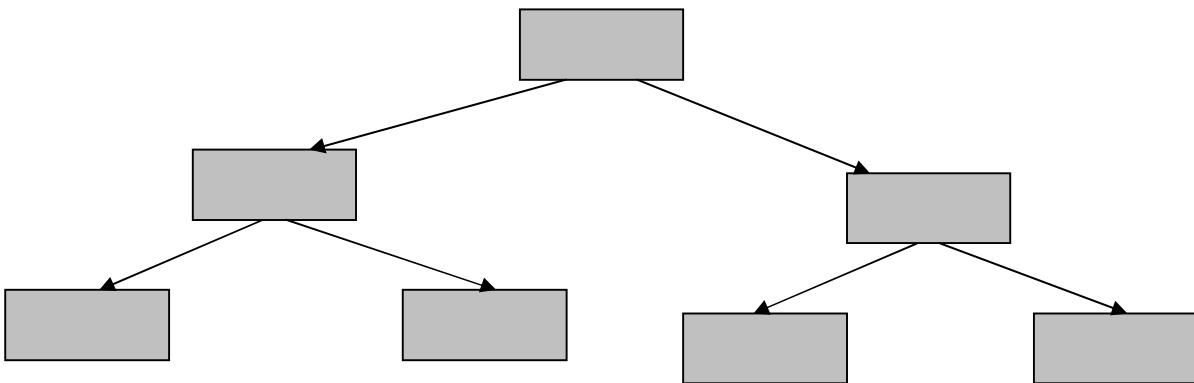
Hierarchy

The most common hypertext structure on the web is the tree or hierarchy form. The hierarchy is very important because it can be modified to hide or expose as much information as is necessary. Hierarchies start with a root page that is often the home page of the site, which serves as a landmark page and often looks much different than other pages in the site. Site landmarks such as home pages are key to successful user navigation. From the home page, various choices are presented. As the user clicks deeper into the site, the choices tend to get more and more specific until eventually a destination, or leaf page, in the tree is reached. Because of this, trees tend to be described by their depth and breadth.

Narrow Trees

A narrow tree presents only a few choices but may require many mouse clicks to get to the final destination. This organization emphasizes depth over breadth.

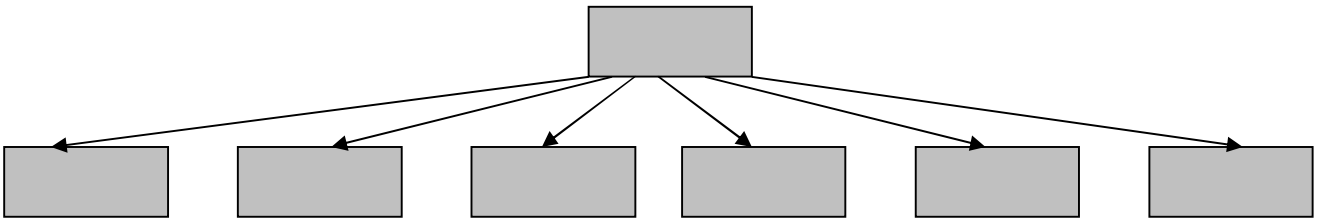
Narrow hierarchy



Wide Trees

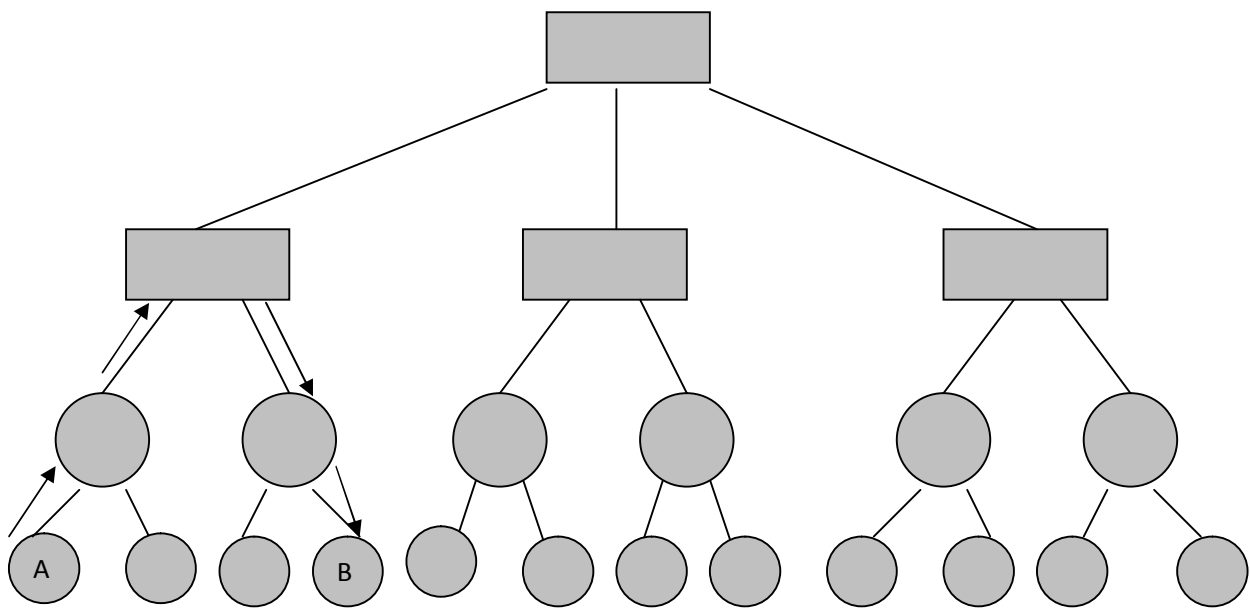
A wide tree or wide hierarchy is based on a breadth of choices. Its main disadvantage is that it may present too many options as pages have numerous choices emanating from them. While the user only has to click once or twice to reach the content, the time spent hunting through all the initial choices may be counterproductive.

Wide hierarchy



Web Trees

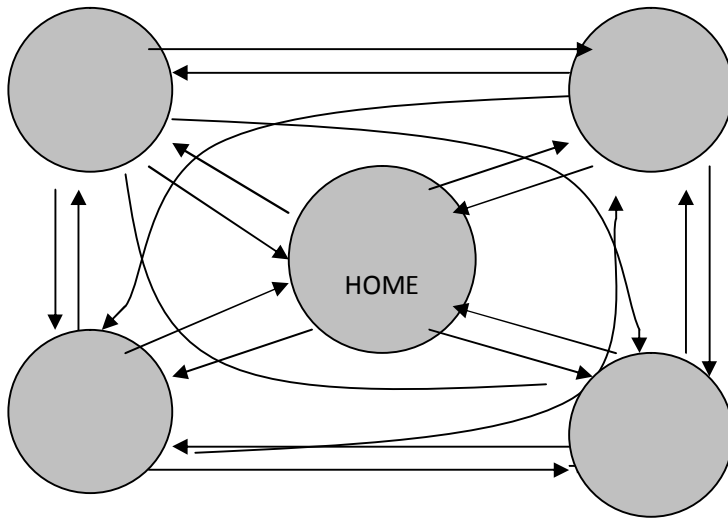
The reality of the Web is that the typical pure-tree structures are rarely used. In a pure tree, there are no cross-links, and backtracking is often required to reach other parts of the tree. Consider if a user is at page A in the structure below; to reach page B, they have to back up two levels and then proceed forward. Trees may require backtrack



While on the Web backtracking is possible using the browser's back button, links are often added to pages so that users who reach a page not through its primary path can navigate the site. In many cases pages are cross-linked using a navigation bar or explicit back-links to help users quickly navigate the site structure. The back and cross links within the site increase the complexity greatly. In this case, consider that only main section pages are cross-linked.

Full Mesh

A site that links every page to every other page could be considered to exhibit a structure called a full mesh. A full mesh for a site with five pages looks as shown below:

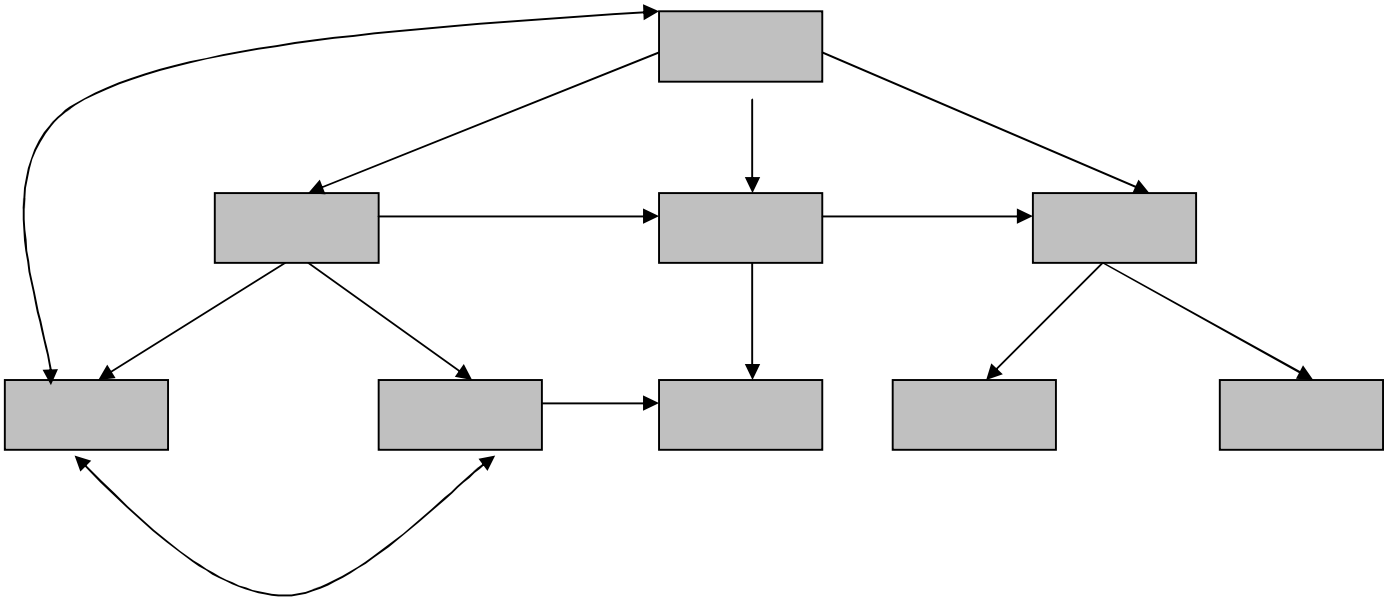


In a full mesh, the number of links is equal to the number of pages \times (number of pages $- 1$). In reality, most sites tend to use a partial mesh style with cross-links to only the most important pages.

Mixed Forms

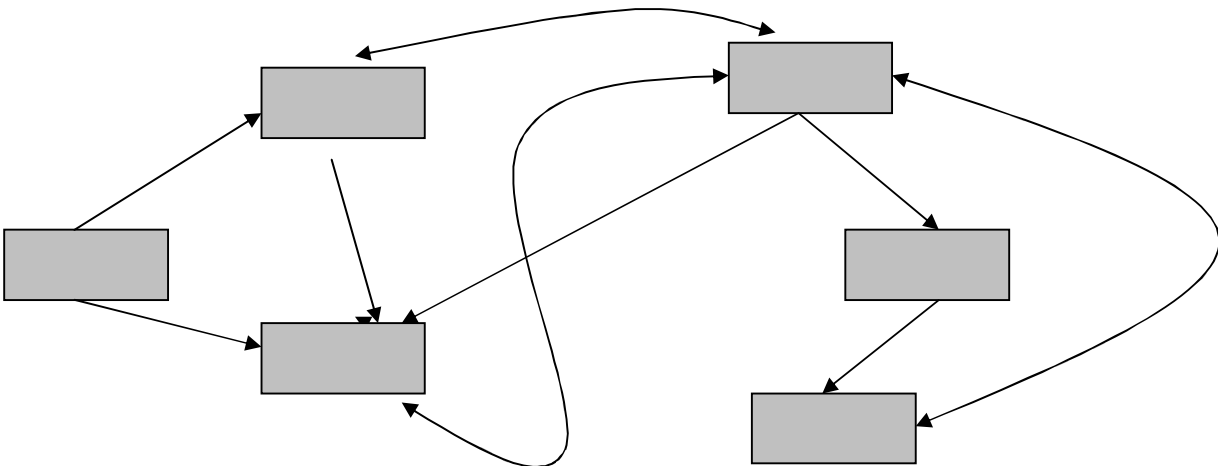
In some cases, there will be a need to augment the hierarchy to allow choices to bubble up to the top. This structure is called a mixed form or a mixed hierarchy, as the tree is the dominant form of the structure. A mixed form is probably the most common form of the site organization used on the Web. Though spatial organization is not as pronounced as in the other site structures, a hierarchy is still generally evident in most mixed sites.

Mixed hierarchy



Web Style

When too many cross-links, skip-aheads, and other augmentations are made to a structured documentation collection, the form will become unclear to the user. When a collection of documents appears to have no discernible structure, it is called a pure web as shown below:



A pure-web structure can be difficult to use because it lacks a clear spatial orientation. Though information can be accessed quickly if the correct choice is made, it may be difficult to orient oneself in a Web site with an unclear structure. Yet the benefit of a less structured form is that it provides a great deal of expressiveness.

Users and Site Structures

While a linear structure may be easier for users to comprehend than a mixed tree or pure web, users do not necessarily memorize the layout of the site or visualize a flowchart in their head as they move around. The entry and exit are really the key milestones for the users. Therefore, another way to categorize Web sites would be on the number of entry points to a site. When a site exposes all documents with public URLs, it could be said to exhibit a porous structure. In contrast, a site with a solid structure would be one that severely limits the entry points to the site to a few URLs or even single URL. The following figure presents the graphical representation of porous, semi porous, and solid site structures.

The basic pros and cons of the two site forms are summarized in the below table:

Site Type	Pros	Cons
Porous form	<p>Puts user in control.</p> <p>Allows the user to enter any URL directly or enter by bookmark.</p>	<p>Decreases ability to change deep pages without addressing outside linking.</p> <p>Does not easily provide a common entry point for announcement, setup, or orientation information.</p>
Solid form	<p>Does not expose site structure, making modification and maintenance easier.</p> <p>Forces user to enter through known points.</p> <p>Makes tracking of users more predictable.</p>	<p>Removes user from control.</p> <p>May limit the effectiveness of outside search engines.</p>

Deep Vs. Shallow Sites Another way to characterize sites would be the number of clicks required to reach a destination. In this regard the following points could be kept in mind:

1. Aim for a site-click depth of three.
2. Aim for positive feedback indicating progress towards a destination every click, with a maximum of three clicks without feedback.
3. Even for wide-site structures, consider a range of 25-81 links per page when page links are ideally clustered.
4. The more important the page, the more redundant links should be provided to it.
5. Redundant links in a site should be no more than 10-20 percent of a page's total exit links.

Specific Types of Web Sites

There are numerous ways to characterize sites, including their audience, their frequency of change, or the structure. One very general way to categorize sites would be as commercial, informational, entertainment, navigational, community, artistic, or personal. The general goals, audience, and features of each type of site vary dramatically. Therefore, do not apply the same design philosophy to each form.

Commercial Sites

Commercial sites are those sites that are built primarily to support the business of some organization. Generally, the primary audience of a commercial site is potential and current customers of the organization. A secondary audience often includes potential and current investors, potential employees, and interested third parties such as the news media or even competitors. Common purpose for commercial sites includes:

1. **Basic information distribution:** The site is used to distribute information about products and services provided by the organization.
2. **Support:** Portions of the site might be built to provide information to help existing customers effectively use products or services provided by the organization.

- 3. Investor relations:** A public company or one seeking outside investment might build a site or a section within a site to distribute information about the current financial situation of the company as well as future opportunities for investment.
- 4. Public relations:** Many firms use their Web sites to distribute information to various newsgathering organizations as well as provide general goodwill information to the community.
- 5. Employee recruiting:** A web site is often used to post information about employment opportunities and benefits of working for a company.
- 6. E-commerce:** A growing number of commercial Web sites allow a visitor, whether an end consumer or a business partner like a reseller, to conduct business directly on the web site. Common facilities supported by e-commerce sites include transactions like ordering, order-status inquiries, and account-balance inquiries.

The overriding purpose of any commercial site is to serve the user in a way that hopefully benefits the company either directly or indirectly.

Informational

Government, educational, news, nonprofit organizations, religious groups, or various social-oriented sites are often considered informational sites. The primary purpose of the site may be to inform for reasons beyond causing a transaction to happen. About all that can be said is that the audience of the site is someone who has an interest or requirement to view the information provided.

Entertainment

Entertainment sites are generally commercial, but they bear special consideration. The purpose of an entertainment site is simply to entertain the site's visitors. Web sites that are built to entertain are often required to break with convention to be successful. Entertainment sites may find novelty or surprise more useful than structure or consistency.

Navigational

A navigational site helps people find their way on the Internet. These sites are called portals since the sites serve as major hubs pointing to other destinations. Navigational sites would also include search engines or site directories that are backbone of many portal sites.

A portal is a site that is generally a primary starting point for a user's online journey and serves to help people find information. Portals often attempt to provide as much information and serve as many tasks for the user as possible to encourage them to stay or to at least continually revisit the site.

Community

A community site is one whose purpose is to create a central location for members or a particular community to congregate and interact. Community sites are very interactive and are often dynamically generated and personalized. A community site is very informational in nature, not just to find content that is interesting to them but also to interact with other like-minded individuals.

Artistic

An artistic site is a site that is purely the expression of the individual or artist. The purpose of the site would be to inspire, enlighten, or entertain its viewers. In some cases, the site may simply be the product of the artist just trying to express his or her feelings. They may not really care what the viewer thinks of the site. As long as the site makes the artist happy, it is successful.

Personal

Similar to an artistic site, a personal often called a personal home page or just a home page is often an expression of its creator. Personal pages may be built to inform friends or family, or they might just be built to try to learn a new skill like HTML. The purpose of the personal page is to personify the individual on the Web. However, Uses should consider that stating all your likes and dislikes online in the form of a personal page is a direct-marketer's dream. Profiles are easy to build from such information and may result in highly targeted and potentially intrusive junk mail and junk email.

Picking a site structure

The idea of picking the correct structure for a web site by organizing information into a collection of pages is often called information architecture. Novice users prefer sites with predictable structure and may put up with extra clicks or a lack of control to achieve a comfortable balance. Power users or frequent site users want control and will favor structures that provide more navigation choices. The key point of site structure is to make the sit easier for the user to navigate. While structure can improve a site's organization, users may not always be aware of a site's form as

they navigate towards desired content or attempt to complete a particular task. So, any structure that we choose for a site should help users navigate around and improve their likelihood of success.

6.5. Designing a web Site

The main purpose of any website is to deliver specific information or services in an organized and user friendly manner. The success of the website is measured in terms of the number of visitors to the site and the ease with which the visitors find the required information. An effective and efficient website is highly influenced by how well interface is designed and managed. Keeping the scenario in mind, one cannot ignore the importance of proper planning and designing considering the legal rights and hosting the website in a reliable web hosting server.

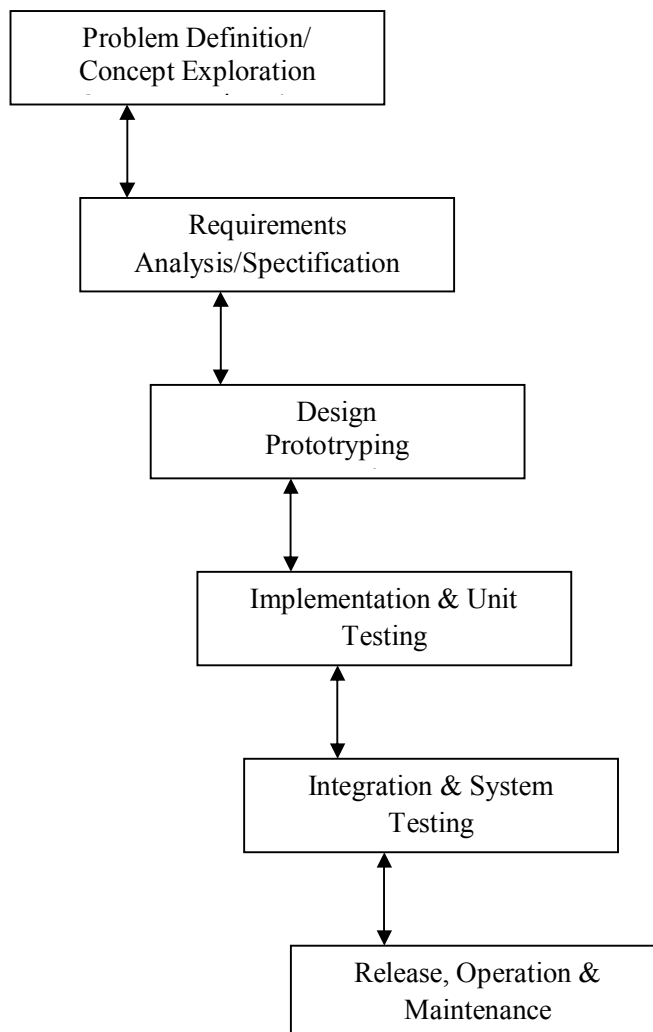
6.5.1. Web design process and Web Site design process

Building a modern Web site can be challenging, so site builders should adopt a methodology or process model to guide the development process and hopefully minimize risk, manage complexity, and generally improve the end result. To help reduce the difficulty in construction sites, we should adopt a process model that describes the various phases involved in Web site development. Each step can then be carefully performed by the developer, using guidelines and documentation along the way that tell the developer how to do things and ensure that each step is carried out properly.

Basic Web Process Model

The basic model starts with the big picture and narrows down to the specific steps necessary to complete the site. In software engineering this model is called the waterfall model, or sometimes the software lifecycle model, because it describes the phases in the lifetime of software. Each stage in the waterfall model proceeds one after another until conclusion. The model starts with a planning stage, then a design phase, then implementation and testing, and ends with a maintenance phase. The actual number of steps and their names varies from person to person, but a general idea of the waterfall model is shown below:

Fig. The Waterfall Model



The good thing about the pure waterfall approach is that it makes developers plan everything up front. That is also its biggest weakness. Another problem is that this process doesn't deal well with change. However, the waterfall model for site design continues to be very popular because it is both easy to understand and easy to follow. Furthermore, the distinct steps in the process appeal to management as they can be easily monitored and serve as project milestones.

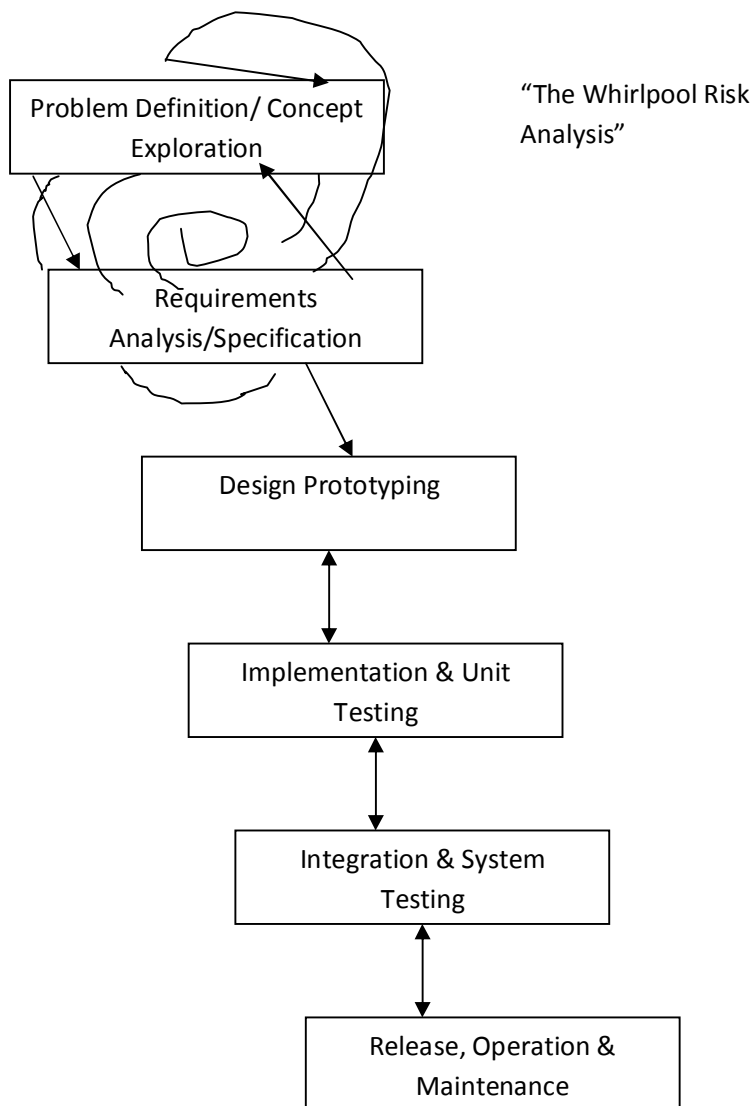
Modified Waterfall

One important aspect of the waterfall model is that it forces planning up front. Because of all the steps required in the process, many developers tend to rush through the early stages and end up repeating them again later on. The process is so rigid that it doesn't support much

exploration and may cause unnecessary risk. One possible improvement is to spend more time in the first few stages of the waterfall and iterate a few times, exploring the goals and requirements of the site before entering into the design and implementation phase. Because of the cyclical nature of this process, it has been dubbed the modified waterfall with whirlpool to relate to the small whirlpools that are often found before a waterfall in nature. When you approach a project with high degree of uncertainty, the modified waterfall with whirlpool approach is a good idea. The various steps in modified waterfall model can be illustrated with the help of a figure as shown below:

Fig: Modified waterfall with risk-analysis whirlpool

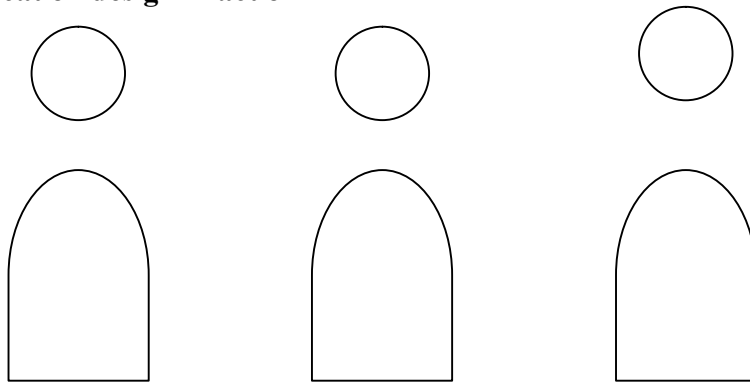
The Waterfall Model



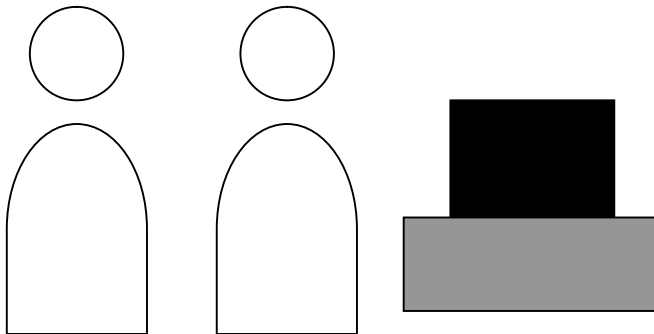
Joint Application Development Model

The last software development process model that makes sense for Web site development is called joint application design or JAD. It is also called evolutionary prototyping because it involves evolving a prototype site to its final form in a series of steps. Rather than creating a mock site to test a theory, a prototype is built and shown to the user. The user then provides direct feedback that is used to guide the next version of the prototype, and so on until the final form is developed. The basic concept of JAD is shown in the following figure:

Fig: Joint application design in action



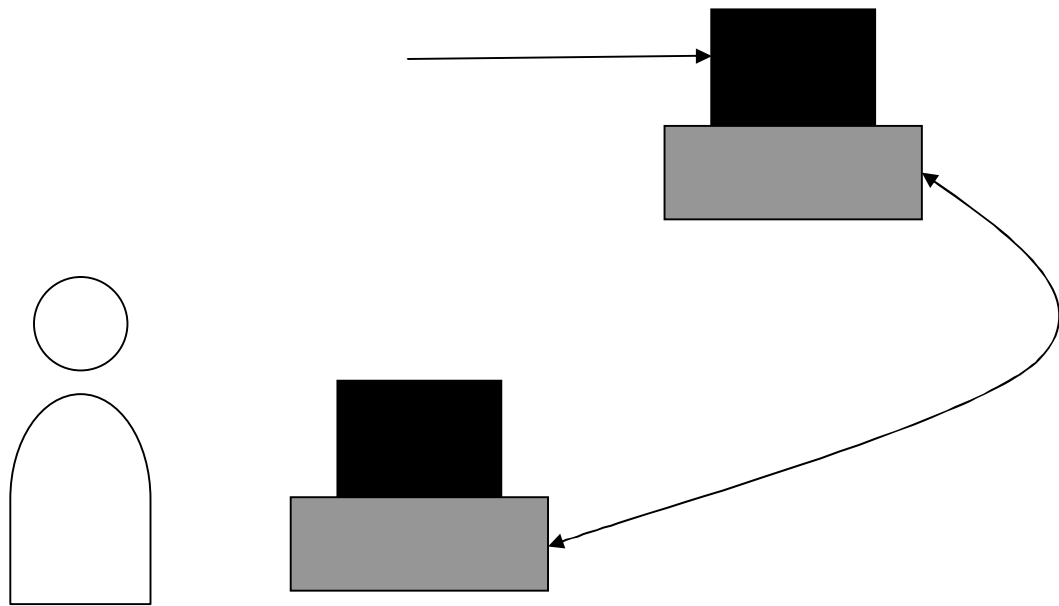
1. Developer talks to clients to understand requirements. Makes first prototype.



2. Client tries prototype and suggests changes

3. And extensions.

4. If OK, release.



3. Developer makes a new prototype. Return to step 2.

Many aspects of the JAD process model seem appropriate for Web development, particularly when it is difficult to determine the specifics of the project. The process is very incremental as compared to the large release approach of the waterfall model, so it also appears to be faster. However, JAD can have some serious drawbacks. First, letting users see an unfinished site could harm the relationship between the users and developer. Even when users want to actively participate in guiding the project, we must always remember that users are not designers. This guiding Web design principle should always be remembered as users may steer development off course with unrealistic demands. Projects run in a JAD style are also difficult to budget for since the number of revisions can't be predicted. The core concept behind JAD is to build the wrong site numerous times until the correct site falls out. Despite its drawbacks, JAD has its place in Web development, particularly in maintenance projects.

Approaching a Web Site Project

Site development rarely works in a consistent manner because of the newness of the field, the significant time constraints, and the ever-changing nature of the Web projects. To guide development, a process model should be adopted at the start of the project. If the site is brand new or the addition is very complex, the waterfall model or the modified waterfall with whirlpool model should be adopted. If the project is a maintenance project, is relatively simple, or has many unknown factors, joint application design may make sense. Regardless of the project, the first step is always the same. Set the overall goal for the project.

Goals and Problems

As familiarity with the Web has grown, the reasons for having Web sites have become clearer. Today, site goals have become important and are usually clearly articulated up front. Coming up with a goal for a Web site isn't difficult. The problem is refining it.

Brainstorming

Generally, coming up with a goal statement is fairly straightforward. The largest problem is keeping the statement concise and realistic. To determine goals, a brainstorming session is often required. The purpose of a brainstorming session is simply to bring out as many possible ideas about the site as possible.

Narrowing the Goal

During the brainstorming session, all ideas are great. The point of the session is to develop what might be called the wish list. A wish list is a document that describes all possible ideas for inclusion in a site regardless of price, feasibility, or applicability. However, eventually the wish list will have to be narrowed down to what is reasonable and appropriate for the site. This can be a significant challenge with a site with many possible goals.

Audience

The best way to narrow a goal is to make sure that the audience is always considered. Consider asking some basic questions about the site's users, such as:

Where they are located?

How old are they?

What is their gender?

What language they speak?

How technically proficient are they?

What kind of computer, connection to the Internet and browser would they

probably use?

Next, consider what the users are doing at the site:

How did they get to the site?

What do the users want to accomplish at the site?

When will they visit the site?

How long will they stay during a particular visit?

From what pages will they leave the site?

When will they return to the site, if ever?

While you might be able to describe the user from these questions, you should quickly determine that your site would probably not have one single type of user with a single goal. For most sites, there are many types of users, each with different characteristics and goals.

User profiling

The best way to understand users is to actually talk to them. If possible, interview the users directly. A survey may also be appropriate. From user interviews, surveys, or even just thinking about users generally, you should attempt to create stereotypical but detailed profiles of common users.

Requirements

Based on the goals of the site and what the audience is like, the site's requirements should begin to present themselves. What kind of content will be required? What kind of look should the site have? What types of programs will have to be built? And so on. Requirements will begin to show site costs and potential implementation problems. If the requirements seem excessive over the potential gain, then the goal stage should be revisited or question if the audience was accurately defined.

The Site Plan

Once a goal, audience, and site requirements have been discussed and documented, a formal site plan should be drawn up. The site plan should contain the following sections:

1. Short goal statement: This section would contain a brief discussion to explain the overall purpose of the site and its basic success measurements.

2. Detailed goal discussion: This section would discuss the site's goals in detail and provide measurable goals to verify the benefit of the site.

3. Audience discussion: This section would profile the users that would visit the site. It describes both audience characteristics and the tasks the audience would try to accomplish at the site.

4. Use scenario discussion: This section discusses the various task visit scenarios for the site's users. Start first with how the user will arrive at the site and then follow the visit to its conclusion.

5. Content requirements: This section should provide a list of all text, images, and other media required in the site.

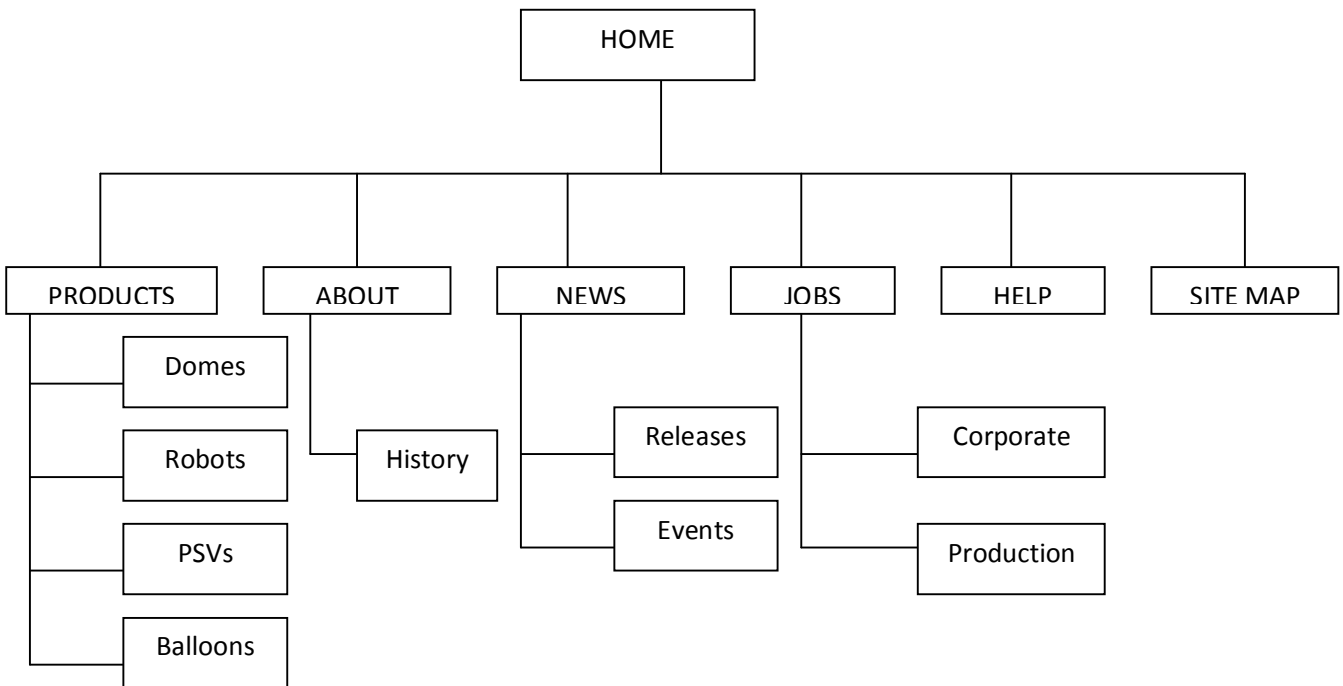
6. Technical requirements: This section should provide an overview of the types of technology the site will employ, such as HTML, JavaScript, CGI, Java, plug-ins, and so on.

7. Visual requirements: This section should outline basic considerations for interface design. This section may outline some specifics such as font or color use, but many of the details of the details of the site's visuals will be determined later in the development process.

8. Delivery requirements: This section should indicate the delivery requirements, particularly any hosting considerations. A basic discussion of how many users will visit the site. How many pages will be consumed on a typical page, and the size of a typical page should be included in this section.

9. Site structure diagram: This section should provide a site structure or flow diagram detailing the various sections within a site. A site diagram will look something like the one shown below:

Fig: Typical site diagram



10. Staffing: This section should detail the resources required to execute the site. Measurements can be in simple man-hours and should relate to each of the four staffing areas: content, technology, visual design, and management.

11. Time line: The time line should show how the project would proceed using the staffing estimates combined with the typical waterfall process.

12. Budget: A budget is primarily determined from the staffing requirements and the delivery requirements. However, marketing costs or other issues such as content licensing could be addressed in the budget.

Design Phase

The design or prototyping stage brings form to the project. During this phase, both technical and visual prototypes should be developed. The following rules are worth to consider during this phase.

1. Always collect content before design if at all possible.

2. Visual design should proceed in a top-down fashion from home page to sub section pages and finally content pages.
3. Always consider the bordering effect of the browser window when developing visual composites.
4. Don't stick on to your design prototypes. Listen to your users and refine your designs.

The Mock Site

After all design prototypes have been finalized, it is time to create the mock or alpha site. Implementation of the mock site starts first by cutting a digital comp into its pieces and assembling the pages using HTML and CSS. Once the mock site is assembled, the site should be fully navigable but contain no content.

Beta Site Implementation

Once the mock site is acceptable, it is time to actually implement the real site. Real content should be placed in pages, and back-end components and interactive elements should be integrated with the final visual design.

Testing

Testing is key to a positive user takeaway value. Always remember the following design rules:

1. Sites always have bugs, so test your site well.
2. Testing should address all aspects of a site, including content, visuals, function, and purpose.

The following are the basic aspects of Web testing:

1. Visual Acceptance Testing: Visual acceptance testing ensures the site looks the way it was intended.

2. Functionality Testing: Most basic function of a page is to simply render onscreen. However, most sites contain at least basic functions such as navigation. Make sure to check every link in a site and rectify any broken links.

3. Content Proofing: Make sure content is all in place and that word usage is consistent. And always remember to check the spelling.

4. System and Browser Compatibility Testing: System and browser restrictions have been respected during development, but this must be verified during testing. Make sure the site works under the specified browsers.

5. Delivery Testing: Check to make sure the site is delivered adequately. Try browsing the site under real user conditions.

6. User Acceptance Testing: User acceptance testing should be performed after the site appears to work correctly. This testing is also called beta testing. Do not perform this type of testing until the more obvious bugs have been rectified, as stated by the following rule:

- User test is the most important form of testing and should be performed last.

Release and Beyond

Once the site is ready to be released, observe the site in action. Consider the following rule in this regard:

Site development is an ongoing process that includes the stages as plan, design, develop, release, and repeat. After adequate testing overtime maintenance and continued vigilance will be required otherwise your finely crafted site will begin to degrade.

6.6. Elements of web site design

Here are seven key elements of modern web design:

- **A Strong, but Limited, Color Palette:** This might sound rudimentary, but color schemes and color usage are very important when it comes to modern web design. A strong color palette will help create cohesiveness between everything your business puts out. Plenty of White Space: This goes along with the last modern website design element, but white space is also very attractive. It doesn't necessarily even have to be white.

White space is a term used for the amount of "empty" space that acts as a buffer between all the elements on your page, including copy, sidebar, margins, etc. Things should have room to breathe; if your website is crowded, it is very hard to direct the attention of your visitor's eye.

- **Relevant Calls-to-Action:** Websites are meant to connect you with the people who are interested in your content, products, and services. Once this connection is made, you want to retain some sort of relationship with these visitors.

Things like email subscription forms, free downloadable ebooks or whitepapers, free product forms, free consultations, or other invites are great calls-to-action (CTAs). These should be strategically incorporated into your website design and are very important for gathering the contact information (typically just an email address) of your visitors so that you can continue conversations with them as leads and convert them into customers.

- **Clean Backend Coding:** This modern website design element is one that you might not notice visually, but one that is probably the most important when it comes to the functionality of your site. Behind every website is a great deal of coding in the backend that will dictate how your website performs.
- **Design for the User First:** This element of modern website design is exactly what it sounds like: You should design your site for the user, not just to boost your rankings. Companies, out of a sense of desperation to get better rankings, tend to do things that are “good” for Google but bad for the user.
- **SEO-Boosting Elements:** There are modern website design elements that can greatly improve the Search Engine Optimization (SEO) of your site. A lot of these are invisible to the naked eye and also appear in the backend coding of your pages and posts.

Design tricks like meta tags, title tags, heading tags, and other HTML coding go a long way in helping your site climb the ranks of Google's search engine. Make sure you fill out, tweak, and optimize these elements so they are relevant to your site and better your search ranking.

- **Speed Optimization:** Optimizing for speed is an imperative design element that should not be overlooked. With today's technology, people expect things to load immediately, or they'll probably throw in the towel three seconds later and never return. As a business, you don't want leads and prospects to think negatively of your brand just because your website is slow.

6.7. Web page and Layout

Web page: A web page is nothing more than a file, a HTML file to be exact. It's called HTML because web page documents have the file extension .html or .htm. HTML stands for **H**yper **T**ext **M**ark-up **L**anguage.

Running a web page

Once you have your HTML document on the floppy disc or your hard drive, you'll need to open it up in the browser. It's easy enough. Since you're using a browser to look at this Primer, follow along.

1. Under the FILE menu at the very top left of this screen, you'll find OPEN, OPEN FILE, OPEN DOCUMENT, or words to that effect.
2. Click on it. Some browsers give you the dialogue box that allows you to find your document right away. Internet Explorer, and later versions of Netscape Navigator, requires you to click on a BROWSE button or OPEN FILE button to get the dialogue box. When the dialogue box opens up, switch to the A:\ drive (or the floppy disc for MAC users) and open your document. If you saved the file to your hard drive, get it from there.
3. You might have to then click an OK button. The browser will do the rest.

Web Layout: A website layout is a pattern (or framework) that defines a website's structure. It has the role of structuring the information present on a site both for the website's owner and for users. It provides clear paths for navigation within webpages and puts the most important elements of a website front and center. Web Layout displays the page width, text position and format as the document would appear in a browser. When it comes to the success of a website, the most important aspect of the site is the page's layout. The website must be clean with an easy-to-follow navigation system to contribute to a usable webpage layout. A good layout will also encourage the visitor to view more pages on the site.

6.8. Summary

- Web page design is a process of conceptualization, planning, modeling, and execution of electronic media content delivery via Internet in the form of technologies (such as markup

languages) suitable for interpretation and display by a web browser or other web-based graphical user interfaces (GUIs).

- The basic model starts with the big picture and narrows down to the specific steps necessary to complete the site.
- In software engineering this model is called the waterfall model, or sometimes the software lifecycle model, because it describes the phases in the lifetime of software.
- The last software development process model that makes sense for Web site development is called joint application design or JAD. It is also called evolutionary prototyping because it involves evolving a prototype site to its final form in a series of steps.

- Brainstorming: Generally, coming up with a goal statement is fairly straightforward. The largest problem is keeping the statement concise and realistic. To determine goals, a brainstorming session is often required. The purpose of a brainstorming session is simply to bring out as many possible ideas about the site as possible.
- Web page: A web page is nothing more than a file, a HTML file to be exact. It's called HTML because web page documents have the file extension .html or .htm. HTML stands for Hyper Text Mark-up Language.
- Web Layout: A website layout is a pattern (or framework) that defines a website's structure. It has the role of structuring the information present on a site both for the website's owner and for users.

6.9. SAQ

1. Explain various steps involved in Web design process.
2. What are the key elements in Web site design?
3. Explain various types of Web site structure.

Unit – VII Basic HTML

7.0. Structure

7.1. Objective

7.2. Basic HTML

7.2.1. HTML

7.2.2. Structure of HTML

7.3. Working with text

7.3.1. Heading tag, Font tag, Address tag, Line break and paragraph tags

7.4. Center tag and marquee tag

7.5. List tag

7.6. Summary

7.7. SAQ

7.1. Objective

In this chapter we discuss all basic tags of HTML

7.2. Basic HTML

HTML is the standard markup language for creating Web pages.

- HTML stands for Hyper Text Markup Language
- HTML describes the structure of a Web page
- HTML consists of a series of elements
- HTML elements tell the browser how to display the content
- HTML elements are represented by tags
- HTML tags label pieces of content such as "heading", "paragraph", "table", and so on
- Browsers do not display the HTML tags, but use them to render the content of the page

7.2.1. HTML

HTML (Hypertext Markup Language) is a markup language, which consists of tags embedded in the text of a document.

- **Hyper** is the opposite of linear. It used to be that computer programs had to move in a linear fashion. This before this, this before this, and so on. HTML does not hold to that pattern and allows the person viewing the World Wide Web page to go anywhere, any time they want.
- **Text** is what you will use. Real, honest to goodness English letters.

- **Mark up** is what you will do. You will write in plain English and then mark up what you wrote. More to come on that in the next Primer.

- **Language** because they needed something that started with "L" to finish HTML and Hypertext Markup Louie didn't flow correctly. Because it's a language, really -- but the language is plain English

The browser reading the document interprets these markup tags to help format the document for subsequent display to a reader. However, the browser makes many of the decisions about layout. Remember, web browsers are available for a wide variety of computer systems.

The browser thus displays the document with regard to features that the viewer selects either explicitly or implicitly. Factors affecting the layout and presentation include:

- The markup tags used.
- The physical page width.
- The fonts used to display the text.
- The color depth of the display.

Basic concepts of HTML

The HTML document on the word processor, or Notepad, WordPad, or Simple Text. When you are finished creating the HTML document, you'll then open the document in a browser, like Netscape Navigator. The browser will interpret the HTML commands for you and display the Web page. HTML documents must be text only. When you save an HTML document, you must save only the text.

The Word Processor

When you write to the word processor you will need to follow a few steps:

1. Write the page as you would any other document.
2. When you go to save the document (Here's the trick), ALWAYS choose SAVE AS.

3. When the SAVE AS box pops up, you will need to save the page in a specific format. Look at the SAVE AS dialogue box when it pops up: Usually at the bottom, you find where you will be able to change the file format.

4. If you have a PC, save your document as ASCII TEXT DOS or just TEXT. Either one will work.

5. If you have a MAC, save your document as TEXT.

6. When I started writing HTML, I saved pages by assigning every Web page its own floppy disc. It just helped me keep it all straight, but if you want to save right to your hard drive, do it. I only offer the floppy disc premise as a suggestion.

Note: It is very important to choose SAVE AS EVERY time you save your document. If you don't, the program won't save as TEXT, but rather in its default format. In layman's terms -- use SAVE AS or screw up your document.

Name Your Document

What you name your document is very important. You must first give your document a name and then add a suffix to it. That's the way everything works in HTML. You give a name and then a suffix.

Follow this format to name your document:

1. Choose a name. Anything. If you have a PC not running Windows 95, you are limited to eight letters, however.

2. Add a suffix. For all HTML documents, you will add either ".htm" or ".html".

Running a web page

Once you have your HTML document on the floppy disc or your hard drive, you'll need to open it up in the browser. It's easy enough. Since you're using a browser to look at this Primer, follow along.

1. Under the FILE menu at the very top left of this screen, you'll find OPEN, OPEN FILE, OPEN DOCUMENT, or words to that effect.

2. Click on it. Some browsers give you the dialogue box that allows you to find your document right away. Internet Explorer, and later versions of Netscape Navigator, requires you to click on a BROWSE button or OPEN FILE button to get the dialogue box. When the dialogue box opens up, switch to the A:\ drive (or the floppy disc for MAC users) and open your document. If you saved the file to your hard drive, get it from there.

3. You might have to then click an OK button. The browser will do the rest.

7.2.2. Structure of HTML

HTML Markup Tags

HTML markup tags are usually called HTML tags

- HTML tags are keywords surrounded by angle brackets like <html>
- HTML tags normally come in pairs like and
- The first tag in a pair is the start tag, the second tag is the end tag

Note: The start and end tags are also called the opening and closing tags

Syntax and example

```
<html>
<body>
<h1>My First HTML header</h1>
<p>My first HTML paragraph</p>
</body>
</html>
```

When a browser displays a web page, it will not display the markup tags.

The text between the <html> and </html> tags describes a web page.

The text between the <body> and </body> tags is displayed in the web browser.

The text between the <p> and </p> tags is displayed as paragraphs.

The text between the and tags is displayed in a bold font.

7.3. Working with text

HTML Elements

An HTML element usually consists of a **start** tag and an **end** tag, with the content inserted in between:

```
<tagname>Content goes here...</tagname>
```

7.3.1. Heading tag, font tag, address tag, line break and paragraph tags

Headings are defined with the <h1> to <h6> tags. <h1> defines the largest heading. <h6> defines the smallest heading.

```
<h1>This is a heading</h1>
```

```
<h2>This is a heading</h2>
```

```
<h3>This is a heading</h3>
```

Out put

This is a heading

This is a heading

This is a heading

HTML Paragraphs

Paragraphs are defined with the <p> tag.

Syntax

```
<p> paragraph</p>
```

```
<p> another paragraph</p>
```

Example

<html>

<body>

<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>

<p>Use heading tags only for headings.

Don't use them to make something BIG or BOLD.

Use other tags for that. </p>

</body>

</html>

Line Breaks

Use the
 tag if you want a line break (a new line) without starting a new paragraph:

<p>This is
a para
graph with line breaks</p>

The
 tag is an empty tag. It has no end tag like </br>.

You can read more about empty HTML tags in the next chapter of this tutorial.

Syntax

 or

Even if
 works in all browsers, writing
 instead is more future proof.

Example

<html>

```
<body>
<p>This is<br>a para<br>graph with line breaks</p>
</body>
</html>
```

Output

This is
a para
graph with line breaks

Comments

Comments can be inserted in the HTML code to make it more readable and understandable.
Comments are ignored by the browser and not displayed.

Syntax

```
<!-- This is a comment -->
```

Note: There is an exclamation point after the opening bracket, but not before the closing bracket.

Example

```
<html>
<body>
<!--This comment will not be displayed-->
<p>This is a regular paragraph</p>
</body>
</html>
```

Horizontal tag

The <hr> tag inserts a horizontal rule. In HTML the <hr> tag has no end tag.

Syntax

```
<hr>
```

Example

```
<html>
```

```
<body>
```

```
<p>The hr element defines a horizontal rule:</p>
```

```
<hr />
```

```
<p>This is a paragraph</p>
```

```
<hr />
```

```
<p>This is a paragraph</p>
```

```
<hr />
```

```
<p>This is a paragraph</p>
```

```
</body>
```

```
</html>
```

Output

The hr tag defines a horizontal rule:



This is a paragraph



This is a paragraph



This is a paragraph

HTML Attributes

HTML tags can have attributes. Attributes provide additional information about the HTML element.

Attributes always come in name/value pairs like this: name="value".

Attributes are always specified in the start tag of an HTML element.

Example for align

`<h1>` defines the start of a heading.

`<h1 align="center">` has additional information about the alignment.

Example body

`<body>` defines the body of an HTML document.

`<body bgcolor="yellow">` has additional information about the background color.

Tag	Description
<code><html></code>	Defines an HTML document
<code><body></code>	Defines the document's body
<code><h1></code> to <code><h6></code>	Defines header 1 to header 6
<code><p></code>	Defines a paragraph
<code>
</code>	Inserts a single line break
<code><hr></code>	Defines a horizontal rule
<code><!--></code>	Defines a comment

Text Formatting Tags

Tag	Description
-----	-------------

	Defines bold text
<big>	Defines big text
	Defines emphasized text
<i>	Defines italic text
<small>	Defines small text
	Defines strong text
<sub>	Defines subscripted text
<sup>	Defines superscripted text
<ins>	Defines inserted text
	Defines deleted text
<s>	Deprecated. Use instead
<strike>	Deprecated. Use instead
<u>	Deprecated. Use styles instead

"Computer Output" Tags

Tag	Description
<code>	Defines computer code text
<kbd>	Defines keyboard text
<samp>	Defines sample computer code
<tt>	Defines teletype text

<var>	Defines a variable
<pre>	Defines preformatted text
<listing>	Deprecated. Use <pre> instead
<plaintext>	Deprecated. Use <pre> instead
<xmp>	Deprecated. Use <pre> instead

Citations, Quotations, and Definition Tags

Tag	Description
<abbr>	Defines an abbreviation
<acronym>	Defines an acronym
<address>	Defines an address element
<bdo>	Defines the text direction
<blockquote>	Defines a long quotation
<q>	Defines a short quotation
<cite>	Defines a citation
<dfn>	Defines a definition term

Reserved characters in HTML must be replaced with character entities.

Character Entities

Some characters are reserved in HTML. For example, you cannot use the greater than or less than signs within your text because the browser could mistake them for markup.

If we want the browser to actually display these characters we must insert character entities in the HTML source.

A character entity looks like this: `&entity_name;` OR `&#entity_number;`

To display a less than sign we must write: `<` or `<`

The advantage of using an entity name instead of a number is that the name often is easier to remember. However, the disadvantage is that browsers may not support all entity names (while the support for entity numbers is very good).

Non-breaking Space

The most common character entity in HTML is the non-breaking space.

Normally HTML will truncate spaces in your text. If you write 10 spaces in your text HTML will remove 9 of them. To add lots of spaces to your text, use the ` ` character entity.

Example

```
<html>
```

```
<body>
```

```
<p>Character entities</p>
```

```
<p>&X;</p>
```

```
<p>
```

Substitute the "X" with an entity number like "#174" or an entity name like "pound" to see the result.

```
</p>
```

```
</body>
```

```
</html>
```

Out put

Character entities

&X;

Substitute the "X" with an entity number like "#174" or an entity name like "pound" to see the result

Commonly Used Character Entities

Note Entity names are case sensitive!

Result	Description	Entity Name	Entity Number
	non-breaking space	 	
<	less than	<	<
>	greater than	>	>
&	Ampersand	&	&
¢	Cent	¢	¢
£	Pound	£	£
¥	Yen	¥	¥
€	Euro	€	€
§	Section	§	§
©	Copyright	©	©
®	registered trademark	®	®

7.4.Center tag and marquee tag

Center tag : When writing in HTML, the **<center>** tag was used to contain both block-level and inline elements on a web page. Anything contained in the **<center>** tags would be aligned with the middle of the page. But this tag doesn't support in HTML 5 version.

<center> this text will be center-aligned.</center>

The <center> tag is used to center-align text.

Browser Support

Element	Chrome	Internet explore	Firefor	opera
<center>	Yes	Yes	Yes	Yes

Marquee tag: **HTML <marquee> Tag**

The <marquee> is a non-standard HTML tag which was used to create a scrolling text or an image. It was used to make the text/image scroll horizontally across or vertically down the web page. The <marquee> element comes in pairs. It means that the tag has opening <marquee> and closing </marquee> elements. Consider the following example for marquee tag

```
<! DOCTYPE HTML>
<html>
<head>
<title> title of the document </title>
</head>
<body>
<marquee> a scrolling text created with HTML marquee element.
</marquee>
</body>
</html>
```

Use direction attribute of <marquee> element to change the direction of the text/image. See another example where the text scrolls from up to down.

```
<! DOCTYPE HTML>
<html>
<head>
<title> title of the document </title>
</head>
<body>
<marquee direction="down"> a scrolling text created with HTML marquee element.
</marquee>
```

```
</body>
</html>
```

Let us one more example

```
<!DOCTYPE HTML>
<html>
<head>
<title> title of the document </title>
</head>
<body>
<marquee behavior="scroll" direction="up">

</marquee>
</body>
</html>
```

Attributes

The following attributes can be used to adjust the appearance of the <marquee> element.

Attribute	Value	Description
behavior	scroll slide alternate	Defines the scrolling type.
bgcolor	rgb(x,x,x) #xxxxxx colorname	Is used to give a background color.
direction	up down left right	Sets the direction for the scrolling content.
Height	pixels %	Defines the marquee's height.

Attribute	Value	Description
Hspace	Pixels	Defines horizontal space around the marquee.
Loop	Number	Defines how many times the content will scroll. If we don't define this, the content will scroll forever.
scrollamount	Number	Defines the scrolling amount at each interval in pixels. Default value is 6.
scrolldelay	Seconds	Defines how long delay will be between each jump. The default value is 85 and smaller amounts than 60 will be ignored.
truespeed	Seconds	Is used to delay the scroll lesser than 60.
Vspace	Pixels	Defines vertical space around the marquee.
Width	pixels %	Defines the marquee's width.

7.5. List tag

HTML lists are used to present list of information in well formed and semantic way. There are three different types of list in HTML and each one has a specific purpose and meaning.

- **Unordered list** — Used to create a list of related items, in no particular order.
- **Ordered list** — Used to create a list of related items, in a specific order.
- **Description list** — Used to create a list of terms and their descriptions.

Note: inside a list item you can put text, images, links, line breaks, etc. you can also place an entire list inside a list item to create the nested list.

HTML Unordered Lists

An unordered list is a list of items. The list items are marked with bullets (typically small black circles).

An unordered list starts with the `` tag. Each list item starts with the `` tag.

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

Here is how it looks in a browser:

- Coffee
- Milk

Inside a list item you can put paragraphs, line breaks, images, links, other lists, etc.

Ordered Lists

An ordered list is also a list of items. The list items are marked with numbers.

An ordered list starts with the `` tag. Each list item starts with the `` tag.

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

Here is how it looks in a browser:

1. Coffee
2. Milk

Inside a list item you can put paragraphs, line breaks, images, links, other lists, etc.

Description Lists

A definition list is not a list of items. This is a list of terms and explanation of the terms.

A definition list starts with the `<dl>` tag. Each definition-list term starts with the `<dt>` tag. Each definition-list definition starts with the `<dd>` tag.

```
<dl>
```

```
<dt>Coffee</dt>
<dd>Black hot drink</dd>
<dt>Milk</dt>
<dd>White cold drink</dd>
</dl>
```

Here is how it looks in a browser:

Coffee

Black hot drink

Milk

White cold drink

Inside a definition-list definition (the <dd> tag) you can put paragraphs, line breaks, images, links, other lists, etc.

Example

```
<html>
<body>
<h4>Numbered list:</h4>
<ol>
<li>Apples</li>
<li>Bananas</li>
<li>Lemons</li>
<li>Oranges</li>
</ol>
<h4>Letters list:</h4>
<ol type="A">
<li>Apples</li>
<li>Bananas</li>
<li>Lemons</li>
```

Oranges

<h4>Lowercase letters list:</h4>

<ol type="a">

Apples

Bananas

Lemons

Oranges

<h4>Roman numbers list:</h4>

<ol type="I">

Apples

Bananas

Lemons

Oranges

<h4>Lowercase Roman numbers list:</h4>

<ol type="i">

Apples

Bananas

Lemons

Oranges

</body>

</html>

Output

Numbered list:

1. Apples
2. Bananas
3. Lemons
4. Oranges

Letters list:

- A. Apples
- B. Bananas
- C. Lemons
- D. Oranges

Lowercase letters list:

- a. Apples
- b. Bananas
- c. Lemons
- d. Oranges

Roman numbers list:

- I. Apples
- II. Bananas
- III. Lemons
- IV. Oranges

Lowercase Roman numbers list:

- i. Apples
- ii. Bananas
- iii. Lemons
- iv. Oranges

List Tags

Tag	Description
	Defines an ordered list

	Defines an unordered list
	Defines a list item
<dl>	Defines a definition list
<dt>	Defines a definition term
<dd>	Defines a definition description
<dir>	Deprecated. Use instead
<menu>	Deprecated. Use instead

7.6. Summary

- **HTML** stands for Hyper Text Markup Language.
- It is used to design web pages using markup language. ... **HTML** is a markup language which is used by the browser to manipulate text, images and other content to display it in required format. **HTML** was created by Tim Berners-Lee in 1991
- Headings are defined with the <h1> to <h6> tags. <h1> defines the largest heading. <h6> defines the smallest heading.
- <body> defines the body of an HTML document.
- HTML supports ordered, unordered and definition lists.

7.7. SAQ

1. What is HTML? Explain various features of HTML.
2. What are the disadvantages of HTML?
3. Explain the basic structure of HTML with example.
4. Discuss about various elements for text formatting.
5. Explain various heading tags that are used in HTML.
6. What is the use of marquee tag?
7. What is the purpose of span tag?
8. Discuss about various list tags available in HTML.

Unit – VIII Inserting images and tables in Web page

8.0. Structure

8.1. Objective

8.2. Inserting images

8.3. Creating Hyperlinks in a Web page

8.4. Creating Table in a Web page

8.5. Frames

8.6. Summary

8.7. SAQ

8.1. Objective

This unit helps in inserting images and tables in web pages.

8.2. Inserting images

With HTML you can display images in a document

The Image Tag and the Src Attribute

In HTML, images are defined with the `` tag.

The `` tag is empty, which means that it contains attributes only and it has no closing tag.

To display an image on a page, you need to use the `src` attribute. `Src` stands for "source". The value of the `src` attribute is the URL of the image you want to display on your page.

The syntax of defining an image:

```

```

The URL points to the location where the image is stored. An image named "boat.gif" located in the directory "images" on "www.images.com" has the URL: <http://www.images.com/images/boat.gif>.

The browser puts the image where the image tag occurs in the document. If you put an image tag between two paragraphs, the browser shows the first paragraph, then the image, and then the second paragraph

The Alt Attribute

The alt attribute is used to define an "alternate text" for an image. The value of the alt attribute is an author-defined text:

```

```

The "alt" attribute tells the reader what he or she is missing on a page if the browser can't load images. The browser will then display the alternate text instead of the image. It is a good practice to include the "alt" attribute for each image on a page, to improve the display and usefulness of your document for people who have text-only browsers.

Notes: If an HTML file contains ten images - eleven files are required to display the page right. Loading images take time, so my best advice is: Use images carefully

Example

```
<html>
```

```
<body>
```

```
<p>
```

An image:

```

```

</p>

<p>

A moving image:

```

```

</p>

<p>

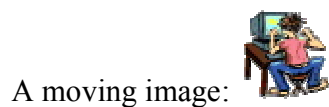
Note that the syntax of inserting a moving image is no different from that of a non-moving image.

</p>

</body>

</html>

Output



Note that the syntax of inserting a moving image is no different from that of a non-moving image.

Example for ALT

<html>

<body>

```

```

<p>

Text-only browsers cannot display images and will only display the text that is specified in the "alt" attribute for the image. Here, the "alt"-text is "Go Left".</p>

<p>

Note that if you hold the mouse pointer over the image, most browsers will display the "alt"-text.

</p>

</body>

</html>

Output



Text-only browsers cannot display images and will only display the text that is specified in the "alt" attribute for the image. Here, the "alt"-text is "Go Left".

Note that if you hold the mouse pointer over the image, most browsers will display the "alt"-text

Image Tags

Tag	Description
	Defines an image
<map>	Defines an image map
<area>	Defines a clickable area inside an image map

8.3. Creating Hyperlinks in a Web page

HTML make it possible to define hyperlinks to other information items located all over the world, thus allowing documents to join the global information space known as the World Wide

Web. Linking is possible because every document on the Web has a unique address, known as a Uniform Resource Locator (URL).

Links and addressing

HTML uses a hyperlink to link to another document on the Web

The Anchor Tag and the Href Attribute

HTML uses the <a> (anchor) tag to create a link to another document.

An anchor can point to any resource on the Web: an HTML page, an image, a sound file, a movie, etc.

The syntax of creating an anchor:

```
<a href="url">Text to be displayed</a>
```

The <a> tag is used to create an anchor to link from, the href attribute is used to address the document to link to, and the words between the open and close of the anchor tag will be displayed as a hyperlink.

The Target Attribute

With the target attribute, you can define where the linked document will be opened.

The line below will open the document in a new browser window:

```
<a href="http://www.rastiya_sanskrit_vidyapeetha.ac.in/"  
target="_blank">Visit RSVP!</a>
```

The Anchor Tag and the Name Attribute

The name attribute is used to create a named anchor. When using named anchors we can create links that can jump directly into a specific section on a page, instead of letting the user scroll around to find what he/she is looking for.

Below is the syntax of a named anchor

```
<a name="label">Text to be displayed</a>
```

The name attribute is used to create a named anchor. The name of the anchor can be any text you care to use.

The line below defines a named anchor:

```
<a name="tips">Useful Tips Section</a>
```

You should notice that a named anchor is not displayed in a special way.

To link directly to the "tips" section, add a # sign and the name of the anchor to the end of a URL, like this:

```
<a href="http:// www.rastiya sanskrit vidyapeetha.ac.in /html_links.asp#tips">
```

```
Jump to the Useful Tips Section</a>
```

A hyperlink to the Useful Tips Section from WITHIN the file "html_links.asp" will look like this:

```
<a href="#tips">Jump to the Useful Tips Section</a>
```

Example

```
<html>
```

```
<body>
```

```
<a href="lastpage.htm" target="_blank">Last Page</a>
```

```
<p>
```

If you set the target attribute of a link to "_blank",

the link will open in a new window.

```
</p>
```

```
</body>
```

</html>

Output

Last Page

If you set the target attribute of a link to "_blank", the link will open in a new window

8.4. Creating Tables in a Web page

Tables are defined with the <table> tag. A table is divided into rows (with the <tr> tag), and each row is divided into data cells (with the <td> tag). The letters td stands for "table data," which is the content of a data cell. A data cell can contain text, images, lists, paragraphs, forms, horizontal rules, tables, etc.

Syntax

```
<table border="1">
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td>row 2, cell 2</td>
</tr>
</table>
```

How it looks in a browser:

row 1, cell 1	row 1, cell 2
row 2, cell 1	row 2, cell 2

Tables and the Border Attribute

If you do not specify a border attribute the table will be displayed without any borders. Sometimes this can be useful, but most of the time, you want the borders to show.

To display a table with borders, you will have to use the border attribute:

```
<table border="1">
<tr>
<td>Row 1, cell 1</td>
<td>Row 1, cell 2</td>
</tr>
</table>
```

Headings in a Table

Headings in a table are defined with the `<th>` tag.

```
<table border="1">
<tr>
<th>Heading</th>
<th>Another Heading</th>
</tr>
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td>row 2, cell 2</td>
</tr>
</table>
```

defined with the `<th>` tag.

How it looks in a browser:

Heading	Another Heading
row 1, cell 1	row 1, cell 2
row 2, cell 1	row 2, cell 2

Empty Cells in a Table

Table cells with no content are not displayed very well in most browsers.

```
<table border="1">
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td></td>
</tr>
</table>
```

How it looks in a browser:

row 1, cell 1	row 1, cell 2
row 2, cell 1	

Note that the borders around the empty table cell are missing (NB! Mozilla Firefox displays the border).

To avoid this, add a non-breaking space () to empty data cells, to make the borders visible:

```

<table border="1">
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td>&nbsp;</td>
</tr>
</table>

```

How it looks in a browser:

row 1, cell 1	Row 1, cell 2
row 2, cell 1	

Table Tags

Tag	Description
<table>	Defines a table
<th>	Defines a table header
<tr>	Defines a table row
<td>	Defines a table cell
<caption>	Defines a table caption
<colgroup>	Defines groups of table columns
<col>	Defines the attribute values for one or more columns in a table
<thead>	Defines a table head

<code><tbody></code>	Defines a table body
<code><tfoot></code>	Defines a table footer

8.5. Frames

A frame is an independent scrolling region, or window, of a Web page. Every Web page may be divided into many individual frames, which can even be nested within other frames. Fixed screen sizes limit how many frames can really be used simultaneously. Each frame in a window may be separated from the others with a border, in this way; a framed document may resemble a table. However, frames aren't a fancy form of tables. Each separate frame may contain a different document, indicated by a unique URL. Because the documents included in a framed region may be much larger than the room available onscreen, each frame may provide a scroll bar or other controls to manipulate the size of the frame. Individual frames usually are named, so that they may be referenced through links or scripting, allowing the contents of one frame to affect the contents of another. This referencing capability is a major difference between tables and frames. Frames provide layout facilities and, potentially, navigation.

Frames are included in a HTML document through the `<FRAMESET>` and `<FRAME>` elements. The `<FRAMESET>` and `<FRAME>` elements constitute the HTML needed for frames.

The disadvantages of using frames are:

- The web developer must keep track of more HTML documents

- It is difficult to print the entire page

The Frameset Tag

- The `<frameset>` tag defines how to divide the window into frames

- Each frameset defines a set of rows **or** columns

The values of the rows/columns indicate the amount of screen area each row/column will occupy

syntax:

```
<FRAMESET>
```

```
<FRAMESET CLASS="class name(s)" COLS="list of columns" ID="unique alphanumeric identifier" ROWS="list of rows" STYLE="style information" TITLE="advisory text" onload="script" onunload="script">
```

```
<FRAME> elements and <NOFRAMES>
```

```
</FRAMESET>
```

syntax:

```
<FRAME>
```

```
<FRAME CLASS="class name(s)" FRAMEBORDER="0 | 1" ID="unique alphanumeric identifier" LONGDESC="URL of description" MARGINHEIGHT="pixels" MARGINWIDTH="pixels" NAME="string" NORESIZE SCROLLING="AUTO | NO | YES" SRC="URL" of frame contents" STYLE="style information" TITLE="advisory text">
```

In the example below we have a frameset with two columns. The first column is set to 25% of the width of the browser window. The second column is set to 75% of the width of the browser window. The HTML document "frame_a.htm" is put into the first column, and the HTML document "frame_b.htm" is put into the second column:

```
<frameset cols="25%,75%">
  <frame src="frame_a.htm">
  <frame src="frame_b.htm">
</frameset>
```

Note: The frameset column size value can also be set in pixels (cols="200,500"), and one of the columns can be set to use the remaining space (cols="25%,*").

Basic Notes - Useful Tips

If a frame has visible borders, the user can resize it by dragging the border. To prevent a user from doing this, you can add noresize="noresize" to the <frame> tag.

Add the <noframes> tag for browsers that do not support frames.

Important: You cannot use the <body></body> tags together with the <frameset></frameset> tags! However, if you add a <noframes> tag containing some text for browsers that do not support frames, you will have to enclose the text in <body></body> tags! See how it is done in the first example below.

Frame Targeting

When using frames, you often may find that making the links in one frame target another frame is beneficial. This way, when a user clicks a button or activates a link in one framed document, the requested page loads in another frame. Link targeting has two steps:

1. Ensure frame naming by setting the NAME attribute in the <FRAME> element to a unique name.

2. Use the TARGET attribute in the <A> element to set the target for the anchor. For example, a link such as loads the site specified by the HREF into the window called Display, if such a frame exists. If the target specified by the name doesn't exist, the link loads over the window it is in.

Some particular values for the TARGET attribute have special meaning, which are summarized in the following table.

Reserved TARGET values:

Value	Meaning
_blank	Load the page into a new, generally unnamed, window.
_self	Load the page over the current frame.
_parent	Load the link over the parent frame.
_top	Load the link over the frames in the window.

No frames

The noframes element displays text for browsers that do not handle frames. The noframes element goes inside the frameset element.

Note: If a browser handles frames, it will not display the text in the noframes element.

Important: If you add a <noframes> tag to a frameset, you will have to enclose the text in <body></body> tags!

Example

```
<html>

<frameset cols="25%,50%,25%">

  <frame src="frame_a.htm">

  <frame src="frame_b.htm">

  <frame src="frame_c.htm">

<noframes>

<body>Your browser does not handle frames!</body>

</noframes>

</frameset>
```

</html>

The Use of <NOFRAMES>

The <NOFRAMES> element is used to enclose the HTML and text that should be displayed when a browser that does not support frames accesses the Web page. The <NOFRAMES> element should be found only within the <FRAMESET> element. The contents of the <NOFRAMES> element should be correct HTML, potentially including the <BODY> element, which can be used as an alternative form for browsers that don't support frames.

Floating Frames

A floating frame introduced by Microsoft, has been incorporated in the HTML 4 standard. The idea of the floating frame is to create an inline framed region, or window, that acts similarly to any other embedded object, insofar as text can be flowed around it. An inline frame is defined by the <IFRAME> element and may occur anywhere within the <BODY> element of an HTML document. Compare this to the <FRAME> element, which should occur only within the <FRAMESET> element and the <FRAMESET> element should preclude the <BODY> element. The major attributes to set for the <IFRAME> element include SRC, HEIGHT, and WIDTH. The SRC is set to the URL of the file to load, while the HEIGHT and WIDTH are set to either the pixel or percentage value of the screen that the floating frame region should consume. Like an element, floating frames should support ALIGN, HSPACE, and VSPACE attributes for basic positioning within the flow of text. Note that, unlike the <FRAME> element, the <IFRAME> element comes with a close tag. <IFRAME> and </IFRAME> should contain any HTML markup code and text that is supposed to be displayed in browsers that don't support floating frames.

Using Frames

One of the biggest problems with frames is that they initially were used simply because they existed/. Framed documents can provide considerable benefit, but at a price. A potential benefit of frames is that they allow content to be fixed onscreen. One frame may contain a table of contents, while the other frame contains the actual information. Keeping the table of contents onscreen provides a convenient way to navigate the body of information. Furthermore, if one

frame has fixed navigation, the user may perceive the Web interface to be more responsive, because only part of the screen needs to update between selections. The primary benefit of frames is to present two or more things simultaneously, but this extra window of information has its costs.

Frame Problems

The problems with frames are numerous, including design issues, navigation confusion, bookmarking problems, loss of URL context, and printing issues. The only potential design issue is the possibility that a framed document may sacrifice valuable screen real estate because of scroll bars, which could pose trouble for people with lower-resolution monitors. The only way to get around this problem is to limit the number of frames used on a page. Additional navigational problems include loss of context, because the URL of the document tends not to change, which accounts for why bookmarking does not work as expected. Many people use URLs as a way to orient themselves at a site. Frames give up this clue to location.

Frame Tags

Tag	Description
<code><frameset></code>	Defines a set of frames
<code><frame></code>	Defines a sub window (a frame)
<code><noframes></code>	Defines a noframe section for browsers that do not handle frames
<code><iframe></code>	Defines an inline sub window (frame)

8.6. Summary

1. In HTML, images are defined with the `` tag.
2. The `` tag is empty, which means that it contains attributes only and it has no closing tag.

3. Tables are defined with the `<table>` tag. A table is divided into rows (with the `<tr>` tag), and each row is divided into data cells (with the `<td>` tag).
4. A frame is an independent scrolling region, or window, of a Web page. Every Web page may be divided into many individual frames, which can even be nested within other frames.
5. Fixed screen sizes limit how many frames can really be used.

8.7. SAQ

1. How to insert image into the web page.
2. Design time table of course in tabular fashion using `<table>` tag.
3. What is the use of `<p>` tag?
4. Explain how to create hyper links of the web page.
5. Design a home page of the college using frames.

Unit – IX DHTML and CSS

9.0. Structure

9.1. Objective

9.2. DHTML – CSS

9.3. Inline style sheets

9.4. Embedding a style sheet

9.5. Linking a External style sheet

9.6. Java Script programming

9.6.1. Arrays in Java Script

9.6.2. Functions in Java Script

9.6.3. Java Script Events

9.7. Summary

9.8. SAQ

9.1. Objective

The aim of this chapter is to understand about DHTML, CSS and Java Script.

9.2. DHTML (Dynamic Hyper Text Markup Language)

Dynamic Hyper Text Markup Language (DHTML) is a combination of Web development technologies used to create dynamically changing websites. Web pages may include animation, dynamic menus and text effects. The technologies used include a combination of HTML, Java Script or VB Script, CSS and the Document Object Model(DOM).

Designed to enhance a Web user's experience, DHTML includes the following features:

- Dynamic content, which allows the user to dynamically change Web page content
- Dynamic positioning of Web page elements

- Dynamic style, which allows the user to change the Web page's color, font, size or content.

CSS(Cascading Style Sheets)

What is CSS?

- 1) CSS stands for Cascading Style Sheets
- 2) Styles define how to display HTML elements
- 3) Styles are normally stored in Style Sheets
- 4) Styles were added to HTML 4.0 to solve a problem
- 5) External Style Sheets can save you a lot of work
- 6) External Style Sheets are stored in CSS files
- 7) Multiple style definitions will cascade into one

Styles Solve a Common Problem

HTML tags were originally designed to define the content of a document. They were supposed to say "This is a header", "This is a paragraph", "This is a table", by using tags like `<h1>`, `<p>`, `<table>`, and so on. The layout of the document was supposed to be taken care of by the browser, without using any formatting tags.

As the two major browsers - Netscape and Internet Explorer - continued to add new HTML tags and attributes (like the `` tag and the color attribute) to the original HTML specification, it became more and more difficult to create Web sites where the content of HTML documents was clearly separated from the document's presentation layout.

The Rise of Style Sheets

Basically, style sheets separate the structure of a document from its presentation. Dividing layout and presentation has many theoretical benefits, most importantly, it can provide for flexible documents that display equally well across many types of output devices. As early as 1993, people have been interested in adding more layout control to HTML. Many approaches have been discussed and many continue to be used. Because of the theoretical benefits of style sheets, they have been the favorite solution of the standards bodies. More than one type of style sheet

exists. Many industry pundits support a type of sheet known as Document Style Semantics and Specification Language (DSSSL), developed by the SGML community. The most recent addition is Extensible Style Language (XSL), an industry proposal based on DSSSL that uses Extensible Markup Language (XML) syntax.

Style Sheet Basics

CSS1 style sheets rely on an underlying markup structure, such as HTML. They are not a replacement for HTML. Without a binding to an element, a style really doesn't mean anything. The purpose of a style sheet is to create a presentation for a particular element or set of elements. Binding an element to a style specification consists of an element, followed by its associated style information within curly braces:

```
Element      {style specification}
```

Suppose that you want to bind a style rule to the <H1> element so that a 28-point Impact font is always used. The following rule would result in the desired display:

```
H1      {font-family: Impact;
         font-size: 28pt}
```

In general, a style specification or style sheet is simply a collection of rules. These rules include a selector, an HTML element, a CLASS name, or an ID value, which is bound to a style property such as font-family, followed by a colon and the values for that style property. Multiple style rules may be included in a style specification by separating the rules with semicolons. You can also use many shorthand notations and grouping rules that are available. Style sheets alone do nothing. First, you must bind the rule to a tag or class of HTML objects. Currently, more than 50 properties are specified under CSS1 that affect the presentation of an HTML document, and more than 50 more properties are defined under CSS2. Unfortunately, not all of them are supported consistently across the major browsers. Even worse, most of the newer style properties defined by CSS2 are not supported by any browser.

Adding Style to a Document

Style information may be included in an HTML document in any one of three basic ways:

1. Use an outside style sheet, either by importing it or by linking to it.
2. Embed a document-wide style in the <HEAD> element of the document.
3. Provide an inline style exactly where the style needs to be applied.

Each of these style sheet approaches has its own pros and cons, as listed in the following table:

Comparison of Style Sheet Approaches:

	External Style Sheets	Document-Wide Style	Inline Style
Pros	Can set style for many documents with one style sheet	Can control style for a document in one place. No additional download time for style information.	Can control style to a single character instance. Overrides any external or document styles.
Cons	Require extra download time for the style sheet, which may delay page rendering	Need to reapply style information for other documents	Need to reapply style information throughout the document and outside documents. Bound too closely to HTML, difficult to update.

In brief, Cascading style sheets provide better control over the look and feel of the Web pages. Style sheets aren't just useful for making attractive pages. By dividing structure and style, they make documents simpler, and easier to manipulate. While style sheets provide a great deal of

flexibility in creating pages, they are not fully implemented yet in today's browsers. Some inconsistencies exist between implementations. When used in a non-obtrusive manner, style sheets are a great way to improve the layout of pages, without locking into a proprietary solution.

The following is an example that implements CSS.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
  <title>My first styled page</title>
  <style type="text/css">
< body> {
  padding-left: 1em;
  font-family: Georgia, "Times New Roman",
    Times, serif;
  color: purple;
  background-color: #d8da3d }
ul.navbar {
  list-style-type: none;
  padding: 0;
  margin: 0;
  position: absolute;
  top: 2em;
  left: 1em;
  width: 9em }
h1 {
  font-family: Helvetica, Geneva, Arial,
    SunSans-Regular, sans-serif }
ul.navbar li {
  background: white;
  margin: 0.5em 0;
  padding: 0.3em;
  border-right: 1em solid black }
ul.navbar a {
  text-decoration: none }
a:link {
  color: blue }
a:visited {
  color: purple }
address {
  margin-top: 1em;
  padding-top: 1em;
  border-top: thin dotted }
</style>
</head>
```

```
</body>
</html>
```

9.3. InLine style sheets

Inline style sheets is a term that refers to style sheet information being applied to the current element. By this, I mean that instead of defining the style once, then applying the style against all instances of an element (say the<p> tag), you only apply the style to the instance you want the style to apply to. Actually, it's not really a style sheet as such, so a more accurate term would be inline styles.

Consider the following example

```
<p style =”color:#ff9900”> this text has been styled using inline style sheets.</p>
```

Example:

```
<!DOCTYPE html>
<html>
<body>

<h1 style="color:blue;">This is a Blue Heading</h1>

</body>
</html>
```

9.4. Embedding a style sheet

Embedded CSS is also know as Internal style sheets. This can be used when a single HTML document must be styled uniquely. The CSS rule set should be within the HTML file in the head section i.e the CSS is embedded within the HTML file.

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>Internal CSS</title>
<style>
.main {
text-align:center;
}
.GFG {
color:#009900;
```

```

        font-size:50px;
        font-weight:bold;
    }
    .geeks {
        font-style:bold;
        font-size:20px;
    }
</style>
</head>
<body>
    <div class = "main">
        <div class ="GFG">RASTRIYA SANSKRIT KIDYAPEETA</div>

        <div class ="geeks">
            A computer science portal for VIDYAPEETA students
        </div>
    </div>
</body>
</html>

```

9.5. Linking an External style sheet

External CSS: External CSS contains separate CSS file which contains only style property with the help of tag attributes (For example class, id, heading, ... etc). CSS property written in a separate file with .css extension and should be linked to the HTML document using **link** tag. This means that for each element, style can be set only once and that will be applied across web pages. **Example:** The file given below contains CSS property. This file save with .css extension. Let us consider an example

```

Body{
    Background-color: darklategrey;
    Color: azure;
    font-size: 1.1em;
}
H1 {
    Color:coral;
}
#intro{
    Font-size:1.3em;
}

```



```
}  
.colorful{  
    Color:orange;  
}
```

Link to external style sheet

Below is the HTML file that is making use of the created external style sheet

- **link** tag is used to link the external style sheet with the html webpage.
- **href** attribute is used to specify the location of the external style sheet file.

```
<!DOCTYPE html>  
<html>  
  <head>  
    <link rel="stylesheet" href="geeks.css"/>  
  </head>  
  
  <body>  
    <div class = "main">  
      <div class="GFG">GeeksForGeeks</div>  
      <div id ="geeks">  
        A computer science portal for geeks  
      </div>  
    </div>  
  </body>  
</html>
```

9.6. Java Script programming

Javascript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities. JavaScript was first known as LiveScript, but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java. JavaScript made its first appearance in Netscape 2.0 in 1995 with the name LiveScript. The general-purpose core of the

language has been embedded in Netscape, Internet Explorer, and other web browsers. The ECMA-262 Specification defined a standard version of the core JavaScript language. JavaScript is a lightweight, interpreted programming language.

- Designed for creating network-centric applications.
- Complementary to and integrated with Java.
- Complementary to and integrated with HTML.
- Open and cross-platform.

Advantages of Java Script

The merits of using JavaScript are:

Less server interaction: You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.

Immediate feedback to the visitors: They don't have to wait for a page reload to see if they have forgotten to enter something.

Increased interactivity: You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.

Richer interfaces: You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

Syntax

Java Script can be implemented using Java Script statements that are placed within the `<script>...</script>` HTML tags in a web page.

You can place the `<script>` tags, containing java script, any where within the web page, but normally recommended script tag is kept within the `<head>` tags.

The `<script>` tag alters the browser program to start interpreting all the text between these tags as a script. Consider the following simple syntax for java script

```
<script>  
  Javascript code  
</script>
```

The script tag takes two important attributes:

Language: This attribute specifies what scripting language you are using. Typically, its value will be javascript. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.

Type: This attribute is what is now recommended to indicate the scriptin language in use and its value should be set to "text/javascript".

Then the javascript syntax will look as follows

```
<script language="javascript" type="text/javascript">  
Javascript code  
</script>
```

Comments in JavaScript

Java Script supports both C-style and C++-style comments. Any text between a // and the end of a line is treated as a comment and is ignored by JavaScript. Any text between the characters /* and */ is treated as a comment. This may span multiple lines. JavaScript also recognizes the HTML comment opening sequence is not recognized by JavaScript so it should be written as

```
</script>
```

Note: where var is key word

Storing a value in a variable is called variable initialization. You can do variable initialization at the time of variable creation or at a later point in time when you need that variable. For instance, you might create a variable named money and assign the value 2000.50 to it later. For another variable, you can assign a value at the time of initialization as follows.

```
<script type="text/javascript">
```

```
<!
```

```
Var slno=01;
```

```
Var name="vidyapeeta";
```

```
//-->
```

```
</script>
```

Conditional and control statements in Java Script

Conditional Statements

Very often when you write code, you want to perform different actions for different decisions.

You can use conditional statements in your code to do this.

In JavaScript we have the following conditional statements:

- Use if to specify a block of code to be executed, if a specified condition is true
- Use else to specify a block of code to be executed, if the same condition is false
- Use else if to specify a new condition to test, if the first condition is false
- Use switch to specify many alternative blocks of code to be executed

The if Statement

Use the “if” statement to specify a block of JavaScript code to be executed if a condition is true.

Syntax

```
if(condition) {  
    // block of code to be executed if the condition is true  
}
```

consider the following example for if statement

```
<!DOCTYPE html>
<html>
<body>

<p>Display "Good day!" if the hour is less than 18:00:</p>

<p id="demo">Good Evening!</p>

<script>
if (new Date().getHours() < 18) {
  document.getElementById("demo").innerHTML = "Good day!";
}
</script>

</body>
</html>
```

The else Statement

Use the else statement to specify a block of code to be executed if the condition is false.

```
if (condition) {
  // block of code to be executed if the condition is true
} else {
  // block of code to be executed if the condition is false
}

<!DOCTYPE html>
<html>
<body>

<p>Click the button to display a time-based greeting:</p>

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

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

<script>
function myFunction() {
  var hour = new Date().getHours();
  var greeting;
  if (hour < 18) {
    greeting = "Good day";
```

```

    } else {
      greeting = "Good evening";
    }
    document.getElementById("demo").innerHTML = greeting;
  }
</script>
</body>
</html>

```

The JavaScript Switch Statement

Use the switch statement to select one of many code blocks to be executed.

Syntax

```

switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}

```

JavaScript Looping

Looping

Very often when you write code, you want the same block of code to run a number of times. You can use looping statements in your code to do this.

In JavaScript we have the following looping statements:

- **while** - loops through a block of code while a condition is true
- **do...while** - loops through a block of code once, and then repeats the loop while a condition is true
- **for** - run statements a specified number of times

while

The while statement will execute a block of code while a condition is true..

Syntax

```

While (condition)
{
  Code to be executed
}

```

do...while

The do...while statement will execute a block of code once, and then it will repeat the loop while a condition is true

Syntax

Do

```
{  
Code to be executed  
}while(condition)
```

for

The for statement will execute a block of code a specified number of times

Syntax

```
for(initialization; condition; increment/decrement)  
{  
Code to be executed  
}
```

Consider the following example for loop control statement

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<p>Click the button to loop through a block of code five times.</p>
```

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

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

```
<script>
```

```
function myFunction() {
```

```
var x = "", i;
```

```
for (i=0; i<5; i++) {
```

```
    x = x + "The number is " + i + "<br>";
```

```
}
```

```
document.getElementById("demo").innerHTML = x;
```

```
}
```

```
</script>
```

```
</body>
```

```
</html>
```

Sr.No.	Statement	Description
1.	switch case	A block of statements in which execution of code depends upon different cases. The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a default condition will be used.
2.	If else	The if statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.
3.	While	The purpose of a while loop is to execute a statement or code block repeatedly as long as expression is true. Once expression becomes false, the loop will be exited.
4.	do while	Block of statements that are executed at least once and continues to be executed while condition is true.
5.	for	Same as while but initialization, condition and increment/decrement is done in the same line.
6.	for in	This loop is used to loop through an object's properties.
7.	continue	The continue statement tells the interpreter to immediately start the next iteration of the loop and skip remaining code block.
8.	break	The break statement is used to exit a loop early, breaking out of the enclosing curly braces.
9.	function	A function is a group of reusable code which can be called anywhere in your programme. The keyword function is used to declare a function.
10.	return	Return statement is used to return a value from a function.
11.	var	Used to declare a variable.
12.	try	A block of statements on which error handling is implemented.
13.	catch	A block of statements that are executed when an error occur.
14.	throw	Used to throw an error.

9.6.1. Arrays in Java Script

JavaScript array is an object that represents a collection of similar type of elements.

There are 3 ways to construct array in JavaScript

1. By array literal
2. By creating instance of Array directly (using new keyword)
3. By using an Array constructor (using new keyword)

1) JavaScript array literal

The syntax of creating array using array literal is given below:

```
Var arrayname=[value1,value2,...valueN];
```

Values are contained inside [] are separated by , (comma). Let us consider example on array in javascript.

```
<html>
<body>
<script>
var emp=["Vishnu","Chutki","Srivalli"];
for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br/>");
}
</script>
</body>
</html>
```

Output:

```
Vishnu
Chutki
Srivalli
```

2) JavaScript Array directly (new keyword)

The syntax of creating array directly is given below:

```
Var arrayname=new Array( );
```

Here in the above syntax new keyword is used to create instance of array.

```
<html>
```

```

<body>
<script>
var i;
var emp = new Array();
emp[0] = "Vishnu";
emp[1] = "Chutki";
emp[2] = "Srivalli";

for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br>");
}
</script>
</body>
</html>

```

Output

Vishnu
Chutki
Srivalli

3) JavaScript array constructor (new keyword)

Here, you need to create instance of array by passing arguments in constructor so that we don't have to provide value explicitly.

The example of creating object by array constructor is given below.

```

<html>
<body>
<script>
var emp=new Array("Jai","Vijay","Smith");
for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br>");
}
</script>

```

</body>

</html>

Output

Jai

Vijay

Smith

JavaScript Array Methods

Now let us see list of java script array methods and their description in detail

Methods	Description
<u>concat()</u>	It returns a new array object that contains two or more merged arrays.
<u>copywithin()</u>	It copies the part of the given array with its own elements and returns the modified array.
<u>every()</u>	It determines whether all the elements of an array are satisfying the provided function conditions.
<u>fill()</u>	It fills elements into an array with static values.
<u>filter()</u>	It returns the new array containing the elements that pass the provided function conditions.
<u>find()</u>	It returns the value of the first element in the given array that satisfies the

	specified condition.
<u>findIndex()</u>	It returns the index value of the first element in the given array that satisfies the specified condition.
<u>forEach()</u>	It invokes the provided function once for each element of an array.
<u>includes()</u>	It checks whether the given array contains the specified element.
<u>indexOf()</u>	It searches the specified element in the given array and returns the index of the first match.
<u>join()</u>	It joins the elements of an array as a string.
<u>lastIndexOf()</u>	It searches the specified element in the given array and returns the index of the last match.
<u>map()</u>	It calls the specified function for every array element and returns the new array
<u>pop()</u>	It removes and returns the last element of an array.
<u>push()</u>	It adds one or more elements to the end of an array.

<u>reverse()</u>	It reverses the elements of given array.
<u>shift()</u>	It removes and returns the first element of an array.
<u>slice()</u>	It returns a new array containing the copy of the part of the given array.
<u>sort()</u>	It returns the element of the given array in a sorted order.
<u>splice()</u>	It add/remove elements to/from the given array.
<u>unshift()</u>	It adds one or more elements in the beginning of the given array.

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:

Example:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>Arrays use numbers to access its elements.</p>

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

<script>
var person = ["John", "Doe", 46];
document.getElementById("demo").innerHTML = person[0];
```

```
</script>
```

```
</body>
```

```
</html>
```

Objects use **names** to access its "members". In this example, "person.firstName" returns John:

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h2>JavaScript Objects</h2>
```

```
<p>JavaScript uses names to access object properties.</p>
```

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

```
<script>
```

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

```
document.getElementById("demo").innerHTML = person["firstName"];
```

```
</script>
```

```
</body>
```

```
</html>
```

.00009.6.2. Functions in Java Script

A function is a set of statements that take inputs, do some specific computation and produces output. Basically, a function is a set of statements that performs some tasks or does some computation and then returns the result to the user.

The idea is to put some commonly or repeatedly done task together and make a function so that instead of writing the same code again and again for different inputs, we can call that function.

Like other programming languages, JavaScript also supports the use of functions. You must already have seen some commonly used functions in JavaScript like alert(), this is a built-in function in JavaScript. But JavaScript allows us to create user-defined functions also.

We can create functions in JavaScript using the keyword *function*. The basic syntax to create a function in JavaScript is shown below.

Syntax

```
function functionname(parameter1, paramater2,...)
```

```
{
```

```
    //function body
```

```
}
```

To create a function in JavaScript, we have to first use the keyword *function*, separated by name of function and parameters within parenthesis. The part of function inside the curly braces {} is the body of the function.

Function Definition

Before, using a user-defined function in JavaScript we have to create one. We can use the above syntax to create a function in JavaScript. Function definition are sometimes also termed as function declaration or function statement.

Below are the rules for creating a function in JavaScript:

- Every function should begin with the keyword *function* followed by,
- A user defined function name which should be unique,
- A list of parameters enclosed within parenthesis and separated by commas,
- A list of statement composing the body of the function enclosed within curly braces {}.

Example:

```
function calcAddition(number1, number2)
{
    return number1 + number2;
}
```

In the above example, we have created a function named `calcAddition`, this function accepts two numbers as parameters and returns the addition of these two numbers.

Function Parameters

Till now we have heard a lot about function parameters but haven't discussed them in details. Parameters are additional information passed to a function. For example, in the above example, the task of the function *calcAddition* is to calculate addition of two numbers. These two numbers on which we want to perform the addition operation are passed to this function as parameters. The parameters are passed to the function within parentheses after the function name and separated by commas. A function in JavaScript can have any number of parameters and also at the same time a function in JavaScript can not have a single parameter.

Calling Functions: After defining a function, the next step is to call them to make use of the function. We can call a function by using the function name separated by the value of parameters

enclosed between parenthesis and a semicolon at the end. Below syntax shows how to call functions in JavaScript:

Function (value1, value2,...)

Consider the following program that illustrate working of functions in java script

```
<script type = "text/javascript">

// Function definition
function welcomeMsg(name) {
    document.write("Hello " + name + " welcome to Rastriya Sanskrit viyapeeta");
}

// creating a variable
var nameVal = "Admin";

// calling the function
welcomeMsg(nameVal);

</script>
```

Output:

Hello Admin welcome to Rastriya Sanskrit vidyapeeta

9.6.3. Java Script Events

Event

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.

When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.

Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.

Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events which can trigger JavaScript Code.

```
<!DOCTYPE html>
<html>
<body>
<script>
function sayHello() {
    alert("Onclick event.")
}
```



```

}
</script>
<p>Click Say Hello button and see result.</p>
<form>
  <input type="button" onclick="sayHello()" value="Say Hello" />
</form>
</body>
</html>

```

Below are the different types of EVENTS used in java script

Input Events

Attribute	Description
Offline	Triggers when the document goes offline
Onabort	Triggers on an abort event
onafterprint	Triggers after the document is printed
onbeforeonload	Triggers before the document loads
onbeforeprint	Triggers before the document is printed
onblur	Triggers when the window loses focus
oncanplay	Triggers when media can start play, but might has to stop for buffering
oncanplaythrough	Triggers when media can be played to the end, without stopping for buffering
onchange	Triggers when an element changes
onclick	Triggers on a mouse click
oncontextmenu	Triggers when a context menu is triggered
ondblclick	Triggers on a mouse double-click
ondrag	Triggers when an element is dragged
ondragend	Triggers at the end of a drag operation
ondragenter	Triggers when an element has been dragged to a valid drop target
ondragleave	Triggers when an element is being dragged over a valid drop target
ondragover	Triggers at the start of a drag operation
ondragstart	Triggers at the start of a drag operation
ondrop	Triggers when dragged element is being dropped
ondurationchange	Triggers when the length of the media is changed
onemptied	Triggers when a media resource element suddenly becomes empty.
onended	Triggers when media has reach the end

onerror	Triggers when an error occur
onfocus	Triggers when the window gets focus
onformchange	Triggers when a form changes
onforminput	Triggers when a form gets user input
onhaschange	Triggers when the document has change
oninput	Triggers when an element gets user input
oninvalid	Triggers when an element is invalid
onkeydown	Triggers when a key is pressed
onkeypress	Triggers when a key is pressed and released
onkeyup	Triggers when a key is released
onload	Triggers when the document loads
onloadeddata	Triggers when media data is loaded
onloadedmetadata	Triggers when the duration and other media data of a media element is loaded
onloadstart	Triggers when the browser starts to load the media data
onmessage	Triggers when the message is triggered
onmousedown	Triggers when a mouse button is pressed
onmousemove	Triggers when the mouse pointer moves
onmouseout	Triggers when the mouse pointer moves out of an element
onmouseover	Triggers when the mouse pointer moves over an element
onmouseup	Triggers when a mouse button is released
onmousewheel	Triggers when the mouse wheel is being rotated
onoffline	Triggers when the document goes offline
onoine	Triggers when the document comes online
ononline	Triggers when the document comes online
onpagehide	Triggers when the window is hidden
onpageshow	Triggers when the window becomes visible
onpause	Triggers when media data is paused
onplay	Triggers when media data is going to start playing
onplaying	Triggers when media data has start playing
onpopstate	Triggers when the window's history changes
onprogress	Triggers when the browser is fetching the media data
onratechange	Triggers when the media data's playing rate has changed

onreadystatechange	Triggers when the ready-state changes
onredo	Triggers when the document performs a redo
onresize	Triggers when the window is resized
onscroll	Triggers when an element's scrollbar is being scrolled
onseeked	Triggers when a media element's seeking attribute is no longer true, and the seeking has ended
onseeking	Triggers when a media element's seeking attribute is true, and the seeking has begun
onselect	Triggers when an element is selected
onstalled	Triggers when there is an error in fetching media data
onstorage	Triggers when a document loads
onsubmit	Triggers when a form is submitted
onsuspend	Triggers when the browser has been fetching media data, but stopped before the entire media file was fetched
ontimeupdate	Triggers when media changes its playing position
onundo	Triggers when a document performs an undo
onunload	Triggers when the user leaves the document
onvolumechange	Triggers when media changes the volume, also when volume is set to "mute"
onwaiting	Triggers when media has stopped playing, but is expected to resume

9.6.4. Summary

- Dynamic HyerText Markup Language (**DHTML**) is a combination of Web development technologies used to create dynamically changing websites.
- Web pages may include animation, dynamic menus and text effects. The technologies used include a combination of HTML, JavaScript or VB Script, CSS and the document object model (DOM).
- **CSS** is a language that describes the style of an HTML document. **CSS** describes how HTML elements should be displayed
- Javascript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages.

- Java Script can be implemented using Java Script statements that are placed within the `<script>...</script>` HTML tags in a web page.
- An **array** can hold many values under a single name, and you can access the values by referring to an index number. Creating an **Array**. Using the **JavaScript** Keyword `new`. Access the Elements of an **Array**. Changing an **Array** Element. Access the Full **Array**. **Arrays** are Objects.

9.6.5. SAQ

1. What is DHTML? Explain various features of DHTML.
2. What is CSS? How to integrate CSS to DHTML.
3. Discuss about Style sheets.
4. Design a Web page to insert a table dynamically.

Unit – X Working with forms an controls

10.0. Structure

10.1. Objective

10.2. Working with Forms and Controls

10.2.1. Forms

10.2.2. Input element

10.2.3. Select element

10.2.4. Text area elements

10.3. Summary

10.4. SAQ

10.1. Objective

From this unit students can understand what are form, how to generate online application form.

10.2. Working with Forms and Controls

HTML forms introduce the concept of interaction between the web user and the World Wide Web, in particular the Web server or Host.

10.2.1. Forms

A form is exactly what it says it is, a form that the user fills-in. It can have a variety of different types of input fields that you specify when you design the form (e.g. name, address, age, etc.) and the user fills-in prior to submitting the form. The only problem with HTML forms is that they are not much use by themselves because a traditional HTML form will only work in conjunction with a specially written server-side CGI program or script.

There are two exceptions to this rule and the first exception is the 'mailto' protocol (Mailto is an Internet protocol (similar to 'http:', 'ftp:', 'news:', etc) that provides a gateway to the Internet email system). Not supported in Microsoft browsers, which enables the form's contents to be mailed to an email address you specify. Using mailto is a really great way to practice developing

HTML forms without having to get involved in CGI programming or scripting at the same time, but you will have to use a Netscape browser because Microsoft browsers don't fully support the mailto protocol. The other exception is forms where the contents are sent to a JavaScript function for processing. This subject is beyond the scope of this tutorial and will be dealt with more fully in the JavaScript section of Web-Wise-Wizard.

Many other uses for forms and form elements have developed since the introduction of JavaScript extensions to HTML forms and again these will be discussed more fully in the JavaScript section. In the meantime we explain the form tag and its attributes and demonstrate each of the available form elements without the JavaScript extensions.

Syntax

```
<form action="action name" method="method type" enctype="encrypted code">
```

```
-----
```

```
-----
```

```
</form>
```

Where..

Action

This is where the Web browser sends the form's contents when the user submits the form. It can be a CGI program (script) that processes the data and this can be on any Web server. Alternatively, if you are using a Netscape Web browser it could be an email address.

- **cgi program** e.g. action="http://www.web-wise-wizard.com/cgi-bin/orders.cgi"
- **mailto:** e.g. action="mailto:gil@web-wise-wizard.com" (Netscape browsers only)

cgi program A program or script that runs as a process on a Web server and can access the users CGI variables. CGI programs and scripts can be written in a variety of languages which include 'Perl', 'C/C++', etc.

mailto: An Internet protocol (similar to 'http:', 'ftp:', 'news:', etc) that provides a gateway to the Internet email system. Not supported in Microsoft browsers.

Method

Specifies which HTTP method will be used to pass the form's contents to the CGI interface on the Web server.

- **get** (default) The form's contents are placed in a CGI variable called 'QUERY_STRING' on the Web server and are accessed by a CGI program (or script) using that variable.
- **post** The form's contents are sent to stdin (a 'C' type input stream) on the Web server and the length of the input stream is set in a CGI variable called 'CONTENT_LENGTH'. The CGI program (or script) accesses the contents via stdin.

notes ...

http Hyper-Text Transfer Protocol: A pre-defined protocol to enable communication between a Web browser and a Web server on the World Wide Web.

cgi Common Gateway Interface <www>: A standard for running external programs from a World-Wide Web HTTP server. CGI specifies how to pass arguments to the executing program as part of the HTTP request. It also defines a set of environment variables. Commonly, the program will generate some HTML which will be passed back to the browser but it can also request URL redirection.

Form Elements Form elements could be considered the building blocks of a form and I have even heard them described as toys in a toy box. Here we list and demonstrate all elements that you could use when designing and creating an HTML form.

Enctype

Specifies how the form's contents should be encrypted.

- **application/x-www-form-urlencoded** (default) If you are submitting your form using the mailto protocol then our email management program will normally convert this encryption back to plain text in name/value pairs.

- **multipart/form-data** This is normally used in conjunction with the 'file' element (i.e. input type="file").
- **text/plain** Specifies that the form's contents should not be encrypted. Contents sent as plain text in name/value pairs.

Target

Commonly, a CGI program or script will generate an HTML response which it will pass back to the Web browser. The target attribute specifies where that response should be sent. Possible options are the four HTML magic target names or a names window or frame.

- **_blank** Display the response in a newly opened blank browser window.
- **_self** (the default) Display the response in the current frame or window.
- **_parent** Display the response in the parent of the current frame or window.
- **_top** Display the response as the top document in the current window.
- **windowName or frameName** Display the response in a named frame or window, or if no match is found for the name then display the response in a newly opened browser window using the specified name for the window.

Example

```
<form action="mailto:you@youremail" method="get" enctype="application/x-www-form-
urlencoded">
<!-- FORM ELEMENTS ENCLOSED BETWEEN FORM START AND FORM END TAGS -->
</form>
```

The Form's Action Attribute and the Submit Button

When the user clicks on the "Submit" button, the content of the form is sent to the server. The form's action attribute defines the name of the file to send the content to. The file defined in the action attribute usually does something with the received input

```
<form name="input" action="html_form_submit.asp"
method="get">
```


Username:

```
<input type="text" name="user">  
<input type="submit" value="Submit">  
</form>
```

How it looks in a browser:

Username:

If you type some characters in the text field above, and click the "Submit" button, the browser will send your input to a page called "html_form_submit.asp". The page will show you the received input.

10.2.2. Input elements

The most used form tag is the `<input>` tag. The type of input is specified with the `type` attribute. The most commonly used input types are explained below.

Text Fields

Text fields are used when you want the user to type letters, numbers, etc. in a form

```
<form>  
First name:  
<input type="text" name="firstname">  
<br>  
Last name:  
<input type="text" name="lastname">  
</form>
```

How it looks in a browser:

First

name:

Last name:

Note that the form itself is not visible. Also note that in most browsers, the width of the text field is 20 characters by default.

Radio Buttons

Radio Buttons are used when you want the user to select one of a limited number of choices.

```
<form>
<input type="radio" name="sex" value="male"> Male
<br>
<input type="radio" name="sex" value="female"> Female
</form>
```

How it looks in a browser:

- Male
 Female

Note that only one option can be chosen.

Checkboxes

Checkboxes are used when you want the user to select one or more options of a limited number of choices.

```
<form>
I have a bike:
<input type="checkbox" name="vehicle" value="Bike">
<br>
I have a car:
```

```
<input type="checkbox" name="vehicle" value="Car">
```

```
<br>
```

I have an airplane:

```
<input type="checkbox" name="vehicle" value="Airplane">
```

```
</form>
```

How it looks in a browser:

I have a bike:

I have a car:

I have an airplane:

102.3. Select elements

HTML `<select>` tag is used to create drop down list of options, which appears when clicking on form element and allows the user to choose one of the options.

The `<option>` tag is used to define the possible options to choose from. The tag is put into `<select>` tag.

The first option from the options' list is selected by default. To change predefined option selected attribute is used.

The `<optgroup>` tag is used to group several options into one group. The content of `<optgroup>` looks like heading in bold.

The look of the list depends on `size` attribute, which defines the height of the list. The width of the list depends on the length of the text inside `<option>` tag. The width can be regulated with CSS styles as well.

NOTE: if you need to send the data to the server or refer to the list with scripts, `<select>` tag should be put inside `<form>` tag.

Syntax

The content is written between opening (<select>) and closing (</select>) tags. Closing tag is mandatory.

Example

```
<!DOCTYPE html>
<html>
  <head>
    <title>Title of the document</title>
  </head>
  <body>
    <form>
      <select>
        <option value="books">Books</option>
        <option value="html">HTML</option>
        <option value="css">CSS</option>
        <option value="php">PHP</option>
        <option value="js">JavaScript</option>
      </select>
    </form>
  </body>
</html>
```

Output

Try for your self

10.2.4. Text area

The <textarea> tag defines a multi-line **text** input control. A **text area** can hold an unlimited number of characters, and the **text** renders in a fixed-width font (usually Courier). The size of a **text area** can be specified by the cols and rows attributes, or even better; through CSS' height and width properties.

Example

```
<!DOCTYPE html>
<html>
<body>
```

```
<textarea rows="4" cols="50">
```

At RSVP.com you will learn how to make a website. We offer tutorials in our DDE

</textarea>

</body>

</html>

Output

At RSVP.com you will learn how to make a website. We offer tutorials in our DDE

Form Tags

Tag	Description
<form>	Defines a form for user input
<input>	Defines an input field
<textarea>	Defines a text-area (a multi-line text input control)
<label>	Defines a label to a control
<fieldset>	Defines a fieldset
<legend>	Defines a caption for a fieldset
<select>	Defines a selectable list (a drop-down box)
<optgroup>	Defines an option group
<option>	Defines an option in the drop-down box
<button>	Defines a push button
<isindex>	Deprecated. Use <input> instead

10.3. Summary

1. Forms provide a basic interface for adding interactivity to a web site. Html supports traditional graphical user interface controls such as check boxes, radio buttons, pull-down menus, scrolled lists, multi and single line text areas and buttons.
2. The most used form tag is the <input> tag. The type of input is specified with the type attribute. The most commonly used input types.
3. Text fields are used when you want the user to type letters, numbers, etc. in a form
4. Radio Buttons are used when you want the user to select one of a limited number of choices.
5. Checkboxes are used when you want the user to select one or more options of a limited number of choices.

10.4. SAQ

1. Write short notes on forms?
2. Design a web page which is similar to an application form for taking admission into sastrilaya through DDE mode?