# LECTURE NOTE

# on

# PROGRAMMING IN "C"

By

**Mr. Vikas Singh**
**(Msc Computer Science, Ph.D. Appeared)**

**Founder : Coding Seekho(VS Programming Institute)**

# INTRODUCTION TO JAVA PROGRAMMING

**Objectives:**

This subject aims to introduce students to the Java programming language. Upon successful completion of this subject, students should be able to create Java programs that leverage the object-oriented features of the Java language, such as encapsulation, inheritance and polymorphism; use data types, arrays and other data collections; implement error-handling techniques using exception handling, create and event-driven GUI using Swing components.

**UNIT-I**

**OOP Concepts:-**Data abstraction, encapsulation, inheritance, Benefits of Inheritance, Polymorphism, classes and objects, Procedural and object oriented programming paradigms.

**Java Programming-** History of Java, comments, Data types, Variables, Constants, Scope and Lifetime of variables, Operators, Operator Hierarchy, Expressions, Type conversion and casting, Enumerated types, Control flow- block scope, conditional statements, loops, break and continue statements, simple java stand alone programs, arrays, console input and output, formatting output, constructors, methods, parameter passing, static fields and methods, access control, this reference, overloading methods and constructors, recursion, garbage collection, building strings, exploring string class.

**UNIT – II**

**Inheritance –** Inheritance hierarchies super and sub classes, Member access rules, super keyword, preventing inheritance: final classes and methods, the Object class and its methods.

**Polymorphism – dynamic** binding, method overriding, abstract classes and methods**.**

**Interfaces-** Interfaces Vs Abstract classes, defining an interface, implement interfaces, accessing implementations through interface references, extending interface.

**Inner classes-** Uses of inner classes, local inner classes, anonymous inner classes, static inner classes, examples.

**Packages-** Defining, creating and accessing a package, Understanding CLASSPATH, importing packages.

**UNIT-III**

**Exception handling-** Dealing with errors, benefits of exception handling, the classification of exceptions- exception hierarchy, checked exceptions and unchecked exceptions, usage of try, catch, throw, throws and finally, rethrowing exceptions, exception specification, built in exceptions, creating own exception sub classes.

**Multithreading –** Differences between multiple processes and multiple threads, thread states, creating threads, interrupting threads, thread priorities, synchronizing threads, inter-thread communication, producer consumer pattern, Exploring java.net and java. Text.

**UNIT-IV**

**Applets** – Concepts of Applets, differences between applets and applications, life cycle of an applet, types of applets, creating applets, passing parameters to applets.

**Event Handling:** Events, Handling mouse and keyboard events, Adapter classes.

**Files-** Streams- Byte streams, Character streams, Text input/output.

**Files-** Streams- Byte streams, Character streams, Text input/output, Binary input/output, random access file operations, File management using File class..

**UNIT-V**

**GUI Programming with Java** – AWT class hierarchy, component, container, panel, window, frame, graphics.

**AWT controls:** Labels, button, text field, check box, and graphics.

**Layout Manager –** Layout manager types: border, grid and flow.

**Swing** – Introduction, limitations of AWT, Swing vs AWT.

**TEXT BOOK:**

1. Java Fundamentals – A Comprehensive Introduction, Herbert Schildt and Dale Skrien, TMH.

**REFERENCE BOOKS:**

1. Java for Programmers, P.J.Deitel and H.M.Deitel, PEA (or) Java: How to Program , P.J.Deitel and H.M.Deitel,PHI
2. Object Oriented Programming through Java, P. Radha Krishna, UniversitiesPress.
3. Thinking in Java, Bruce Eckel,PE
4. Programming in Java, S. Malhotra and S. Choudhary, Oxford UniversitiesPress.

**Course Outcomes:**

- An understanding of the principles and practice of object oriented analysis and design in the construction of robust, maintainable programs which satisfy theirrequirements;
- A competence to design, write, compile, test and execute straightforward programs usinga high levellanguage;
- An appreciation of the principles of object orientedprogramming;
- An awareness of the need for a professional approach to design and the importanceof good documentation to the finishedprograms.
- Be able to implement, compile, test and run Java programs comprising more thanone class, to address a particular softwareproblem.
- Demonstrate the ability to use simple data structures like arrays in a Javaprogram.
- Be able to make use of members of classes found in the Java API (such as the Mathclass).

# INDEX


| S. No | Unit | Topic | Page no |
|---|---|---|---|
| 1 | I | **OOP Concepts:-** Data abstraction, encapsulation inheritance | 1 |
| 2 | I | Benefits of Inheritance | 2 |
| 3 | I | Polymorphism, classes and objects | 2 |
| 4 | I | Procedural and object oriented programming paradigms | 3 |
| 5 | I | **Java Programming-** History of Java | 4 |
| 6 | I | Comments, Data types, Variables, Constants | 5-9 |
| 7 | I | Scope and Lifetime of variables | 10 |
| 8 | I | Operators, Operator Hierarchy, Expressions | 11-12 |
| 9 | I | Type conversion and casting, Enumerated types | 12-13 |
| 10 | I | Control flow- block scope, conditional statements, loops, break and continue statements | 13-14 |
| 11 | I | Simple java stand alone programs, arrays | 14-18 |
| 12 | I | Console input and output, formatting output | 18-19 |
| 13 | I | Constructors, methods, parameter passing | 19-20 |
| 14 | I | Static fields and methods, access control, this reference, | 21-30 |
| 15 | I | Overloading methods and constructors, recursion, garbage collection, | 30-34 |
| 16 | I | Building strings, exploring string class. | 34-36 |

| S. No | Unit | Topic | Page no |
|---|---|---|---|
| 17 | II | **Inheritance –** Inheritance hierarchies super and sub classes, Member access rules | 37-40 |
| 18 | II | super keyword, preventing inheritance: final classes and methods, the Object class and its methods. | 40-41 |
| 19 | II | **Polymorphism – dynamic** binding, method overriding, | 41-42 |
| 20 | II | abstract classes and methods**.** | 43 |
| 21 | II | **Interfaces-** Interfaces Vs Abstract classes, defining an interface, implement interfaces | 43-44 |
| 22 | II | Accessing implementations through interface references, extendinginterface. | 45 |
| 23 | II | **Inner classes-** Uses of inner classes, local inner classes | 45-46 |
| 24 | II | Anonymous inner classes, static inner classes, examples. | 46 |
| 25 | II | **Packages-** Defining, creating and accessing a package, | 46-47 |
| 26 | II | Understanding CLASSPATH, importing packages. | 47 |
| 27 | III | **Exception handling-** Dealing with errors, benefits of exception handling | 48 |
| 28 | III | The classification of exceptions- exception hierarchy, checked exceptions and unchecked exceptions | 48-50 |
| 29 | III | Usage of try, catch, throw, throws and finally, | 50-54 |
| 30 | III | Rethrowing exceptions, exception specification, | 54 |
| 31 | III | Built in exceptions, creating own exception sub classes. | 54 |
| 32 | III | **Multithreading –** Differences between multiple processes and multiple threads, thread states | 55-56 |
| 33 | III | Creating threads, interrupting threads, thread priorities, synchronizing threads | 56-59 |
| 34 | III | Inter-thread communication, producer consumer pattern | 59 |
| 35 | III | Exploring java.net and java.text. | 60 |

| S. No | Unit | Topic | Page no |
|---|---|---|---|
| 36 | IV | **Collection Framework in Java** – Introduction to java collections, Overview of java collection framework, Generics | 62 |
| 37 | IV | Commonly used collection classes- Array List, Vector, Hash table, Stack, Enumeration, Iterator | 63-71 |
| 38 | IV | String Tokenizer, Random, Scanner, Calendar and Properties. | 71-76 |
| 39 | IV | **Files-** Streams- Byte streams, Character streams, Text input/output, Binary input/output | 77-82 |
| 40 | IV | Random access file operations, File management using File class. | 83-84 |
| 41 | IV | **Connecting to Database –** JDBC Type 1 to 4 drivers, Connecting to a a database, | 85-88 |
| 42 | IV | Querying a database and processing the results, updating data with JDBC. | 89-94 |
| 43 | V | **GUI Programming with Java-** The AWT class hierarchy, Introduction to Swing, Swing Vs AWT, Hierarchy for Swing components | 95-100 |
| 44 | V | Containers – Jframe, JApplet, JDialog, JPanel | 100-104 |
| 45 | V | Overview of some Swing components – Jbutton, JLabel, JTextField, JTextArea, simple Swing applications, | 104-108 |
| 46 | V | Layout management – Layout manager types – border, grid and flow | 109-111 |
| 47 | V | **Event Handling-** Events, Event sources, Event classes, Event Listeners, | 111-112 |
| 48 | V | Relationship between Event sources and Listeners, Delegation event model, | 112-113 |
| 49 | V | Handling a button click, Handling Mouse events, Adapter classes. | 114-116 |
| 50 | V | **Applets –** Inheritance hierarchy for applets | 118-119 |
| 51 | V | Differences between applets and applications, Life cycle of an applet, | 120 |
| 52 | V | Passing parameters to applets, applet security issues. | 121 |

# OOP Concepts

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

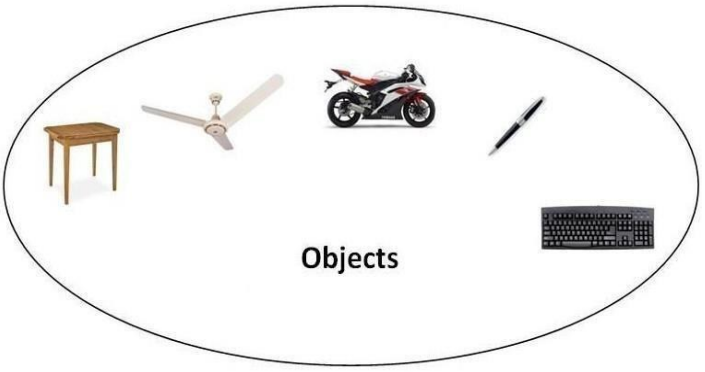
**Simula** is considered as the first object-oriented programming language. The programming paradigm where everything is represented as an object is known as truly object-oriented programming language.

**Smalltalk** is considered as the first truly object-oriented programming language.

### OOPs (Object Oriented Programming System)

**Object** means a real word entity such as pen, chair, table etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing someconcepts:

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




### Object

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

### Class

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

## Inheritance

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

## Polymorphism

When **one task is performed by different ways** i.e. known as polymorphism. For example: to convince the customer differently, to draw something e.g. shape or rectangle etc.

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

Another example can be to speak something e.g. cat speaks meaw, dog barks woof etc.

## Abstraction

**Hiding internal details and showing functionality** is known as abstraction. For example: phone call, we don't know the internal processing.

In java, we use abstract class and interface to achieve abstraction.

## Encapsulation

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

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

## Benefits of Inheritance

- One of the key benefits of inheritance is to minimize the amount of duplicate code in an application by sharing common code amongst several subclasses. Where equivalent code exists in two related classes, the hierarchy can usually be refactored to move the common code up to a mutual superclass. This also tends to result in a better organization of code and smaller, simpler compilationunits.
  - Inheritance can also make application code more flexible to change because classesthat inherit from a common superclass can be used interchangeably. If the return type of a method issuperclass
  - **Reusability** - facility to use public methods of base class without rewriting thesame.
  - **Extensibility** - extending the base class logic as per business logic of the derivedclass.

- **Data hiding -** base class can decide to keep some data private so that it cannotbe altered by the derived class

## Procedural and object oriented programming paradigms

| Features | Procedural Oriented Programming (POP) | Object Oriented Programming (OOPS) |
|---|---|---|
| Divided into | In POP, program is divided into smaller parts called as functions. | in OOPs , the program is divided into parts known as **objects**. |
| Importance | In POP, importance is not given to **data** but to functions as well as **sequence** of actions to be done. | In OOPs, Importance is given to the data rather than procedures or functions because it works as a **real world**. |
| Approach | POP follows **Top Down approach**. | OOPs follows **Bottom Up approach**. |
| Access Specifiers | POP does not have any access specifier. | OOPs has access specifiers named Public, Private, Protected, etc. |
| Data Moving | In POP, Data can move freely from function to function in the system. | In OOPs, objects can move and communicate with each other through member functions. |
| Data Access | In POP, Most function uses Global data for sharing that can be accessed freely from function to function in the system. | In OOPs, data can not move easily from function to function,it can be kept public or private so we can control the access of data. |
| Data Hiding | POP does not have any proper way for hiding data so it is **less secure**. | OOPs provides Data Hiding so provides **more security**. |
| Overloading | In POP, Overloading is not possible. | In OOPs, overloading is possible in the form of Function Overloading and Operator Overloading. |
| Examples | C, VB, FORTRAN, Pascal. | C++, JAVA, VB.NET, C#.NET. |

# Java Programming- History of Java

The history of java starts from Green Team. Java team members (also known as **Green Team**), initiated a revolutionary task to develop a language for digital devices such as set-top boxes, televisionsetc.

For the green team members, it was an advance concept at that time. But, it was suited for internet programming. Later, Java technology as incorporated by Netscape.

Currently, Java is used in internet programming, mobile devices, games, e-business solutions etc. There are given the major points that describes the history of java.

1) **James Gosling**, **Mike Sheridan**, and **Patrick Naughton** initiated the Java language project in June 1991. The small team of sun engineers called **Green Team**.

2) Originally designed for small, embedded systems in electronic appliances like set-topboxes.

3) Firstly, it was called **"Greentalk"** by James Gosling and file extension was.gt.

**4) After that, it was called Oak and was developed as a part of the Green project.**

## Java Version History

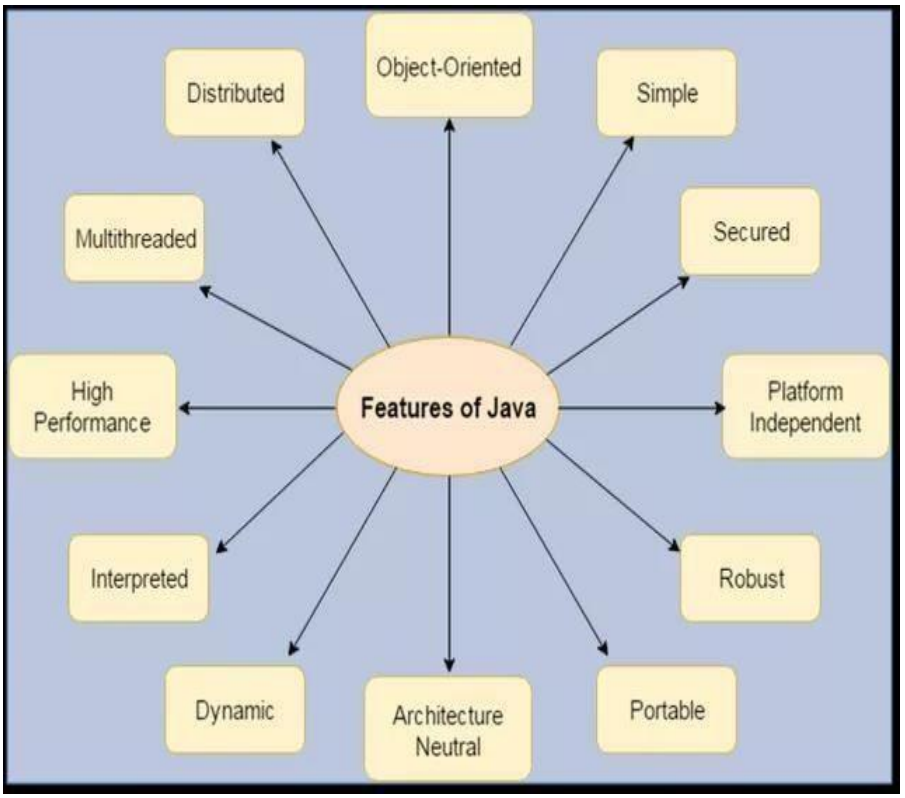
There are many java versions that has been released. Current stable release of Java is Java SE 8.

1. JDK Alpha and Beta (1995)
2. JDK 1.0 (23rd Jan, 1996)
3. JDK 1.1 (19th Feb, 1997)
4. J2SE 1.2 (8th Dec, 1998)
5. J2SE 1.3 (8th May, 2000)
6. J2SE 1.4 (6th Feb, 2002)
7. J2SE 5.0 (30th Sep,2004)
8. Java SE 6 (11th Dec,2006)
9. Java SE 7 (28th July, 2011)
10. Java SE 8 (18th March,2014)

## Features of Java

There is given many features of java. They are also known as java buzzwords. The Java Features given below are simple and easy to understand.

1. Simple
2. Object-Oriented
3. Portable
4. Platformindependent
5. Secured
6. Robust
7. Architectureneutral
8. Dynamic
9. Interpreted
10. HighPerformance
11. Multithreaded
12. Distributed




## Java Comments

The java comments are statements that are not executed by the compiler and interpreter. The comments can be used to provide information or explanation about the variable, method, class or any statement. It can also be used to hide program code for specific time.

## Types of Java Comments

There are 3 types of comments in java.

1. Single LineComment
2. Multi LineComment
3. DocumentationComment

## Java Single Line Comment

The single line comment is used to comment only one line.

**Syntax:**

```
1. //This is single line comment
```

**Example:**

```
public class CommentExample1 {
public static void main(String[] args) {
   int i=10;//Here, i is a variable
   System.out.println(i);
}
}
```

Output:

```
10
```

## Java Multi Line Comment

The multi line comment is used to comment multiple lines of code.

**Syntax:**

```
 /*
 This
 is
 multi line
 comment
 */
```

**Example:**

```
public class CommentExample2 {
public static void main(String[] args) {
/* Let's declare and
 print variable in java.*/
   inti=10;
   System.out.println(i);
} }
```

Output:

```
10
```

## Java Documentation Comment

The documentation comment is used to create documentation API. To create documentation API, you need to use **javadoc tool**.

**Syntax:**

```
/**
This
is
documentation
comment
*/
```

**Example:**

```
/** The Calculator class provides methods to get addition and subtraction of given 2 numbers.*/
public class Calculator {
/** The add() method returns addition of given numbers.*/
public static int add(int a, int b){return a+b;}
/** The sub() method returns subtraction of given numbers.*/
public static int sub(int a, int b){return a-b;}
}
```

Compile it by javac tool:

```
javac Calculator.java
```

Create Documentation API by javadoc tool:

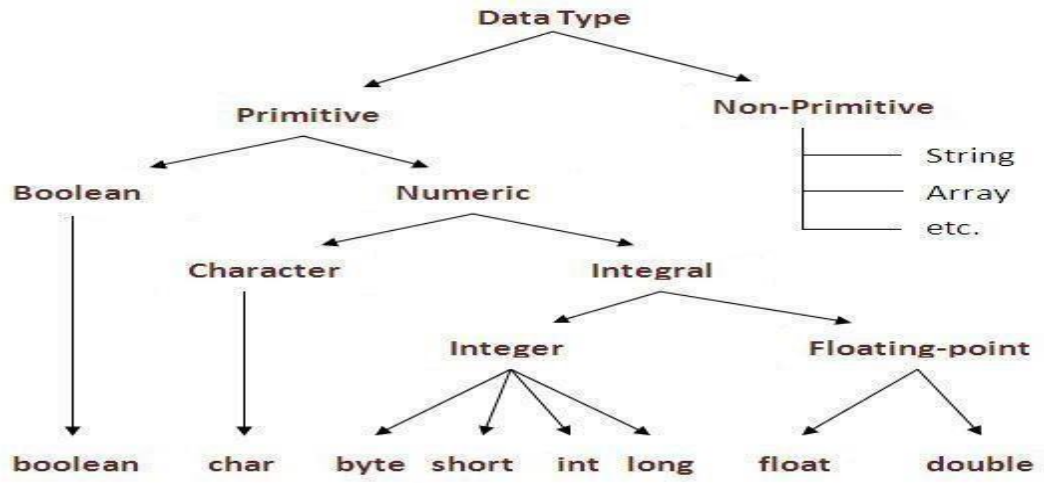
```
javadoc Calculator.java
```

Now, there will be HTML files created for your Calculator class in the current directory. Open the HTML files and see the explanation of Calculator class provided through documentation comment.

## Data Types

Data types represent the different values to be stored in the variable. In java, there are two types of data types:

- Primitive datatypes
- Non-primitive datatypes




| DataType | DefaultValue | 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 |

### Java Variable Example: Add Two Numbers

```
classSimple{
public static void main(String[] args){
inta=10;
int b=10;
int c=a+b;
System.out.println(c);
}}
```

Output:20

## Variables and Data Types in Java

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

There are two types of data types in java: primitive and non-primitive.

## Types of Variable

There are three types of variables in java:

- localvariable
- instancevariable
- staticvariable

### *1) LocalVariable*

A variable which is declared inside the method is called local variable.

### *2) Instance Variable*

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

### *3) Staticvariable*

A variable that is declared as static is called static variable. It cannot be local.

We will have detailed learning of these variables in next chapters.

Example to understand the types of variables in java

```
class A{
int data=50;//instance variable
static int m=100;//static variable
void method(){
int n=90;//local variable
}
}//end of class
```

## Constants in Java

A constant is a variable which cannot have its value changed after declaration. It uses the **'final' keyword.**

**Syntax**

```
modifier final dataType variableName = value; //global constant

modifier static final dataType variableName = value; //constant within a c
```

## Scope and Life Time of Variables

The scope of a variable defines the section of the code in which the variable is visible. As a general rule, variables that are defined within a block are not accessible outside that block. The lifetime of a variable refers to how long the variable exists before it isdestroyed. Destroying variables refers to deallocating the memory that was allotted to the variables when declaring it. We have written a few classes till now. You might have observed that not all variables are the same. The ones declared in the body of a method were different from those that were declared in the class itself. There are three types of variables: instance variables, formal parameters or local variables and localvariables.

### Instance variables

Instance variables are those that are defined within a class itself and not in any method or constructor of the class. They are known as instance variables because every instance of the class (object) contains a copy of these variables. The scope of instance variables is determined by the access specifier that is applied to these variables. We have already seen about it earlier. The lifetime of these variables is the same as the lifetime of the object to which it belongs. Object once created do not exist for ever. They are destroyed by the garbage collector of Java when there are no more reference to that object. We shall see about Java's automatic garbage collector later on.

### Argument variables

These are the variables that are defined in the header oaf constructor or a method. The scope of these variables is the method or constructor in which they are defined. The lifetime is limited to the time for which the method keeps executing. Once the method finishes execution, these variables aredestroyed.

### Local variables

A local variable is the one that is declared within a method or a constructor (not in the header). The scope and lifetime are limited to the methoditself.

One important distinction between these three types of variables is that access specifiers can be applied to instance variables only and not to argument or local variables.

In addition to the local variables defined in a method, we also have variables that are defined in bocks life an if block and an else block. The scope and is the same as that of the block itself.

## Operators in java

**Operator** in java is a symbol that is used to perform operations. For example: +, -, *, / etc.

There are many types of operators in java which are given below:

- UnaryOperator,
- ArithmeticOperator,
- shiftOperator,
- RelationalOperator,
- BitwiseOperator,
- LogicalOperator,
- Ternary Operatorand
- AssignmentOperator.

## Operators Hierarchy

Operator Precedence

| Operators | Precedence |
|---|---|
| postfix | `expr++ expr--` |
| unary | `++expr --expr +expr -expr ~ !` |
| multiplicative | `* / %` |
| additive | `+ -` |
| shift | `<< >> >>>` |
| relational | `< > <= >= instanceof` |
| equality | `== !=` |
| bitwise AND | `&` |
| bitwise exclusive OR | `^` |
| bitwise inclusive OR | `\|` |
| logical AND | `&&` |
| logical OR | `\|\|` |
| ternary | `? :` |
| assignment | `= += -= *= /= %= &= ^= \|= <<= >>= >>>=` |

## Expressions

Expressions are essential building blocks of any Java program, usually created to produce a new value, although sometimes an expression simply assigns a value to a variable. Expressions are built using values, variables, operators and method calls.

**Types of Expressions**

While an expression frequently produces a result, it doesn't always. There are three types of expressions in Java:

- Those that produce a value, i.e. the result of (1 + 1)
- Those that assign a variable, for example (v =10)
- Those that have no result but might have a "side effect" because an expression can include a wide range of elements such as method invocations or increment operators that modify the state (i.e. memory) of aprogram.

## Java Type casting and Type conversion

**Widening or Automatic Type Conversion**

Widening conversion takes place when two data types are automatically converted. This happens when:

- The two data types arecompatible.
- When we assign value of a smaller data type to a bigger datatype.

For Example, in java the numeric data types are compatible with each other but no automatic conversion is supported from numeric type to char or boolean. Also, char and boolean are not compatible with each other.

Byte –> Short –> Int –> Long – > Float –> Double

Widening or Automatic Conversion

**Narrowing or Explicit Conversion**

If we want to assign a value of larger data type to a smaller data type we perform explicit type casting or narrowing.

- This is useful for incompatible data types where automatic conversion cannot bedone.
- Here, target-type specifies the desired type to convert the specified valueto.

Double –> Float –> Long –> Int –> Short –> Byte

- Narrowing or Explicit Conversion

## Java Enum

**Enum in java** is a data type that contains 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) etc. The java enum constants are static and final implicitly. It is available from JDK 1.5.

Java Enums can be thought of as classes that have fixed set of constants.

### Simple example of java enum

```
classEnumExample1{
public enum Season { WINTER, SPRING, SUMMER, FALL }

public static void main(String[] args) {
for (Season s : Season.values())
System.out.println(s);
}}
```

**Output:**

```
WINTER
SPRING
SUMMER
FALL
```

### Control Flow Statements

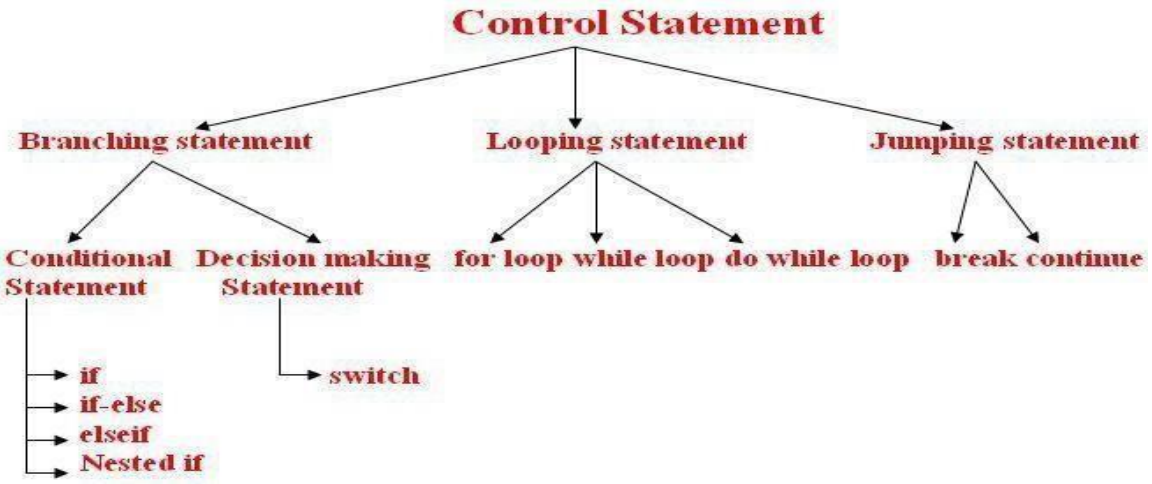
The control flow statements in Java allow you to run or skip blocks of code when special conditions are met.

**The "if" Statement**

The "if" statement in Java works exactly like in most programming languages. With the help of "if" you can choose to execute a specific block of code when a predefined condition is met. The structure of the "if" statement in Java looks like this:

```
if(condition) {
// execute this code
}
```

The condition is Boolean. Boolean means it may be true or false. For example you may put a mathematical equation as condition. Look at this full example:




## Creating a Stand-Alone Java Application

**1.** Write a main method that runs your program. You can write this method anywhere. In this example, I'll write my main method in a class called Main that has no other methods. **For example:**

```
2.  public class Main
3.  {
4.    public static void main(String[] args)
5.    {
6.      Game.play();
7.    } }
```

8. Make sure your code is compiled, and that you have tested it thoroughly.
9. If you're using Windows, you will need to set your path to include Java, if you haven't done so already. This is a delicate operation. Open Explorer, and look inside C:\ProgramFiles\Java, and you should see some version of the JDK. Open this folder, and then open the bin folder. Select the complete path from the top of the Explorer window, and press Ctrl-C to copyit.

Next, find the "My Computer" icon (on your Start menu or desktop), right-click it, and select properties. Click on the Advanced tab, and then click on the Environment variables button. Look at the variables listed for all users, and click on the Path variable. Do not delete the contents of this variable! Instead, edit the contents by moving the cursor to the right end, entering a semicolon (;), and pressing Ctrl-V to paste the path you copied earlier. Then go ahead and save your changes. (If you have any Cmd windows open, you will need to close them.)

10. If you're using Windows, go to the Start menu and type "cmd" to run a program that brings up a command prompt window. If you're using a Mac or Linux machine, run the Terminal program to bring up a commandprompt.
11. In Windows, type dir at the command prompt to list the contents of the currentdirectory. On a Mac or Linux machine, type ls to dothis.

12. Now we want to change to the directory/folder that contains your compiled code. Look at the listing of sub-directories within this directory, and identify which one contains yourcode. Type cd followed by the name of that directory, to change to that directory. For example, to change to a directory called Desktop, you would type:

**cd Desktop**

To change to the parent directory, type:

**cd ..**

Every time you change to a new directory, list the contents of that directory to see where to go next. Continue listing and changing directories until you reach the directory that contains your .class files.

13. If you compiled your program using Java 1.6, but plan to run it on a Mac, you'll needto recompile your code from the command line, bytyping:

javac -target 1.5 *.java

14. Now we'll create a single JAR file containing all of the files needed to run yourprogram.

## Arrays

Java provides a data structure, the **array**, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the sametype.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables.

This tutorial introduces how to declare array variables, create arrays, and process arrays using indexed variables.

**Declaring Array Variables:**

To use an array in a program, you must declare a variable to reference the array, and you must specify the type of array the variable can reference. Here is the syntax for declaring an array variable:

```
dataType[] arrayRefVar; // preferred way.
or
dataType arrayRefVar[]; // works but not preferred way.
```

**Note:** The style**dataType[] arrayRefVar** is preferred. The style **dataType arrayRefVar[]** comes from the C/C++ language and was adopted in Java to accommodate C/C++programmers.

**Example:**

The following code snippets are examples of this syntax:

```
double[] myList;        // preferred way.
or
doublemyList[];         // works but not preferred way.
```

Creating Arrays:

You can create an array by using the new operator with the following syntax:

```
arrayRefVar= new dataType[arraySize];
```

The above statement does two things:

- It creates an array using newdataType[arraySize];
- It assigns the reference of the newly created array to the variablearrayRefVar.

Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement, as shown below:

```
dataType[] arrayRefVar = new dataType[arraySize];
```

Alternatively you can create arrays as follows:

```
dataType[] arrayRefVar = {value0, value1, ..., valuek};
```

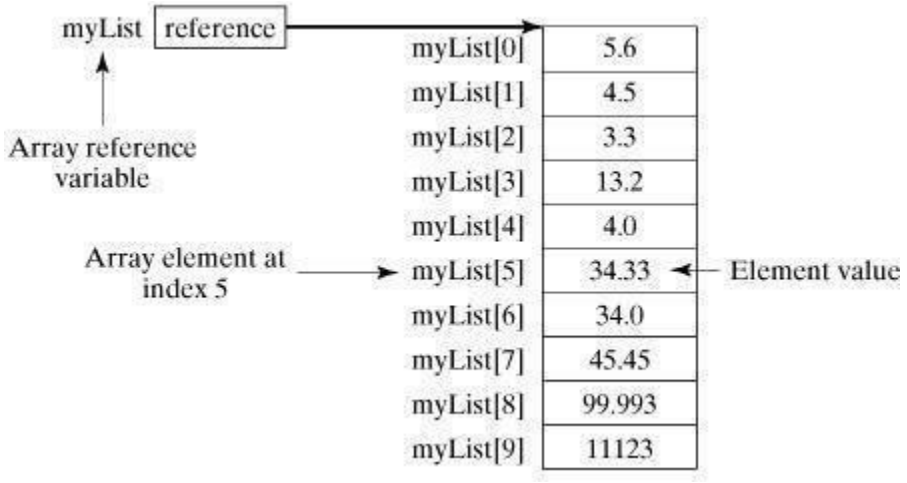
The array elements are accessed through the **index**. Array indices are 0-based; that is, they start from 0 to **arrayRefVar.length-1**.

**Example:**

Following statement declares an array variable, myList, creates an array of 10 elements of double type and assigns its reference tomyList:

```
double[] myList = new double[10];
```

Following picture represents array myList. Here, myList holds ten double values and the indices are from 0 to 9.




**Processing Arrays:**

When processing array elements, we often use either for loop or for each loop because all of the elements in an array are of the same type and the size of the array is known.

**Example:**

Here is a complete example of showing how to create, initialize and process arrays:

```
public class TestArray
{
  public static void main(String[] args) {
    double[] myList = {1.9, 2.9, 3.4, 3.5};
          // Print all the array elements
    for (int i = 0; i <myList.length; i++){
      System.out.println(myList[i] + " ");
    }
    // Summing all elements
    double total = 0;
    for(int i = 0; i <myList.length; i++) {
      total += myList[i];
    }
    System.out.println("Total is " + total);
    // Finding the largest element
    double max = myList[0];
    for(int i = 1; i <myList.length; i++) { if
      (myList[i] >max) max = myList[i];
    }
    System.out.println("Max is " + max);
  }
}
```

This would produce the following result:

```
1.9
2.9
3.4
3.5
Total is 11.7
Max is 3.5
```

```
public class TestArray {
public static void main(String[] args) {
    double[] myList = {1.9, 2.9, 3.4, 3.5};
 // Print all the array elements
    for(double element: myList) {
       System.out.println(element);
}}}
```

## Java Console Class

The Java Console class is be used to get input from console. It provides methods to read texts and passwords.

If you read password using Console class, it will not be displayed to the user.

The java.io.Console class is attached with system console internally. The Console class is introduced since 1.5.

Let's see a simple example to read text from console.

```
1.      Stringtext=System.console().readLine();
2.      System.out.println("Text is:"+text);
```

**Java ConsoleExample**

```
importjava.io.Console;
classReadStringTest{
public static void main(String args[]){
Console c=System.console();
System.out.println("Enter your name: ");
String n=c.readLine();
System.out.println("Welcome"+n);    }    }
```

Output

```
Enter your name: Nakul Jain
Welcome Nakul Jain
```

## Constructors

**Constructor in java** is a *special type of method* that is used to initialize the object.

Java constructor is *invoked at the time of object creation*. It constructs the values i.e. provides data for the object that is why it is known as constructor.

There are basically two rules defined for the constructor.

1. Constructor name must be same as its classname
2. Constructor must have no explicit returntype

### Types of java constructors

There are two types of constructors:

1. Default constructor (no-argconstructor)
2. Parameterizedconstructor

## Java Default Constructor

A constructor that have no parameter is known as default constructor.

### Syntax of default constructor:

1. <class_name>(){}

### Example of default constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

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

**Output:** Bike is created

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.

```
classStudent4{
intid;
Stringname;

Student4(int i,String n){
id = i;
name = n;
}
voiddisplay(){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:**

```
111Karan
222Aryan
```

## Constructor Overloading in Java

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

Example of Constructor Overloading

```
classStudent5{
intid; String
name;
intage;
Student5(int i,String n){
id = i;
name = n;
}
Student5(int i,String n,int a){
id = i;
name = n;
age=a;
}
voiddisplay(){System.out.println(id+" "+name+" "+age);}

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

```
s2.display();
} }
```

**Output:**

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

## Java CopyConstructor

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

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

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

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

```
classStudent6{
intid;
Stringname;
Student6(int i,String n){
id = i;
name = n;
}

Student6(Student6 s){
id = s.id;
name =s.name;
}
voiddisplay(){System.out.println(id+" "+name);}

public static void main(String args[]){
Student6 s1 = new Student6(111,"Karan");
Student6 s2 = new Student6(s1);
s1.display();
s2.display();
} }
```

Output:

```
111Karan
111Karan
```

## Java -Methods

A Java method is a collection of statements that are grouped together to perform an operation. When you call the System.out.**println()** method, for example, the system actually executes several statements in order to display a message on the console.

Now you will learn how to create your own methods with or without return values, invoke a method with or without parameters, and apply method abstraction in the program design.

Creating Method

Considering the following example to explain the syntax of a method −

**Syntax**

```
public static int methodName(int a, int b) {
   // body
}
```

Here,

- **public static** −modifier
- **int** − returntype
- **methodName** − name of the method
- **a, b** − formalparameters
- **int a, int b** − list ofparameters

Method definition consists of a method header and a method body. The same is shown in the following syntax −

**Syntax**

```
modifier returnType nameOfMethod (Parameter List) {
   // method body
}
```

The syntax shown above includes −

- **modifier**− It defines the access type of the method and it is optional touse.
- **returnType**− Method may return avalue.
- **nameOfMethod**− This is the method name. The method signature consists of themethod name and the parameter list.

- **Parameter List** − The list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain zeroparameters.
- **method body** − The method body defines what the method does with thestatements.

## Call by Value and Call by Reference in Java

There is only call by value in java, not call by reference. If we call a method passing a value, it is known as call by value. The changes being done in the called method, is not affected in the calling method.

### Example of call by value in java

In case of call by value original value is not changed. Let's take a simple example:

```
classOperation{
 intdata=50;
 voidchange(int data){
 data=data+100;//changes will be in the local variable only
 }
 public static void main(String args[]){
   Operation op=new Operation();
   System.out.println("before change "+op.data);
   op.change(500);
   System.out.println("after change "+op.data);
 }
}
```

```
Output:before change 50
       after change 50
```

In Java, parameters are always passed by value. For example, following program prints i = 10, j = 20.

```
// Test.java
class Test {
   // swap() doesn't swap i and j
   public static void swap(Integer i, Integer j) {
      Integer temp = new Integer(i);
      i = j;
      j = temp;
   }
   public static void main(String[] args) {
      Integer i = new Integer(10);
      Integer j = new Integer(20);
      swap(i, j);
      System.out.println("i = " + i + ", j = " + j);
```

```
  }
}
```

## Static Fields and Methods

The **static keyword** in java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than instance of the class.

The static can be:

1. variable (also known as classvariable)
2. method (also known as classmethod)
3. block
4. nestedclass

### Java static variable

If you declare any variable as static, it is known static variable.

- The static variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees,college name of studentsetc.
- The static variable gets memory only once in class area at the time of classloading.

### Advantage of static variable

It makes your program **memory efficient** (i.e it saves memory).

### *Understanding problem without static variable*

```
1. class Student{
2.     int rollno;
3.     String name;
4.     String college="ITS";
5.}
```

### Example of static variable

```
//Program of static variable
class Student8{
  int rollno;
```

```
  String name;
  staticString college ="ITS";
  Student8(int r,String n){
  rollno =r;
  name =n;
  }
 voiddisplay (){System.out.println(rollno+" "+name+" "+college);}
 public static void main(String args[]){
 Student8 s1 = new Student8(111,"Karan");
 Student8 s2 = new Student8(222,"Aryan");

 s1.display();
 s2.display();
 } }
```

```
Output:111 KaranITS
       222 AryanITS
```

## Java static method

If you apply static keyword with any method, it is known as static method.

- A static method belongs to the class rather than object of aclass.
- A static method can be invoked without the need for creating an instance of aclass.
- static method can access static data member and can change the value ofit.

### Example of static method

```
//Program of changing the common property of all objects(static field).

classStudent9{
    int rollno;
    String name;
    staticString college = "ITS";
    static void change(){
    college = "BBDIT";
    }
    Student9(int r, String n){
    rollno =r;
    name =n;
```

```
}
 voiddisplay (){System.out.println(rollno+" "+name+" "+college);}
public static void main(String args[]){
Student9.change();
Student9 s1 = new Student9 (111,"Karan");
Student9 s2 = new Student9 (222,"Aryan");
Student9 s3 = new Student9 (333,"Sonoo");
s1.display();
s2.display();
s3.display();
} }
```

```
Output:111 Karan BBDIT
    222 Aryan BBDIT
     333 Sonoo BBDIT
```

## Java static block

- Is used to initialize the static datamember.
- It is executed before main method at the time of classloading.

### Example of static block

```
class A2{
 static{System.out.println("static block is invoked");}
 public static void main(String args[]){
 System.out.println("Hello main");
 } }
```

```
Output: static block is invoked
     Hello main
```

## Access Control

### Access Modifiers in java

There are two types of modifiers in java: **access modifiers** and **non-access modifiers**.

The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.

There are 4 types of java access modifiers:

1. private
2. default
3. protected
4. public

### private access modifier

The private access modifier is accessible only within class.

### Simple example of private access modifier

In this example, we have created two classes A and Simple. A class contains private data member and private method. We are accessing these private members from outside the class, so there is compile time error.

```
classA{
private int data=40;
private void msg(){System.out.println("Hello java");}}
public class Simple{
 public static void main(String args[]){
  A obj=new A();
  System.out.println(obj.data);//Compile Time Error
  obj.msg();//Compile Time Error
  } }
```

### 2) default accessmodifier

If you don't use any modifier, it is treated as **default** bydefault. The default modifier is accessible only withinpackage.

Example of default accessmodifier

In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside thepackage.

```
//save by A.java
package pack;
class A{
  voidmsg(){System.out.println("Hello");}
}

//save by B.java
package mypack;
import pack.*;
```

```
classB{
  public static void main(String args[]){
   A obj = new A();//Compile Time Error
   obj.msg();//Compile Time Error }}
```

In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside thepackage.

### 3) protected accessmodifier

The **protected access modifier** is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

Example of protected access modifier

In this example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only throughinheritance.

```
//save by A.java
package pack;
public class A{
protected void msg(){System.out.println("Hello");} }
//save by B.java
package mypack;
import pack.*;
class B extends A{
  public static void main(String args[]){
   B obj = new B();
   obj.msg();
  } }
```

Output:Hello

### 4) public accessmodifier

The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

Example of public access modifier

```
//save by A.java
package pack;
public class A{
public void msg(){System.out.println("Hello");}}
//save by B.java
package mypack;
import pack.*;
class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  } }
```

Output:Hello

### Understanding all java access modifiers

Let's understand the access modifiers by a simple table.

| Access Modifier | within class | within package | outsidepackageby subclassonly | outside package |
|---|---|---|---|---|
| **Private** | Y | N | N | N |
| **Default** | Y | Y | N | N |
| **Protected** | Y | Y | Y | N |
| **Public** | Y | Y | Y | Y |

## this keyword in java

### Usage of java this keyword

Here is given the 6 usage of java this keyword.

1. this can be used to refer current class instancevariable.
2. this can be used to invoke current class method(implicitly)
3. this() can be used to invoke current classconstructor.

4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructorcall.
6. this can be used to return the current class instance from themethod.

```
class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
this.rollno=rollno;
this.name=name;
this.fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}
class TestThis2{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=newStudent(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

Output:
111 ankit 5000
112 sumit 6000

## Difference between constructor and method in java

| Java Constructor | Java Method |
|---|---|
| Constructor is used to initialize the state of an object. | Method is used to expose behaviour of anobject. |
| Constructor must not have return type. | Method must have return type. |
| Constructor is invoked implicitly. | Method is invoked explicitly. |
| The java compiler provides a default constructor if you don't have any constructor. | Method is not provided by compiler in any case. |
| Constructor name must be same as the class name. | Method name may or may not be |

| | |
|---|---|
| | same as class name. |

There are many differences between constructors and methods. They are given belo

## Constructor Overloading in Java

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

## Example of Constructor Overloading

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

public static void main(String args[]){
Student5 s1 = new Student5(111,"Karan");
Student5 s2 = newStudent5(222,"Aryan",25);
s1.display();
s2.display();
}
}
```

**Output:**

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

## Method Overloading in java

If a class has multiple methods having same name but different in parameters, it is known as **MethodOverloading**.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

## 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.

In this example, we are creating static methods so that we don't need to create instance for calling methods.

```
classAdder{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
classTestOverloading1{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
}}
```

**Output:**

```
22
33
```

## 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.

## Recursion in Java

Recursion in java is a process in which a method calls itself continuously. A method in java that calls itself is called recursive method.

### Java Recursion Example 1: Factorial Number

```
public class RecursionExample3 {
  static int factorial(int n){
   if(n  ==  1)
     return 1;
     else
    return(n * factorial(n-1));
    } }
public static void main(String[] args) {
System.out.println("Factorial of 5 is: "+factorial(5));
} }
```

**Output:**

```
Factorial of 5 is: 120
```

## Java Garbage Collection

In java, garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

### Advantage of Garbage Collection

- It makes java **memory efficient** because garbage collector removes theunreferenced objects from heapmemory.
- It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extraefforts.

### gc() method

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

```
public static void gc(){}
```

**Simple Example of garbage collection in java**

```
public class TestGarbage1{
 public void finalize(){System.out.println("object is garbage collected");}
 public static void main(String args[]){
  TestGarbage1 s1=new TestGarbage1();
  TestGarbage1 s2=new TestGarbage1();
  s1=null;
  s2=null;
  System.gc();
 } }
```

```
object is garbage collected
object is garbage collected
```

## Java String

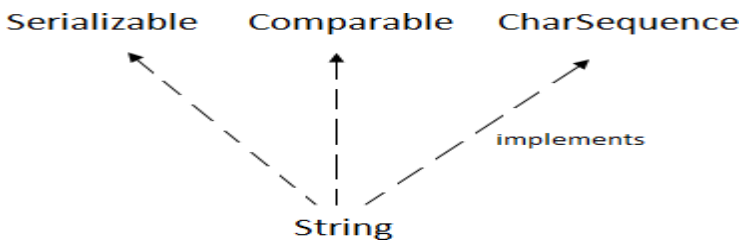
string is basically an object that represents sequence of char values. An array of characters works same as java string. For example:

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

ssame as:

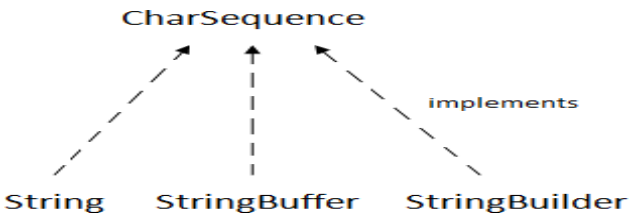
```
1. Strings="javatpoint";
```

2. **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.
3. The java.lang.String class implements *Serializable*, *Comparable* and *CharSequence*interfaces.




CharSequence Interface

The CharSequence interface is used to represent sequence of characters. It is implemented by String, StringBuffer and StringBuilder classes. It means, we can create string in java by using these 3 classes.




The java String is immutable i.e. it cannot be changed. Whenever we change any string, a new instance is created. For mutable string, you can use StringBuffer and StringBuilder classes.

There are two ways to create String object:

1. By stringliteral
2. By newkeyword

## String Literal

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

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

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

```
1. Strings1="Welcome";
2. String s2="Welcome";//will not create newinstance
```

### By new keyword

```
1. 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 heap (non pool).

### Java String Example

```
public class StringExample{
public static void main(String args[]){
String s1="java";//creating string by java string literal
charch[]={'s','t','r','i','n','g','s'};
String s2=new String(ch);//converting char array to string
String s3=new String("example");//creating java string by new keyword
System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
}}
```

```
java
```

strings
example

## Immutable String in Java

In java, **string objects are immutable**. Immutable simply means unmodifiable or unchangeable.

Once string object is created its data or state can't be changed but a new string object is created.

Let's try to understand the immutability concept by the example given below:

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

Output:Sachin

```
class Testimmutablestring1{
  public static void main(String args[]){
    String s="Sachin";
    s=s.concat(" Tendulkar");
    System.out.println(s);
  } }
```

Output:Sachin Tendulkar

# Inheritance in Java

**Inheritance in java** is a mechanism in which one object acquires all the properties and behaviors of parent object. Inheritance represents the **IS-A relationship**, also known as *parent-child*relationship.

### Why use inheritance in java

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

### Syntax of Java Inheritance

```
1. classSubclass-name extends Superclass-name
2. {
3.     //methods and
fields4. }
```

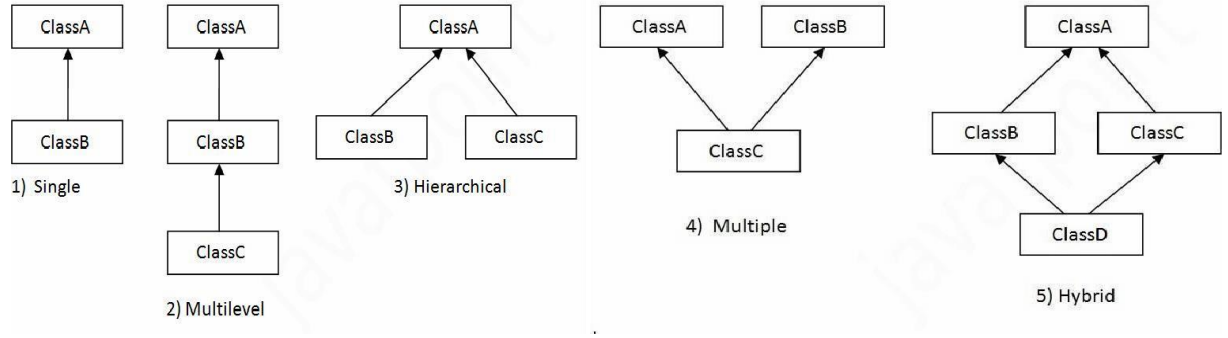
The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

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

```
Programmer salary is:40000.0
```

Bonus of programmeris:10000

## Types of inheritance in java




## Single Inheritance Example

*File: TestInheritance.java*

```
classAnimal{
voideat(){System.out.println("eating...");}
}
classDog extends Animal{
voidbark(){System.out.println("barking...");}
}
classTestInheritance{
public static void main(String args[]){
Dog d=new Dog();
d.bark();
d.eat();
}}
```

Output:
barking...
eating...

## Multilevel Inheritance Example

*File: TestInheritance2.java*

```
classAnimal{
voideat(){System.out.println("eating...");}
}
classDog extends Animal{
voidbark(){System.out.println("barking...");}
}
classBabyDog extends Dog{
voidweep(){System.out.println("weeping...");}
}
classTestInheritance2{
```

```
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}}
```

Output:

```
weeping...
barking...
eating...
```

## Hierarchical Inheritance Example

*File: TestInheritance3.java*

```
classAnimal{
voideat(){System.out.println("eating...");}
}
classDog extends Animal{
voidbark(){System.out.println("barking...");}
}
classCat extends Animal{
voidmeow(){System.out.println("meowing...");}
}
classTestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
c.eat();
//c.bark();//C.T.Error
}}
```

Output:

```
meowing...
eating...
```

## Member access and Inheritance

A subclass includes all of the members of its super class but it cannot access those members of the super class that have been declared as private. Attempt to access a private variable would cause compilation error as it causes access violation. The variables declared as private, is only accessible by other members of its own class. Subclass have no access to it.

## super keyword in java

The **super** keyword in java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

### Usage of java super Keyword

1. super can be used to refer immediate parent class instancevariable.
2. super can be used to invoke immediate parent classmethod.
3. super() can be used to invoke immediate parent classconstructor.

### super is used to refer immediate parent class instance variable.

```
classAnimal{
Stringcolor="white";
}
classDog extends Animal{
Stringcolor="black";
voidprintColor(){ System.out.println(color);//prints
color of Dogclass
System.out.println(super.color);//prints color of Animal class
}
}
classTestSuper1{
public static void main(String args[]){
Dog d=new Dog();
```

```
d.printColor();
}}
```

Output:

```
black
white
```

## Final Keyword in Java

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

1. variable
2. method
3. class

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

## Object class in Java

The **Object class** is the parent class of all the classes in java by default. In other words, it is the topmost class of java.

The Object class is beneficial if you want to refer any object whose type you don't know. Notice that parent class reference variable can refer the child class object, know as upcasting.

Let's take an example, there is getObject() method that returns an object but it can be of any type like Employee,Student etc, we can use Object class reference to refer that object. For example:

```
1. Object obj=getObject();//we don't know what object will be returned from thismethod
```

The Object class provides some common behaviors to all the objects such as object can be compared, object can be cloned, object can be notified etc.

## Method Overriding in Java

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

### Usage of Java Method Overriding

- Method overriding is used to provide specific implementation of a method that is already provided by its superclass.
- Method overriding is used for runtimepolymorphism

### *Rules for Java Method Overriding*

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

### Example of method overriding

```
Class Vehicle{
voidrun(){System.out.println("Vehicle is running");}
}
classBike2 extends Vehicle{
voidrun(){System.out.println("Bike is running safely");}
public static void main(String args[]){
Bike2 obj = new Bike2();
obj.run();
}
```

**Output**:Bike is running safely

```
1. classBank{
intgetRateOfInterest(){return 0;}
}
classSBI extends Bank{
intgetRateOfInterest(){return 8;}
}
 classICICI extends Bank{
intgetRateOfInterest(){return 7;}
}
classAXIS extends Bank{
intgetRateOfInterest(){return 9;}
}
classTest2{
public static void main(String args[]){
SBI s=new SBI();
ICICI i=new ICICI();
AXIS a=new AXIS();
System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());
System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());
System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());
} }
```

```
Output:
SBI Rate of Interest: 8
```

```
ICICI Rate of Interest: 7
AXIS Rate of Interest: 9
```

## Abstract class in Java

A class that is declared with abstract keyword is known as abstract class in java. It can have abstract and non-abstract methods (method with body). It needs to be extended and its method implemented. It cannot be instantiated.

### Example abstract class

```
1. abstract classA{}
```

### abstractmethod

```
1. abstract void printStatus();//no body and abstract
```

### Example of abstract class that has abstract method

```
abstract class Bike{
  abstract void run();
}
classHonda4 extends Bike{
voidrun(){System.out.println("running safely..");}
public static void main(String args[]){
 Bike obj = new Honda4();
 obj.run();
}
1. }
```

```
runningsafely..
```

## Interface in Java

An **interface in java** is a blueprint of a class. It has static constants and abstract methods.

The interface in java is **a mechanism to achieve abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve abstraction and multiple inheritance in Java.

Java Interface also **represents IS-A relationship**.

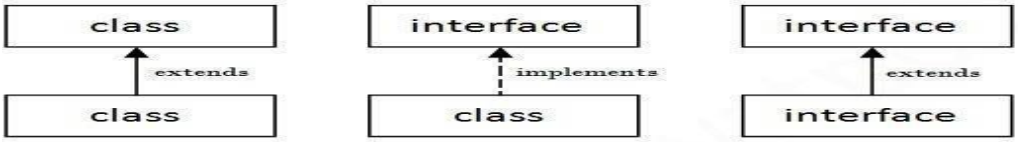
It cannot be instantiated just like abstract class.

There are mainly three reasons to use interface. They are given below.

- It is used to achieveabstraction.
- By interface, we can support the functionality of multipleinheritance.
- It can be used to achieve loosecoupling.

### Internal addition by compiler

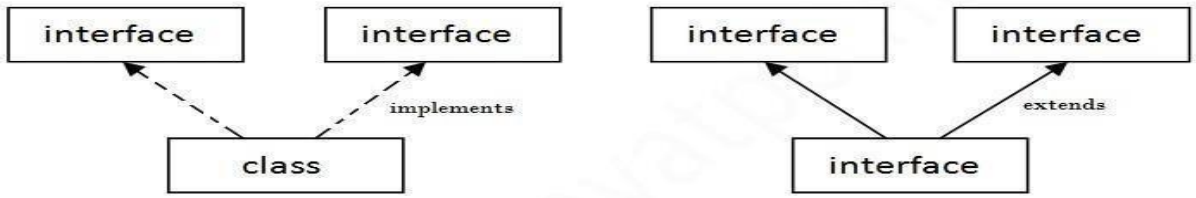

### *Understanding relationship between classes and interfaces*




```
//Interface declaration: by first user
interfaceDrawable{
voiddraw();
}
//Implementation: by second user
classRectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
}
classCircle implements Drawable{
public void draw(){System.out.println("drawing circle");}
}
//Using interface: by third user
classTestInterface1{
public static void main(String args[]){
Drawable d=new Circle();//In real scenario, object is provided by method e.g. getDrawable()
d.draw();
}}
```

Output:drawing circle

### Multiple inheritance in Java by interface




**Multiple Inheritance in Java**

```
interfacePrintable{
```

```
voidprint();
}
interfaceShowable{
voidshow();
}
classA7 implements Printable,Showable{
public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}
public static void main(String args[]){
A7 obj = new A7();
obj.print();
obj.show();
 } }
```

Output:Hello
Welcome

| Abstract class | Interface |
|---|---|
| 1) Abstract class can **have abstract and non-abstract**methods. | Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods**also. |
| 2) Abstract class **doesn't support multipleinheritance**. | Interface **supports multiple inheritance**. |
| 3) Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variables**. |
| 4) Abstract class **can provide the implementation ofinterface**. | Interface **can't provide the implementation of abstract class**. |
| 5) The **abstract keyword** is used to declare abstract class. | The**interface keyword** is used to declare interface. |
| 6) **Example:**<br>public abstract class Shape{<br>public abstract void draw();<br>} | **Example:**<br>public interface Drawable{<br>void draw();<br>} |

## Java Inner Classes

**Java inner class** or nested class is a class which is declared inside the class or interface.

We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.

***Syntax of Inner class***

```
1. classJava_Outer_class{
2.  //code
3.  classJava_Inner_class{
4.   //code
5.  } }
```

### Advantage of java inner classes

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

1) Nested classes represent a special type of relationship that is **it can access all the members (data members and methods) of outer class** includingprivate.

2) Nested classes are used **to develop more readable and maintainable code** because it logically group classes and interfaces in one placeonly.

3) **Code Optimization**: It requires less code to write.

### Difference between nested class and inner class in Java

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

### Types of Nested classes

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

- Non-static nested class (innerclass)
  1. Member inner class
  2. Anonymous innerclass
  3. Local inner class
- Static nestedclass

## Java Package

A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql

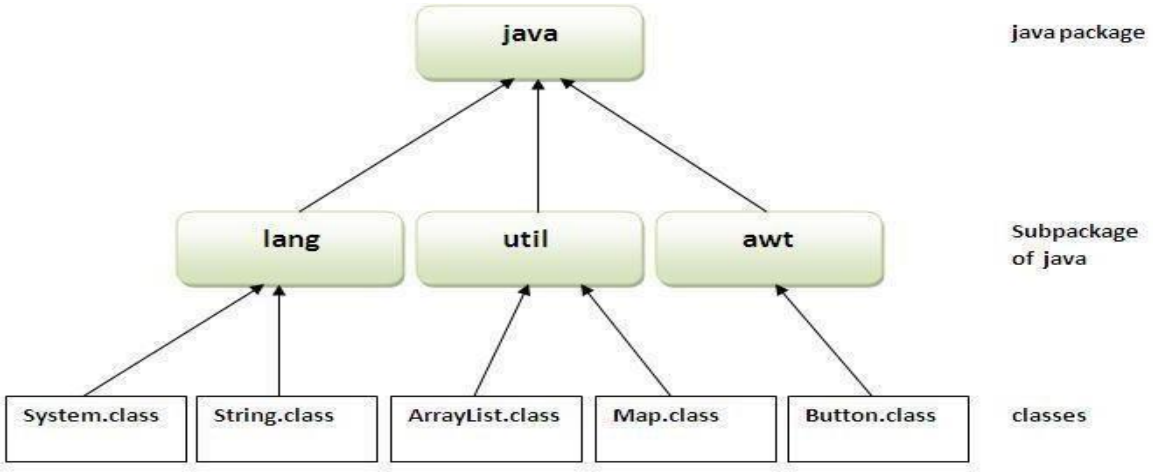
etc.**Advantage of JavaPackage**

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides accessprotection.

3) Java package removes namingcollision.

```
packagemypack;
public class Simple{
 public static void main(String args[]){
   System.out.println("Welcome to package");
  } }
```




How to compile java package

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

javac -d directory javafilename

**How to run java package program**

**To Compile:** javac -d . Simple.java

**To Run:** java mypack.Simple

***Using fully qualified name***

Example of package by import fully qualified name

```
//save by A.java
package pack;
public class A{
  public void msg(){System.out.println("Hello");} }
//save by B.java
package mypack;
class B{
  public static void main(String args[]){
   pack.A obj = new pack.A();//using fully qualified name
   obj.msg();
  }
}
```

Output:Hello

**MALLA REDDY COLLEGE OF ENGINEERING &TECHNOLOGY**
**EPARTMENT OF EEE**

# UNIT-3

# Exception Handling

The **exception handling in java** is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can bemaintained.

## What is exception

In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

## Advantage of Exception Handling

The core advantage of exception handling is **to maintain the normal flow of the application**. Exception normally disrupts the normal flow of the application that is why we use exception handling.

## Types of Exception

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. The sun microsystem says there are three types of exceptions:

1. CheckedException
2. UncheckedException
3. Error

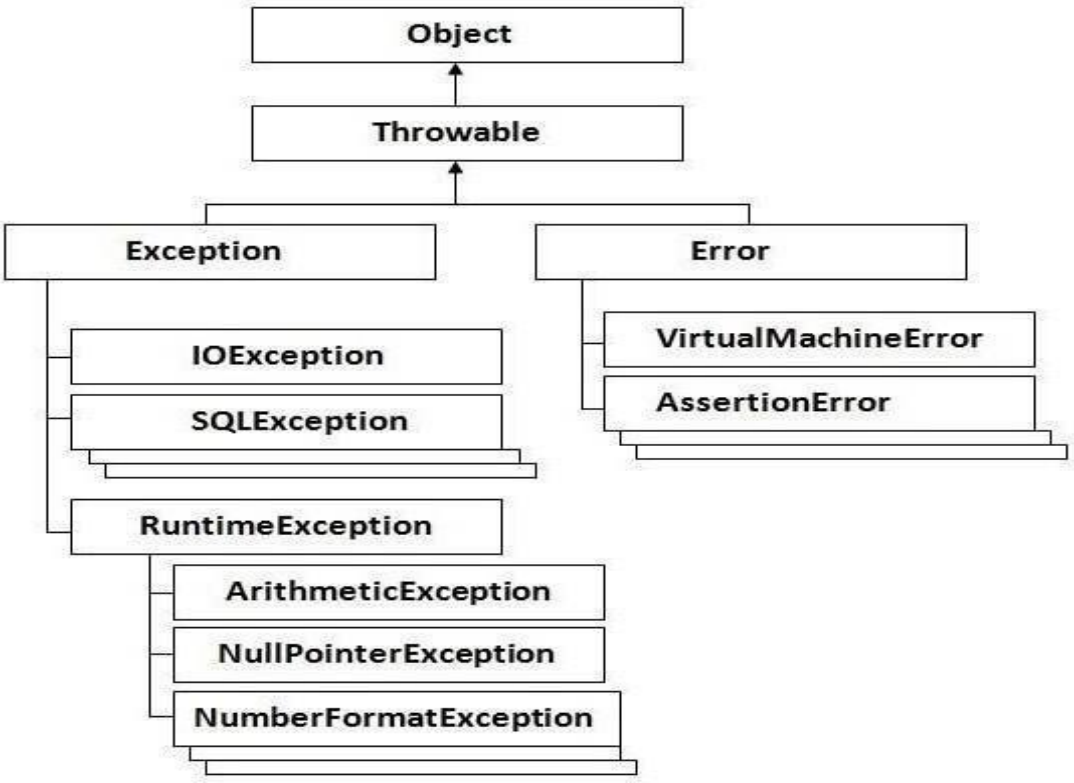
## Difference between checked and unchecked exceptions

**1) Checked Exception: The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException, SQLException etc. Checked exceptions are checked atcompile-time.**

**2) Unchecked Exception: The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checkedatruntime.**

**3) Error: Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionErroretc.**

## Hierarchy of Java Exception classes




## Checked and UnChecked Exceptions

| Checked Exceptions | Unchecked Exceptions |
|---|---|
| • Exception which are checked at Compile time called Checked Exception<br>• If a method throws a checked exception, then the method must either handle the exception or it must specify the exception using throws keyword | • Exceptions whose handling is NOT verified during Compile time.<br>• These exceptions are handled at run-time i.e., by JVM after they occurred by using the try and catch block |
| • Examples:<br>○ IOException<br>○ SQLException<br>○ DataAccessException<br>○ ClassNotFoundException<br>○ InvocationTargetException<br>○ MalformedURLException | • Examples<br>○ NullPointerException<br>○ ArrayIndexOutOfBound<br>○ IllegalArgumentException<br>○ IllegalStateException |

### Java try block

Java try block is used to enclose the code that might throw an exception. It must be used within the method.

Java try block must be followed by either catch or finally block.

### Syntax of java try-catch

```
1. try{
2. //code that may throwexception
3. }catch(Exception_class_Name
```

ref){}Syntax of try-finallyblock

```
1. try{
2. //code that may throwexception
3. }finally{}
```

### Java catch block

Java catch block is used to handle the Exception. It must be used after the try block only.

You can use multiple catch block with a single try.

### Problem without exception handling

Let's try to understand the problem if we don't use try-catch block.

```
public class Testtrycatch1{
  public static void main(String args[]){
    int data=50/0;//may throw exception
    System.out.println("rest of thecode...");
} }
```

Output:

```
Exception in thread main java.lang.ArithmeticException:/ byzero
```

As displayed in the above example, rest of the code is not executed (in such case, rest of the code... statement is not printed).

There can be 100 lines of code after exception. So all the code after exception will not be executed.

### Solution by exception handling

Let's see the solution of above problem by java try-catch block.

```
public class Testtrycatch2{
```

```
 public static void main(String args[]){
  try{
    intdata=50/0;
  }catch(ArithmeticException e){System.out.println(e);}
  System.out.println("rest of the code...");
} }
```

1. Output:

```
Exception in thread main java.lang.ArithmeticException:/ by zero
rest of the code...
```

Now, as displayed in the above example, rest of the code is executed i.e. rest of the code... statement is printed.

### Java Multi catch block

If you have to perform different tasks at the occurrence of different Exceptions, use java multi catch block.

Let's see a simple example of java multi-catch block.

```
1.  public classTestMultipleCatchBlock{
2.    public static void main(Stringargs[]){
3.     try{
4.      inta[]=new int[5];
5.      a[5]=30/0;
6.     }
7.   catch(ArithmeticException e){System.out.println("task1 iscompleted");}
8.   catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");
9. }
10.    catch(Exception e){System.out.println("common taskcompleted");
11. }
12.    System.out.println("rest of the code...");
13. } }
```

```
Output:task1 completed
      rest of the code...
```

### Java nested try example

Let's see a simple example of java nested try block.

```
classExcep6{
 public static void main(String args[]){
  try{t
    ry{
     System.out.println("going to divide");
     intb =39/0;
    }catch(ArithmeticException e){System.out.println(e);}

    try{
```

```
    inta[]=new int[5];
    a[5]=4;
    }catch(ArrayIndexOutOfBoundsException e){System.out.println(e);}
    System.out.println("other statement);
  }catch(Exception e){System.out.println("handeled");}
  System.out.println("normal flow..");
 }
1. }
```

### Java finally block

**Java finally block** is a block that is used *to execute important code* such as closing connection, stream etc.

Java finally block is always executed whether exception is handled or not.

Java finally block follows try or catch block.

### Usage of Java finally

Case 1

Let's see the java finally example where **exception doesn't occur**.

```
classTestFinallyBlock{
  public static void main(String args[]){
  try{
   intdata=25/5;
   System.out.println(data);
  }
  catch(NullPointerException e){System.out.println(e);}
  finally{System.out.println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

```
Output:5
      finally block is always executed
      rest of the code...
```

### Java throw keyword

The Java throw keyword is used to explicitly throw an exception.

We can throw either checked or uncheked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception. We will see custom exceptions later.

The syntax of java throw keyword is given below.

```
1. throwexception;
```

### Java throw keyword example

In this example, we have created the validate method that takes integer value as a parameter. If the age is less than 18, we are throwing the ArithmeticException otherwise print a message welcome to vote.

```
1. public class TestThrow1{
     static void validate(int age){
     if(age<18)
     throw new ArithmeticException("not valid");
     else
     System.out.println("welcome to vote");
     }
     public static void main(String args[]){
     validate(13);
     System.out.println("rest of the code...");
     }}
```

**Output:**

```
Exception in thread main java.lang.ArithmeticException:notvalid
```

### Java throws keyword

The **Java throws keyword** is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing check up before the code being used.

Syntax of java throws

```
1. return_type method_name() throwsexception_class_name{
2. //method
code3. }
4.
```

### Java throws example

Let's see the example of java throws clause which describes that checked exceptions can be propagated by throws keyword.

```
importjava.io.IOException;
classTestthrows1{
voidm()throws IOException{
   throw new IOException("device error");//checked exception
```

```
    }
    voidn()throws IOException{
      m();
    }
    voidp(){
     try{
     n();
     }catch(Exception e){System.out.println("exception handled");}
    }
    public static void main(String args[]){
     Testthrows1 obj=new Testthrows1();
     obj.p();
     System.out.println("normal flow..."); } }
```

Output:

```
exception handled
normal flow...
```

## Java Custom Exception

If you are creating your own Exception that is known as custom exception or user-defined exception. Java custom exceptions are used to customize the exception according to user need.

By the help of custom exception, you can have your own exception and message.

Let's see a simple example of java custom exception.

```
classInvalidAgeException extends Exception{
 InvalidAgeException(String s){
  super(s);
 } }
classTestCustomException1{
  static void validate(int age)throws InvalidAgeException{
    if(age<18)
     throw new InvalidAgeException("not valid");
    else
     System.out.println("welcome to vote");
  }
  public static void main(String args[]){
     try{
     validate(13);
     }catch(Exception m){System.out.println("Exception occured: "+m);}

     System.out.println("rest of the code...");
  }}
```

Output:Exception occured: InvalidAgeException:not valid rest of the code...

# Multithreading

**Multithreading in java** is a process of executing multiple threads simultaneously.

Thread is basically a lightweight sub-process, a smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

But we use multithreading than multiprocessing because threads share a common memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation etc.

## Advantages of Java Multithreading

1) It **doesn't block the user** because threads are independent and you can performmultiple operations at sametime.

2) You **can perform many operations together so it savestime**.

3) Threads are **independent** so it doesn't affect other threads if exception occur in a singlethread.

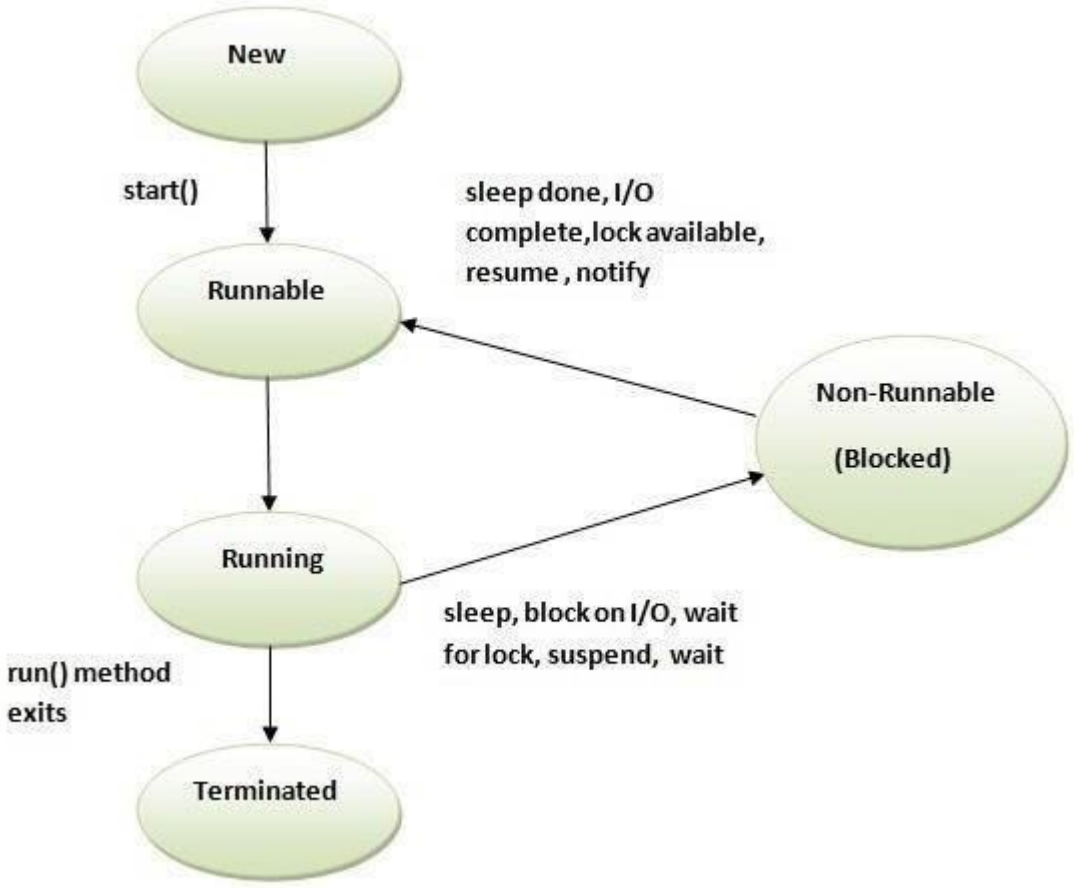
## Life cycle of a Thread (Thread States)

A thread can be in one of the five states. According to sun, there is only 4 states in **thread life cycle in java** new, runnable, non-runnable and terminated. There is no running state.

But for better understanding the threads, we are explaining it in the 5 states.

The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:

1. New
2. Runnable
3. Running
4. Non-Runnable(Blocked)
5. Terminated




## How to create thread

**There are two ways to create a thread:**

1. By extending Threadclass
2. By implementing Runnableinterface.

### Thread class:

Thread class provide constructors and methods to create and perform operations on a thread.Thread class extends Object class and implements Runnable interface.

### Commonly used Constructors of Thread class:

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r,String name)

## Commonly used methods of Thread class:

1. **public void run():** is used to perform action for athread.
2. **public void start():** starts the execution of the thread.JVM calls the run() method onthethread.
3. **public void sleep(long miliseconds):** Causes the currently executing thread to sleep (temporarily cease execution) for the specified number ofmilliseconds.
4. **public void join():** waits for a thread todie.
5. **public void join(long miliseconds):** waits for a thread to die for the specifiedmiliseconds.
6. **public int getPriority():** returns the priority of thethread.
7. **public int setPriority(int priority):** changes the priority of thethread.
8. **public String getName():** returns the name of thethread.
9. **public void setName(String name):** changes the name of the thread.
10. **public Thread currentThread():** returns the reference of currently executingthread.
11. **public int getId():** returns the id of thethread.
12. **public Thread.State getState():** returns the state of thethread.
13. **public boolean isAlive():** tests if the thread isalive.
14. **public void yield():** causes the currently executing thread object to temporarily pause and allow other threads toexecute.
15. **public void suspend():** is used to suspend thethread(depricated).
16. **public void resume():** is used to resume the suspendedthread(depricated).
17. **public void stop():** is used to stop thethread(depricated).
18. **public boolean isDaemon():** tests if the thread is a daemonthread.
19. **public void setDaemon(boolean b):** marks the thread as daemon or userthread.
20. **public void interrupt():** interrupts thethread.
21. **public boolean isInterrupted():** tests if the thread has beeninterrupted.
22. **public static boolean interrupted():** tests if the current thread has beeninterrupted.

## Runnable interface:

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

1. **public void run():** is used to perform action for athread.

## Starting a thread:

**start() method** of Thread class is used to start a newly created thread. It performs following tasks:

- A new thread starts(with new callstack).
- The thread moves from New state to the Runnable state.
- When the thread gets a chance to execute, its target run() method will run.

### Java Thread Example by extending Thread class

```
classMulti extends Thread{
public void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
Multi t1=new Multi();
t1.start();
 } }
```

**Output:**thread isrunning...

### Java Thread Example by implementing Runnable interface

```
class Multi3 implements Runnable{
public void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
Multi3 m1=new Multi3();
Thread t1 =new Thread(m1);
t1.start();
 } }
```

**Output:**thread isrunning...

### Priority of a Thread (Thread Priority):

Each thread have a priority. Priorities are represented by a number between 1 and 10. In most cases, thread schedular schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.

### 3 constants defined in Thread class:

1. public static intMIN_PRIORITY
2. public static intNORM_PRIORITY
3. public static intMAX_PRIORITY

Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

### Example of priority of a Thread:

```
class TestMultiPriority1 extends
Thread{public void run(){
  System.out.println("running thread name is:"+Thread.currentThread().getName());
  System.out.println("running thread priority is:"+Thread.currentThread().getPriority());
  }
 public static void main(String args[]){
```

```
TestMultiPriority1 m1=new TestMultiPriority1();
TestMultiPriority1 m2=new TestMultiPriority1();
m1.setPriority(Thread.MIN_PRIORITY);
m2.setPriority(Thread.MAX_PRIORITY);
m1.start();
m2.start();
} }
```

**Output:**running thread name is:Thread-0
running thread priority is:10
running thread name is:Thread-1
running thread priority is:1

## Java synchronized method

If you declare any method as synchronized, it is known as synchronized method.

Synchronized method is used to lock an object for any shared resource.

When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

## Example of inter thread communication in java

Let's see the simple example of inter thread communication.

```
classCustomer{
intamount=10000;
synchronized void withdraw(int amount){
System.out.println("going to withdraw...");
if(this.amount<amount){
System.out.println("Less balance; waiting for deposit...");
try{wait();}catch(Exception e){}
}
this.amount-=amount;
System.out.println("withdraw completed...");
}
synchronized void deposit(int amount){
System.out.println("going to deposit...");
this.amount+=amount;
System.out.println("deposit completed... ");
notify();
}
}
classTest{
public static void main(String args[]){
final Customer c=newCustomer();
new Thread(){
public void run(){c.withdraw(15000);}
}.start();
newThread(){
```

```
public void run(){c.deposit(10000);}
}
start();
}}
```

```
Output: going to withdraw...
     Less balance; waiting for deposit...
     going to deposit...
     deposit completed...
     withdraw completed
```

## ThreadGroup in Java

Java provides a convenient way to group multiple threads in a single object. In such way, we can suspend, resume or interrupt group of threads by a single method call.

> ***Note: Now suspend(), resume() and stop() methods are deprecated.***

Java thread group is implemented by *java.lang.ThreadGroup*

class.Constructors of ThreadGroup class

There are only two constructors of ThreadGroup class.

ThreadGroup(String name)
ThreadGroup(ThreadGroup parent, String name)

Let's see a code to group multiple threads.

```
1. ThreadGroup tg1 = new ThreadGroup("GroupA");
2. Thread t1 = new Thread(tg1,newMyRunnable(),"one");
3. Thread t2 = new Thread(tg1,newMyRunnable(),"two");
4. Thread t3 = new Thread(tg1,newMyRunnable(),"three");
```

Now all 3 threads belong to one group. Here, tg1 is the thread group name, MyRunnable is the class that implements Runnable interface and "one", "two" and "three" are the thread names.

Now we can interrupt all threads by a single line of code only.

```
1. Thread.currentThread().getThreadGroup().interrupt();
```

**MALLA REDDY COLLEGE OF ENGINEERING &TECHNOLOGY**
**DEPARTMENT OF EEE**

**UNIT 4**

**UNIT IV: Applets –** Concepts of Applets, differences between applets and applications, life cycle of an applet, types of applets, creating applets, passing parameters to applets.
**Event Handling-** Events, Event sources, Event classes, Event Listeners, Delegation event model, handling mouse and keyboard events, Adapter classes.

Streams- Byte streams, Character streams, Text input/output.

## APPLETS:

An applet is a program that comes from server into a client and gets executed at client side and displays the result.

An applet represents byte code embedded in a html page. (Applet = bytecode + html) and run with the help of Java enabled browsers such as Internet Explorer.

An applet is a Java program that runs in a browser. Unlike Java applications applets do not have a main () method.

To create applet we can use java.applet.Applet or javax.swing.JApplet class. All applets inherit the super class „Applet". An Applet class contains several methods that help to control the execution of an applet.

**Advantages:**

ιϖ. Applets provide dynamic nature for a webpage.
ϖ. Applets are used in developing games and animations.
ϖι. Writing and displaying (browser) graphics and animations is easier than applications.
ϖιι. In GUI development, constructor, size of frame, window closing code etc. are not required

**Restrictions of Applets of Applets Vs Applications**

6. Applets are required separate compilation before opening in a browser.
7. In realtime environment, the bytecode of applet is to be downloaded from the server to the client machine.
8. Applets are treated as **untrusted** (as they were developed by unknown people and placedon unknown servers whose trustworthiness is not guaranteed).
9. Extra Code is required to communicate between applets using **AppletContext**.

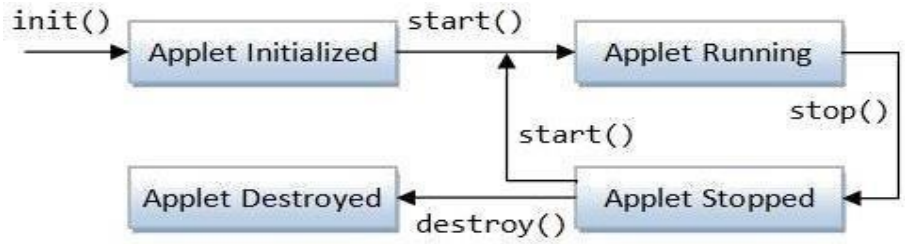
**DEFFERENCES BETWEEN APPLETS AND APPLICATIONS**

| FEATURE | APPLICATION | APPLET |
|---|---|---|
| **main() method** | main() method Present | main() method Not present |
| **Execution** | Can be executed on standalone computer system. (JDK & JRE) | Used to run a program on client Browser like Chrome. |
| **Nature** | Called as stand-alone application as application can be executed from command prompt | Requires some third party tool help like a browser to execute |

| | | |
|---|---|---|
| **Restrictions** | Can access any data or software available on the system | Cannot access anything on the system except browser‟s services |
| **Security** | Does not require any security | Requires highest security for the system as they are untrusted |
| **Programming** | larger programs | small programs |
| **Platform** | platform independent | platform independent |
| **Accessibility** | The java applications are designed to work with the client as well as server. | Applets are designed just for handling the client site problems. |
| **Working** | Applications are created by writing public static void main(String[] s) method | Applets are created by extending thejava.applet.Applet class |
| **Client side / Server side** | The applications don't have such type of criteria | Applets are designed for the client site programming purpose |
| **Methods** | Application has a single start point which is main method | Applet application has 5 methods which will be automatically invoked. |
| **Example** | public class MyClass | import java.awt.*; |
| | { | import java.applet.*; |
| | public static void main(String | |
| | args[]) {} | public |
| | } | class Myclass extends Applet |
| | | { |
| | | public void init() |
| | | { |
| | | } |
| | | public void start() |
| | | { |
| | | } |
| | | public void stop() |
| | | { |
| | | } |
| | | public void destroy() {} |
| | | public void paint(Graphics g) {} |
| | | } |

LIFE CYCLE OF AN APPLET

Let the Applet class extends Applet or JApplet class.




Initialization:

**public void init():** This method is used for initializing variables, parameters to create components. This method is executed only once at the time of applet loaded into memory.

```
public void init()
{
//initialization
}
```

Runnning:

**public void start ():** After init() method is executed, the start method is executed automatically. Start method is executed as long as applet gains focus. In this method code related to opening files and connecting to database and retrieving the data and processing the data is written.

Idle / Runnable:

**public void stop ():** This method is executed when the applet loses focus. Code related to closing the files and database, stopping threads and performing clean up operations are written in this stop method.

Dead/Destroyed:

**public void destroy ():** This method is executed only once when the applet is terminated from the memory.

Executing above methods in that sequence is called applet life cycle.

We can also use **public void paint (Graphics g)** in applets.

**//An Applet skeleton.**

```
import java.awt.*;
import java.applet.*;
/*
<applet code="AppletSkel" width=300 height=100>
</applet>
*/
public class AppletSkel extends Applet {
 3 Called first.
    public void
    init() {
 4 initialization
}
```

```
/* Called second, after init(). Also called whenever the applet is restarted. */
public void start() {
3  start or resume execution
}
4  Called when the applet is
   stopped. public void stop() {
5  suspends execution
}
/* Called when applet is terminated. This is the last method executed. */
public void destroy() {
   5   perform shutdown activities
}
   6   Called when an applet's window must be
       restored. public void paint(Graphics g) {
   7   redisplay contents of window
}
}
```

After writing an applet, an applet is compiled in the same way as Java application but running of an applet is different.

**There are two ways to run an applet.**

- Executing an applet within a Java compatible web browser.
- Executing an applet using „appletviewer". This executesthe applet in a window.

To execute an applet using web browser, we must write a small HTML file which contains the appropriate „APPLET" tag. <APPLET> tag is useful to embed an applet into an HTML page. It has the following form:

*<APPLET CODE="name of the applet class file" HEIGHT = maximum height of applet in pixels WIDTH = maximum width of applet in pixels ALIGN = alignment (LEFT, RIGHT, MIDDLE, TOP, BOTTOM)>*
*<PARAM NAME = parameter name VALUE = its value> </APPLET>*

**Execution:** ***appletviewer programname.java or appletviewer programname.html***

The <PARAM> tag useful to define a variable (parameter) and its value inside the HTML page which can be passed to the applet. The applet can access the parameter value using **getParameter () method**, as: String value = getParameter ("pname");

**Example Program:**

Following is a simple applet named **HelloWorldApplet.java** −

```
import java.applet.*;
import java.awt.*;
public class HelloWorldApplet extends Applet {
  public void paint (Graphics g) {
    g.drawString ("Hello World", 25, 50);
  } }
```

**Invoking an Applet - HelloWorldApplet.html**

```
<html>
  <title>The Hello, World Applet</title>
    <applet code = "HelloWorldApplet.class" width = "320" height = "120">
  </applet>
</html>
```

**OUTPUT**: javac HelloWorldApplet.java
appletviewer HelloWorldApplet.html

Embedded an Applet

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
/*
<applet code="First.class" width="300" height="300">
</applet>
*/
public class First extends Applet
{
  public void paint(Graphics g){
g.drawString("welcome to applet",150,150);
    }
  }
```

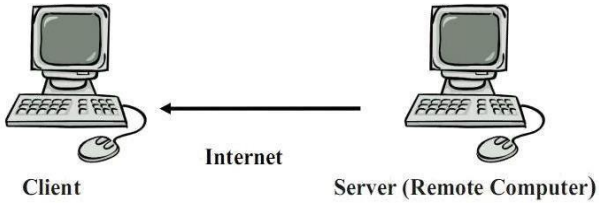
**OUTPUT**: javac First.java
appletviewer First.java

**TYPES OF APPLETS**

Applets are of two types:
// Local Applets
// Remote Applets

**Local Applets:** An applet developed locally and stored in a local system is called local applets. So, local system does not require internet. We can write our own applets and embed them into the web pages.

**Remote Applets:** The applet that is downloaded from a remote computer system and embed applet into a web page. The internet should be present in the system to download the applet and run it. To download the applet we must know the applet address on web known as Uniform Resource Locator(URL) and must be specified in the applets HTML document as the value of CODEBASE.



## PASSING PARAMETERS TO AN APPLET

Java applet has the feature of retrieving the parameter values passed from the html page. So, you can pass the parameters from your html page to the applet embedded in your page. The **param** tag(**<parma name="" value=""></param**>) is used to pass the parameters to an applet. The applet has to call the getParameter() method supplied by the java.applet.Applet parent class.

**Ex1: Write a program to pass employ name and id number to an applet.**

```
import java.applet.*;
import java.awt.*;
/* <applet code="MyApplet2.class" width = 600 height= 450>
<param name = "t1" value="Hari Prasad"> <param name =
"t2" value ="101">
</applet> */

public class MyApplet2 extends Applet
{
String n;
String id;
public void init()
{
  n = getParameter("t1");
  id = getParameter("t2");
}
public void paint(Graphics g)
{
                    rawString("Name is : "
                    + n, 100,100);
                    g.drawString("Id is :
                    "+ id, 100,150);
}
}
```


**Ex2: Write a program to pass two numbers and pass result to an applet.**

```
import java.awt.*;
import java.applet.*;
  /*<APPLET code="Pp" width="300"
height="250"> <PARAM name="a" value="5">
<PARAM name="b" value="5">
</APPLET>*/
public class Pp extends Applet
{
      String  str;
      int a,b,result;
public void init()
 {
          str=getParameter("a");
             a=Integer.parseInt(str);
         str=getParameter("b");
```

```
            b=Integer.parseInt(str);
          result=a+b;
          str=String.valueOf(result);
        }
  public void paint(Graphics g)
   {
         g.drawString(" Result of Addition is : "+str,0,15);
    }
}
```


**Ex3: Hai.java**

```
import java.applet.*;
import java.awt.*;
/*<Applet code="hai" height="250" width="250">
   <PARAM name="Message" value="Hai friend how are you ..?"></APPLET>
*/
 class hai extends Applet
  {
    private String defaultMessage = &amp;quot;Hello!&amp;quot;;
   public void paint(Graphics g) {

    String inputFromPage = this.getParameter(&amp;quot;Message&amp;quot;); if
    (inputFromPage == null) inputFromPage = defaultMessage;
    g.drawString(inputFromPage, 50, 55);
  }
}
```

**Output:**

# EVENT HANDLING

Event handling is at the core of successful applet programming. Most events to which the applet will respond are generated by the user. The most commonly handled events are those generated by the mouse, the keyboard, and various controls, such as a push button.

Events are supported by the **java.awt.event** package.

## The Delegation Event Model

The modern approach to handling events is based on the delegation event model, which defines standard and consistent mechanisms to generate and process events.

Its concept is quite simple: a source generates an event and sends it to one or more listeners. In this scheme, the listener simply waits until it receives an event. Once received, the listener processes the event and then returns.

The advantage of this design is that the application logic that processes events is cleanly separated from the user interface logic that generates those events. A user interface element is able to "delegate" the processing of an event to a separate piece of code.

In the delegation event model, listeners must register with a source in order to receive an event notification. This provides an important benefit: notifications are sent only to listeners that want to receive them.

### EVENTS

In the delegation model, an event is an object that describes a state change in a source. It can be generated as a consequence of a person interacting with the elements in a graphical user interface. Some of the activities that cause events to be generated are pressing a button, entering a character via the keyboard, selecting an item in a list, and clicking the mouse.

Events may also occur that are not directly caused by interactions with a user interface.
**For example**, an event may be generated when a timer expires, a counter exceeds a value, software or hardware failure occurs, or an operation is completed.

### EVENT SOURCES

A source is an object that generates an event. This occurs when the internal state of that object changes in some way. Sources may generate more than one type of event. A source must register listeners in order for the listeners to receive notifications about a specific type of event. Each type of event has its own registration method.

**Here is the general form:**

```
public void add Type Listener( Type Listener el )
```

### EVENT LISTENERS

A listener is an object that is notified when an event occurs. It has two major requirements. First, it must have been registered with one or more sources to receive notifications about specific types of events. Second, it must implement methods to receive and process these notifications. The methods that receive and process events are defined in a set of interfaces found in **java.awt.event**.

**For example**, the MouseMotionListener interface defines two methods to receive notifications when the mouse is dragged or moved.

EVENT CLASSES

The classes that represent events are at the core of Java's event handling mechanism. At the root of the Java event class hierarchy is EventObject, which is in **java.util**. It is the superclass for all events.

**It's one constructor is shown here:**
EventObject(Object src )

**EventObject contains two methods:** getSource( ) and toString( ) .

The **getSource( ) method** returns the source of the event. **Ex:** Object getSource( )
**toString( )** returns the string equivalent of the event.

The package **java.awt.event** defines several types of events that are generated by various user interface elements.

| Event Class | Description |
|---|---|
| ***ActionEvent*** | Generated when a button is pressed, a list item is double-clicked, or a menu item is selected. |
| ***AdjustmentEvent*** | Generated when a scroll bar is manipulated. |
| ***ComponentEvent*** | Generated when a component is hidden, moved, resized or becomes visible. |
| ***ContainerEvent*** | Generated when a component is added to or removed from a container. |
| ***FocusEvent*** | Generated when a component gains or loses keyboard focus. |
| ***InputEvent*** | Abstract super class for all component input event classes. |
| ***ItemEvent*** | Generated when a check box or list item is clicked; so occurs when a choice selection is made or a checkable menu item is selected or deselected. |
| ***KeyEvent*** | Generated when input is received from the keyboard. |
| ***MouseEvent*** | Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component. |
| ***MouseWheelEvent*** | Generated when the mouse wheel is moved. (Added by Java 2, version 1.4) |
| ***TextEvent*** | Generated when the value of a text area or text field is changed. |
| ***WindowEvent*** | Generated when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit. |

### The ActionEvent Class

An ActionEvent is generated when a button is pressed, a list item is double-clicked, or a menu item is selected. The ActionEvent class defines four integer constants that can be used to identify any modifiers associated with an action event: ALT_MASK , CTRL_MASK , META_MASK , and SHIFT_MASK .

### ActionEvent has these three constructors:

ActionEvent(Object src , int type , String cmd )
ActionEvent(Object src , int type , String cmd , int modifiers )
ActionEvent(Object src , int type, String cmd, long when , int modifiers )

The ComponentEvent Class

// ComponentEvent is generated when the size, position, or visibility of a component is changed. There are four types of component events. The constants and their meanings are shown here:

COMPONENT_HIDDEN The component was hidden.
COMPONENT_MOVED The component was moved.
COMPONENT_RESIZED The component was resized.
COMPONENT_SHOWN The component became visible.

The ContainerEvent Class

//ContainerEvent is generated when a component is added to or removed from a container. There are two types of container events.

→

COMPONENT_ADDED and
COMPONENT_REMOVED **The KeyEvent Class**

//KeyEvent is generated when keyboard input occurs. There are three types of key events, which are identified by these integer constants: KEY_PRESSED, KEY_RELEASED, and KEY_TYPED .

The first two events are generated when any key is pressed or released. The last event occurs only when a character is generated.

**The MouseEvent Class**

There are **eight types of mouse events**. The MouseEvent class defines the following integer constants that can be used to identify them:

| | |
|---|---|
| MOUSE_CLICKED | The user clicked the mouse. |
| MOUSE_DRAGGED | The user dragged the mouse. |
| MOUSE_ENTERED | The mouse entered a component. |
| MOUSE_EXITED | The mouse exited from a component. |
| MOUSE_MOVED | The mouse moved. |
| MOUSE_PRESSED | The mouse was pressed. |
| MOUSE_RELEASED | The mouse was released. |
| MOUSE_WHEEL | The mouse wheel was moved (Java 2, v1.4). |

**The WindowEvent Class**

There are **ten types of window events**. The WindowEvent class defines integer constants that can be used to identify them. The constants and their meanings are shown here:

| | |
|---|---|
| WINDOW_ACTIVATED | The window was activated. |
| WINDOW_CLOSED | The window has been closed. |
| WINDOW_CLOSING | The user requested that the window beclosed. |
| WINDOW_DEACTIVATED | The window was deactivated. |
| WINDOW_DEICONIFIED | The window was deiconified. |
| WINDOW_GAINED_FOCUS | The window gained input focus. |
| WINDOW_ICONIFIED | The window was iconified. |
| WINDOW_LOST_FOCUS | The window lost input focus. |
| WINDOW_OPENED | The window was opened. |
| WINDOW_STATE_CHANGED | The state of the window changed. |

When an event occurs, the event source invokes the appropriate method defined by the listener and provides an event object as its argument

| Interface | Description |
|---|---|
| **ActionListener** | Defines one method to receive action events. |
| **AdjustmentListener** | Defines one method to receive adjustment events. |
| **ComponentListener** | Defines **four methods** to recognize when a component is hidden, moved, resized, or shown. |
| **ContainerListener** | Defines **two methods** to recognize when a component is added to or removed from a container. |
| **FocusListener** | Defines **two methods** to recognize when a component gains or losses keyboard focus. |
| **ItemListener** | Defines one method to recognize when the state of an item changes. |
| **KeyListener** | Defines **three methods** to recognize when a key is pressed, released, or typed. |
| **MouseListener** | Defines **five methods** to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released. |
| **MouseMotionListener** | Defines **two methods** to recognize when the mouse is dragged or moved. |
| **MouseWheelListener** | Defines one method to recognize when the mouse wheel is moved. |
| **TextListener** | Defines one method to recognize when a text value changes. |
| **WindowListener** | Defines **seven methods** to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit. |

**The delegation event model has two parts:** sources and listeners. Listeners are created by implementing one or more of the interfaces defined by the **java.awt.event** package.

## The ActionListener Interface

This interface defines the actionPerformed( ) method that is invoked when an action event occurs.

**Its general form is shown here:** void actionPerformed(ActionEvent ae )

### The ItemListener Interface

This interface defines the itemStateChanged( ) method that is invoked when the state of an item changes.

**Its general form is shown here:** void itemStateChanged(ItemEvent ie )

### The KeyListener Interface

This interface defines three methods. The keyPressed( ) and keyReleased( ) methods are invoked when a key is pressed and released, respectively. The keyTyped( ) method is invoked when a character has been entered. For example, if a user presses and releases the key, three events are generated in A sequence: key pressed, typed, and released.

**The general forms of these methods are shown here:**
void keyPressed(KeyEvent ke )
void keyReleased(KeyEvent ke )
void keyTyped(KeyEvent ke )

The MouseListener Interface

This interface defines five methods. If the mouse is pressed and released at the same point, mouseClicked( ) is invoked. When the mouse enters a component, the mouseEntered( ) method is called. When it leaves, mouseExited( ) is called. The mousePressed( ) and mouseReleased( ) methods are invoked when the mouse is pressed and released, respectively.

**The general forms of these methods are shown here:**

```
void mouseClicked(MouseEvent me )
void mouseEntered(MouseEvent me )
void mouseExited(MouseEvent me )
void mousePressed(MouseEvent me )
void mouseReleased(MouseEvent me )
```

The MouseMotionListener Interface

This interface defines two methods. The mouseDragged( ) method is called multiple times as the mouse is dragged. The mouseMoved( ) method is called multiple times as the mouse is moved.

**Their general forms are shown here:**

```
void mouseDragged(MouseEvent me )
void mouseMoved(MouseEvent me )
```

The TextListener Interface

This interface defines the textChanged( ) method that is invoked when a change occurs in a text area or text field.

**Its general form is shown here:** void textChanged(TextEvent te )

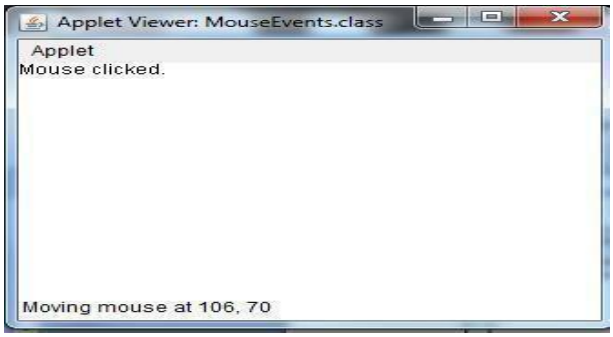
## Handling Mouse Events

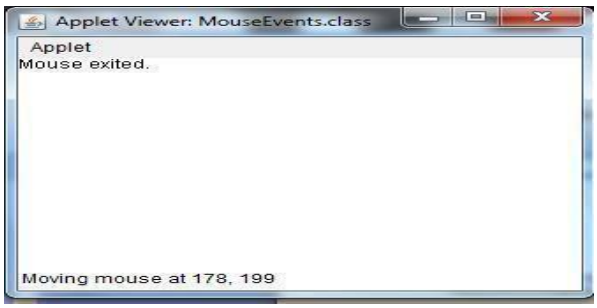
To handle mouse events, we must implement the MouseListener and the MouseMotion Listener interfaces.

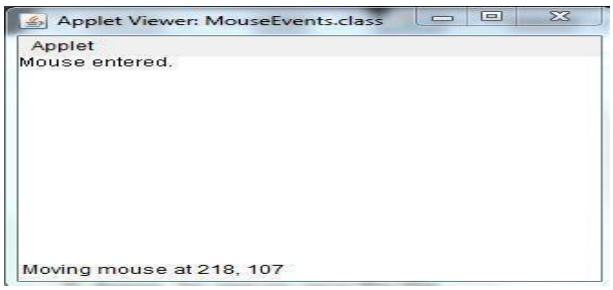
**EX:**

```
// Demonstrate the mouse event handlers.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="MouseEvents" width=300 height=100>
</applet>
*/
public class MouseEvents extends Applet implements MouseListener, MouseMotionListener
{ String msg = "";
int mouseX = 0, mouseY = 0; // coordinates of mouse
public void init() {
addMouseListener(this);
addMouseMotionListener(this);
}
```

```
// Handle mouse clicked.
public void mouseClicked(MouseEvent me) {
mouseX = 0;      // save coordinates
mouseY = 10;
msg = "Mouse clicked.";
repaint();
}

// Handle mouse entered.
public void mouseEntered(MouseEvent me) {
   // save
      coordinates
      mouseX = 0;
      mouseY = 10;
msg = "Mouse entered.";
repaint();
}

   // Handle mouse exited.
public void mouseExited(MouseEvent me) {
      // save
         coordinate
         s mouseX
         = 0;
         mouseY =
         10;
msg = "Mouse exited.";
repaint();
}

      // Handle button pressed.
public void mousePressed(MouseEvent me) {
// save coordinates
   mouseX =
   me.getX();
   mouseY =
   me.getY(); msg =
   "Down"; repaint();
}

// Handle button released.
public void mouseReleased(MouseEvent me) {
   1. save
      coordinates
      mouseX =
      me.getX();
      mouseY =
      me.getY(); msg
      = "Up";
      repaint();
}
```



2. Handle mouse dragged.

```
public void mouseDragged(MouseEvent me) {
```

1. save coordinates

```
   mouseX =
   me.getX();
```

```
mouseY = me.getY();
msg = "*";
showStatus("Dragging mouse at " + mouseX + ", " + mouseY);
repaint();
}

// Handle mouse moved.
public void mouseMoved(MouseEvent me) {
// show status
showStatus("Moving mouse at " + me.getX() + ", " + me.getY());
}

// Display msg in applet window at current X,Y location.
  public void paint(Graphics g) {
g.drawString(msg, mouseX, mouseY);
}
}
```

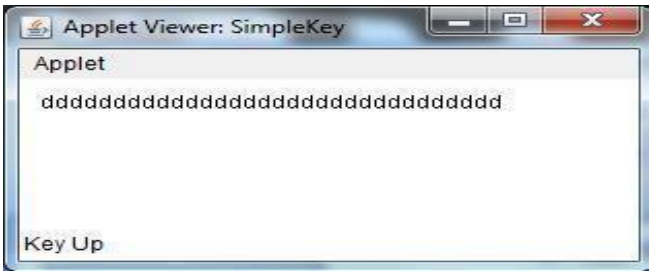
Handling Keyboard Events

When a key is pressed, a **KEY_PRESSED** event is generated. This results in a call to the keyPressed( ) event handler. When the key is released, a **KEY_RELEASED** event is generated and the keyReleased( ) handler is executed. If a character is generated by the keystroke, then a **KEY_TYPED** event is sent and the keyTyped( ) handler is invoked.

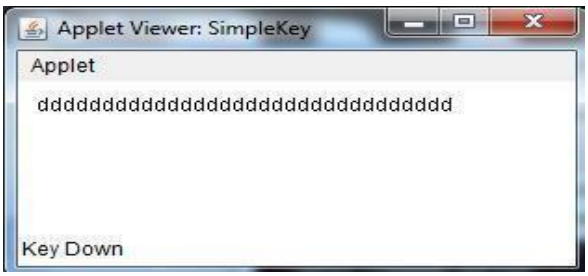
Thus, each time the user presses a key, at least two and often three events are generated. If all you care about are actual characters, then you can ignore the information passed by the key press and release events.

**EX:**

```
// Demonstrate the key event handlers.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/* <applet code="SimpleKey" width=300
height=100> </applet> */
public class SimpleKey extends Applet implements KeyListener
  {
    String msg = "";
    int X = 10, Y = 20; // output coordinates
    public void init() {
    addKeyListener(this);
    requestFocus(); // request input focus
}

 public void keyPressed(KeyEvent ke)
{ showStatus("Key Down"); }

 public void keyReleased(KeyEvent ke) {
showStatus("Key Up");

}
```



```
public void keyTyped(KeyEvent ke) {
msg += ke.getKeyChar();
repaint();
}

// Display keystrokes.
public void paint(Graphics g)
{ g.drawString(msg, X, Y); }

}
```




Adapter Classes

Java provides a special feature, called an adapter class , that can simplify the creation of event handlers in certain situations. An adapter class provides an empty implementation of all methods in an event listener interface.

Adapter classes are useful when you want to receive and process only some of the events that are handled by a particular event listener interface.

**For example,** the MouseMotionAdapter class has two methods, mouseDragged( ) and mouseMoved( ) . The signatures of these empty methods are exactly as defined in the MouseMotionListener interface. If you were interested in only mouse drag events, then you could simply extend MouseMotionAdapter and implement mouseDragged( ) . The empty implementation of mouseMoved( ) would handle the mouse motion events for you.

| **Adapter Class** | **Listener Interface** |
|---|---|
| ComponentAdapter | ComponentListener |
| ContainerAdapter | ContainerListener |
| FocusAdapter | FocusListener |
| KeyAdapter | KeyListener |
| MouseAdapter | MouseListener |
| MouseMotionAdapter | MouseMotionListener |
| WindowAdapter | WindowListener |

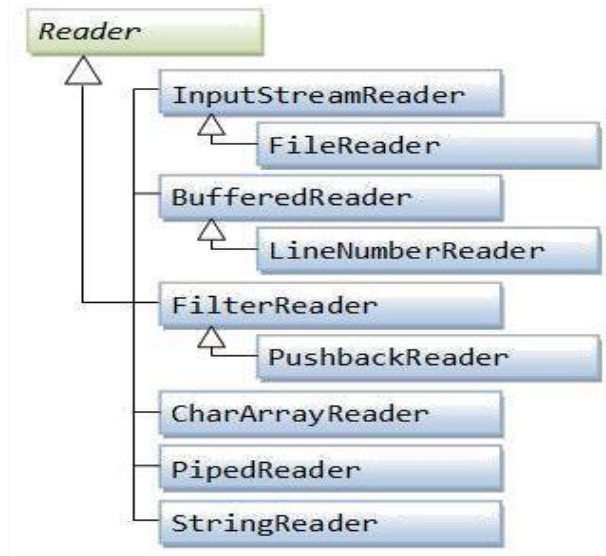
**EX:** // Demonstrate an adapter.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="AdapterDemo" width=300 height=100>
</applet>
*/

public class AdapterDemo extends Applet
{
public void init() {
addMouseListener(new MyMouseAdapter(this));
addMouseMotionListener(new
MyMouseMotionAdapter(this)); }
}
```

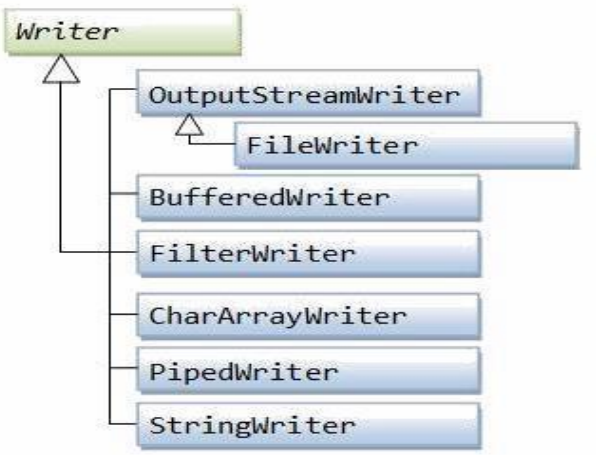
```
class MyMouseAdapter extends MouseAdapter
{
AdapterDemo adapterDemo;
public MyMouseAdapter(AdapterDemo adapterDemo)
{
   this.adapterDemo = adapterDemo;
}

// Handle mouse clicked.
public void mouseClicked(MouseEvent me) {
adapterDemo.showStatus("Mouse clicked");
}
}

class MyMouseMotionAdapter extends
 MouseMotionAdapter  {
 AdapterDemo adapterDemo;
   public MyMouseMotionAdapter(AdapterDemo adapterDemo)
    {
       this.adapterDemo = adapterDemo;
     }

// Handle mouse dragged.
public void mouseDragged(MouseEvent me)
 {
    adapterDemo.showStatus("Mouse dragged");
   }
  }
```

# Files and Streams:

**Text stream classes for reading data**




**Text stream classes for writing data**



## Stream

A stream can be defined as a sequence of data. There are two kinds of Streams −

// **InPutStream** − The InputStream is used to read data from a source.

// **OutPutStream** − The OutputStream is used for writing data to a destination.


Java provides strong but flexible support for I/O related to files and networks.

## Byte Streams

Java byte streams are used to perform **input and output of 8-bit bytes**. Though there are many classes related to byte streams but the most frequently used classes are, **FileInputStream** and **FileOutputStream**.

**Example**

```
import  java.io.*;
public class CopyFile {
  public static void main(String args[]) throws IOException {
    FileInputStream in = null;
    FileOutputStream out = null;
    try {
      in = new FileInputStream("input.txt");
      out = new FileOutputStream("output.txt");
      int c;
      while ((c = in.read()) != -1) {
        out.write(c);
      }
    }finally {
      if (in != null) {
        in.close();
      }
      if (out != null) {
        out.close();
      }     }    } }
```

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

```
This is test for copy file.
```

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

Character Streams

Java **Byte** streams are used to perform input and output of 8-bit bytes, whereas Java **Character** streams are used to perform **input and output for 16-bit unicode**. Though there are many classes related to character streams but the most frequently used classes are, **FileReader** and **FileWriter**.

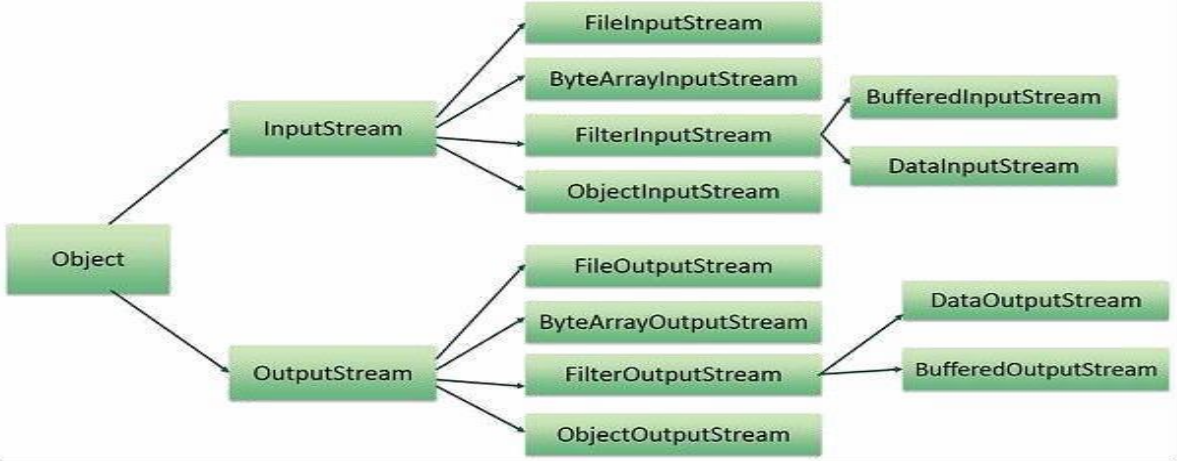
Though internally FileReader uses FileInputStream and FileWriter uses FileOutputStream but here the major difference is that FileReader reads two bytes at a time and FileWriter writes two bytes at a time.

**Example**

```
import  java.io.*;
public class CopyFile {

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

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

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

## Reading and Writing Files Text input/output,

As described earlier, a stream can be defined as a sequence of data. The **InputStream** is used to read data from a source and the **OutputStream** is used for writing data to a destination. Here is a hierarchy of classes to deal with Input and Output streams.



The two important streams are **FileInputStream** and **FileOutputStream** (File Handling)

In Java, FileInputStream and FileOutputStream classes are used to read and write data in file. In another words, they are used for file handling in java.

Java FileOutputStream class

Java FileOutputStream is an output stream for writing data to a file. It is a class belongs to **byte streams.** It can be used to create text files.

→

First we should read data from the keyword. It uses **DataInputStream** class for reading data from the keyboard is as:

**DataInputStream dis=new DataInputStream(System.in);**

→

FileOutputStream **used to send data to the file and attaching the file to FileOutputStream.** i.e., **FileOutputStream fout=new FileOutputStream("File_name");**

→

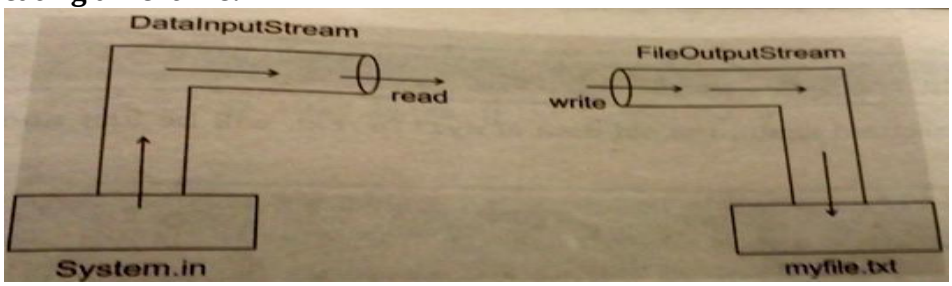
The next step is to read data from DataInputStream and write it into FileOutputStream. It means read data from **dis object** and write it into **fout object. i.e.,**

```
ch=(char)dis.read(); //read one character into ch
fout.write(ch); //write ch into file.
```

→

**Finally** closing the file using: **fout.close();**

**Creating a Text file:**




**Example: Write a program to read data from the keyboard and write it to myfile.txt file.**

```
import java.io.*;
class Test{
  public static void main(String args[])
    {
     DataInputStream dis=new DataInputStream(System.in);
     FileOutputstream fout=new
     FileOutputStream("myfile.txt"); System.out.println("Enter
     text @ at the end:"); char ch;
     while((ch=(char)dis.read())!="@")
       fout.write(ch);
       fout.close();
  }
}
```

**Output:** javac Test.java

Java Test

```
Enter text @ at the end:
This is my file line one
This is my file line two@
```

Java FileInputStream class

It is useful to read data from a file in the form of sequence of bytes. It is possible to read data from a text file using FileInputStream. i.e.,

**FileInputStream fin= new FileInputStream("myfile.txt");**

To read data from the file is, **ch=fin.read();**

When there is no more data available to read then it returns **-1**.

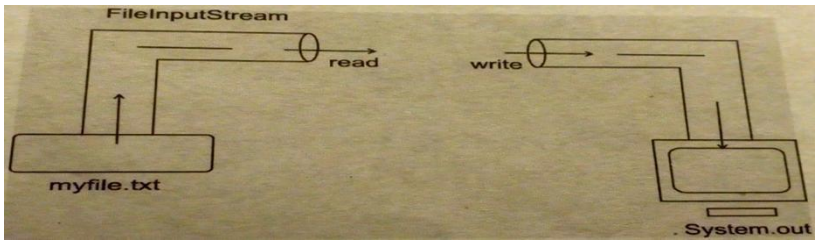
→

The Output Stream is used to send data to the **monitor**. i.e., **PrintStream,** for displaying the data we can use **System.out.**

System.out.print(ch);

## Reading data from a text file using FileInputStream:

Java FileInputStream class obtains input bytes from a file. It is used for reading streams of raw bytes such as image data. It should be used to read byte-oriented data for example to read image, audio, video etc.




**Example: Write a program to read data from myfile.txt using FileInputStream and display it on monitor.**

```
import java.io.*;
class ReadFile
{
 public static void main(String args[])
  {

   FileInputStream fin=new FileInputStream("myfile.txt");
   System.out.println("File Contents:"); int ch;
   while((ch=fin.read())!=-1)
   {
    System.out.println((char)ch);
   }
   fin.close();
  }
 }
```

**Output:** javac ReadFile.java
java ReadFile

**UNIT V: GUI Programming with Java –** AWT class hierarchy, component, container, panel, window, frame, graphics.

**AWT controls:** Labels, button, text field, check box, check box groups, choices, lists, scrollbars, and graphics.

**Layout Manager –** Layout manager types: border, grid and flow.

**Swing –** Introduction, limitations of AWT, Swing vs AWT.

## GUI PROGRAMMING WITH JAVA

### ABSTRACT WINDOW TOOLKIT (AWT)

**Java AWT** (Abstract Window Toolkit) is *an **API to develop GUI** or window-based application in java*. Java AWT components are **platform-dependent** i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components uses the resources of system. The Abstract Window Toolkit(AWT) support for applets. The AWT contains numerous classes and methods that allow you to create and manage windows.

The **java.awt package** provides classes for AWT api such as **TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.**

**AWT Classes**

The AWT classes are contained in the **java.awt package**. It is one of Java's largest packages.

| Class | Description |
|---|---|
| AWTEvent | Encapsulates AWT events. |
| AWTEventMulticaster | Dispatches events to multiple listeners. |
| BorderLayout | Border layouts use five components: North, South, East, West, and Center. |
| CardLayout | Card layouts emulate index cards. Only the one on top is showing. |
| Checkbox | Creates a check box control. |
| CheckboxGroup | Creates a group of check box controls. |
| CheckboxMenuItem | Creates an on/off menu item. |
| Choice | Creates a pop-up list. |
| Color | Manages colors in a portable, platform-independent fashion. |
| Component | An abstract superclass for various AWT components. |
| Container | A subclass of Component that can hold other components. |
| Cursor | Encapsulates a bitmapped cursor. |
| Dialog | Creates a dialog window. |
| Dimension | Specifies the dimensions of an object. The width is stored in width , and the height is stored in height . |
| Event | Encapsulates events. |
| FlowLayout | The flow layout manager. Flow layout positions components left to right, top to bottom. |
| Frame | Creates a standard window that has a title bar, resize corners, and a menu bar. |
| Graphics | Encapsulates the graphics context. |

The AWT supports the following types of controls:

3. Labels
4. Push buttons
5. Check boxes
6. Choice lists
7. Lists
8. Scroll bars
9. Text editing

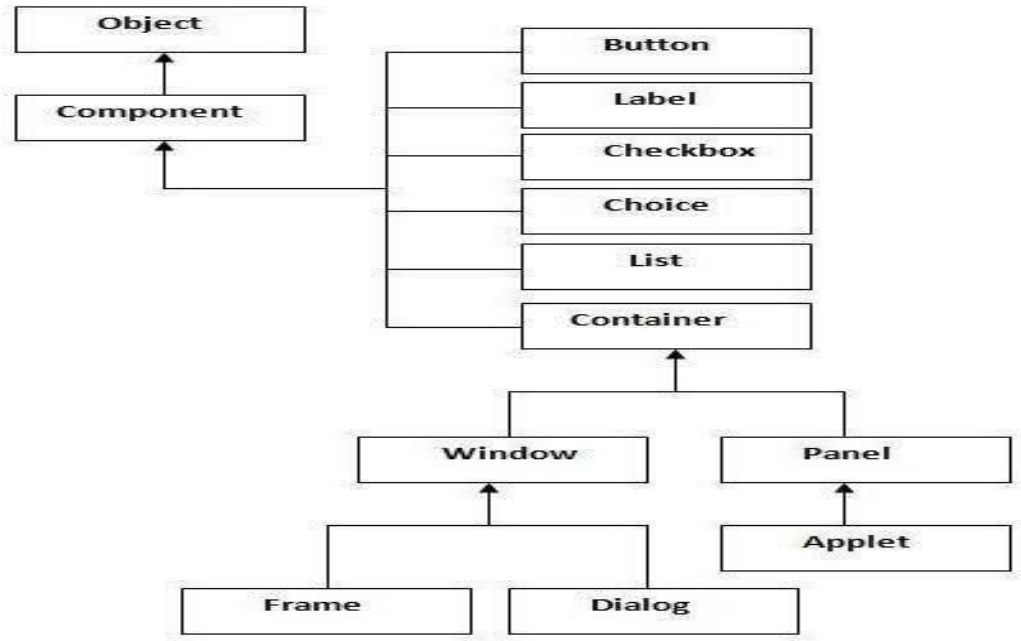
**User interaction with the program is of two types:**

**CUI (Character User Interface):** In CUI user interacts with the application by typing characters or commands. In CUI user should remember the commands. It is not user friendly.

2. **GUI (Graphical User Interface):** In GUI user interacts with the application through graphics. GUI is user friendly. GUI makes application attractive. It is possible to simulate real object in GUI programs. In java to write GUI programs we can use awt (Abstract Window Toolkit) package.

Java AWT Class Hierarchy

The hierarchy of Java AWT classes is given below.




Container

The Container is a component in AWT that can contain other components like buttons, textfields, labels etc. The classes that extend Container class are known as container such as **Frame, Dialog and Panel**.

Window

The window is the container that has no borders and menu bars. You must use frame, dialog or another window for creating a window.

Panel

The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

Frame

The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

**Useful Methods of Component class**

| Method | Description |
|---|---|
| public void add(Component c) | inserts a component on this component. |
| public void setSize(int width,int height) | sets the size(width and height) of the component. |
| public void setLayout(LayoutManager m) | defines the layout manager for the component. |
| public void setVisible(boolean status) | changes the visibility of the component, by default false. |

## Listeners and Listener Methods:

Listeners are available for components. A Listener is an interface that listens to an event from a component. Listeners are available in **java.awt.event package**. The methods in the listener interface are to be implemented, when using that listener.

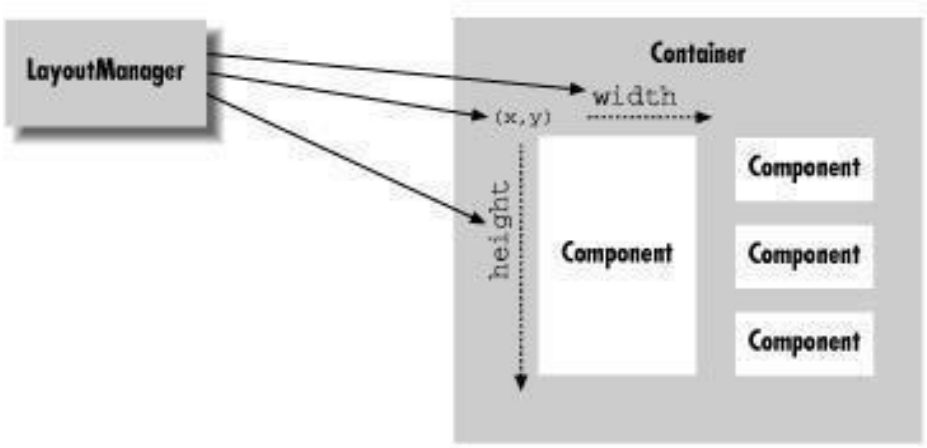
| Component | Listener | Listener methods |
|---|---|---|
| Button | ActionListener | public void actionPerformed (ActionEvent e) |
| Checkbox | ItemListener | public void itemStateChanged (ItemEvent e) |
| CheckboxGroup | ItemListener | public void itemStateChanged (ItemEvent e) |
| TextField | ActionListener<br>FocusListener | public void actionPerformed (ActionEvent e)<br>public void focusGained (FocusEvent e)<br>public void focusLost (FocusEvent e) |
| TextArea | ActionListener<br>FocusListener | public void actionPerformed (ActionEvent e)<br>public void focusGained (FocusEvent e)<br>public void focusLost (FocusEvent e) |
| Choice | ActionListener<br>ItemListener | public void actionPerformed (ActionEvent e)<br>public void itemStateChanged (ItemEvent e) |
| List | ActionListener<br>ItemListener | public void actionPerformed (ActionEvent e)<br>public void itemStateChanged (ItemEvent e) |
| Scrollbar | AdjustmentListener<br>MouseMotionListener | public void<br>adjustmentValueChanged (AdjustmentEvent e)<br>public void mouseDragged (MouseEvent e)<br>public void mouseMoved (MouseEvent e) |
| Label | No listener is needed | |

Layout Managers

A layout manager arranges the child components of a container. It positions and sets the size of components within the container's display area according to a particular layout scheme.

The layout manager's job is to fit the components into the available area, while maintaining the proper spatial relationships between the components. AWT comes with a few standard layout managers that will collectively handle most situations; you can make your own layout managers if you have special requirements.

**LayoutManager at work**




Every container has a **default layout manager**; therefore, when you make a new container, it comes with a LayoutManager object of the appropriate type. You can install a new layout manager at any time with the setLayout() method. Below, we set the layout manager of a container to a BorderLayout:

**setLayout ( new BorderLayout( ) );**

Every component determines three important pieces of information used by the layout manager in placing and sizing it: a minimum size, a maximum size, and a preferred size.

These are reported by the getMinimumSize(), getMaximumSize(), and getPreferredSize(), methods of Component, respectively.

When a layout manager is called to arrange its components, it is working within a fixed area. It usually begins by looking at its container's dimensions, and the preferred or minimum sizes of the child components.

## Layout manager types

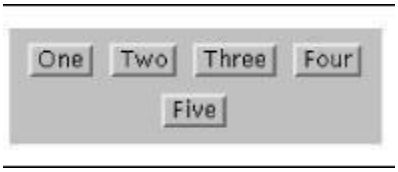
Flow Layout

`FlowLayout` is a simple layout manager that tries to arrange components with their preferred sizes, from left to right and top to bottom in the display. A `FlowLayout` can have a specified justification of `LEFT`, `CENTER`, or `RIGHT`, and a fixed horizontal and vertical padding.

By default, a flow layout uses **`CENTER`** justification, meaning that all components are centered within the area allotted to them. `FlowLayout` is the default for `Panel` components like `Applet`.

The following applet adds five buttons to the default `FlowLayout`.

```
import java.awt.*;
/*
<applet code="Flow" width="500" height="500">
</applet>
*/
public class Flow extends java.applet.Applet
{
   public void init()
  {
        //Default for Applet is FlowLayout
             add( new Button("One") );
       add( new Button("Two") );
       add( new Button("Three") );
       add( new Button("Four") );
       add( new Button("Five") );
   }
}
```




If the applet is small enough, some of the buttons spill over to a second or third row.

### Grid Layout

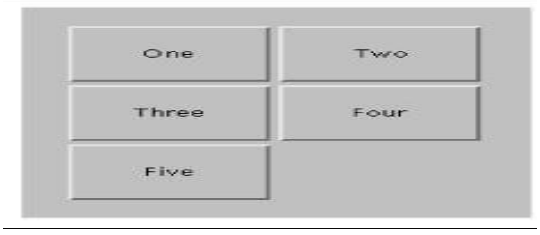
`GridLayout` arranges components into regularly spaced rows and columns. The components are arbitrarily resized to fit in the resulting areas; their minimum and preferred sizes are consequently ignored.

`GridLayout` is most useful for arranging very regular, identically sized objects and for allocating space for Panels to hold other layouts in each region of the container.

`GridLayout` takes the number of rows and columns in its constructor. If you subsequently give it too many objects to manage, it adds extra columns to make the objects fit. You can also set the number of rows or columns to zero, which means that you don't care how many elements the layout manager packs in that dimension.

For example, `GridLayout(2,0)` requests a layout with two rows and an unlimited number of columns; if you put ten components into this layout, you'll get two rows of five columns each. The following applet sets a `GridLayout` with three rows and two columns as its layout manager;

```
import java.awt.*;
/*
   <applet code="Grid" width="500"
    height="500"> </applet>
*/
public class Grid extends java.applet.Applet
  {
   public void init()
    {
      setLayout( new GridLayout( 3, 2 ));
```



```
        add( new Button("One") );
        add( new Button("Two") );
        add( new Button("Three") );
        add( new Button("Four") );
        add( new Button("Five") );
    }
}
```

The five buttons are laid out, in order, from left to right, top to bottom, with one empty spot.

### Border Layout

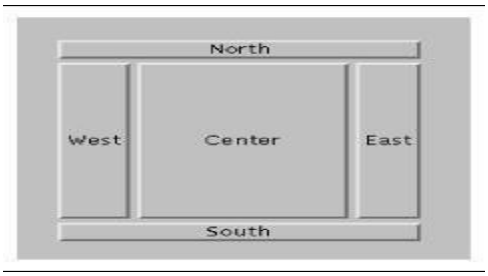
`BorderLayout` is a little more interesting. It tries to arrange objects in one of five geographical locations: "North," "South," "East," "West," and "Center," possibly with some padding between.

**`BorderLayout` is the default layout for `Window` and `Frame` objects**. Because each component is associated with a direction, `BorderLayout` can manage at most five components; it squashes or stretches those components to fit its constraints.

When we add a component to a border layout, we need to specify both the component and the position at which to add it. To do so, we use an overloaded version of the **`add()`** method that takes an additional argument as a constraint.

The following applet sets a `BorderLayout` layout and adds our five buttons again, named for their locations;

```
import java.awt.*;
/*
<applet code="Border" width="500" height="500">
</applet>
*/
public class Border extends java.applet.Applet
  {
    public void init()
   {
        setLayout( new java.awt.BorderLayout() );
        add( new Button("North"), "North" ); add(
        new Button("East"), "East" );
        add( new Button("South"), "South" );
        add( new Button("West"), "West" );
        add( new Button("Center"), "Center" );
    }
}
```




**Compile: javac Border.java**
**Run : appletviewer Border.java**

Java AWT Example

To create simple awt example, you need a frame. There are two ways to create a frame in AWT.

// By extending Frame class (inheritance)

2. By creating the object of Frame class (association)

**Simple example of AWT by inheritance**

```
import java.awt.*;
class First extends Frame{
First(){
Button b=new Button("click me");
b.setBounds(30,100,80,30);// setting button position
add(b);//adding button into frame
setSize(300,300);//frame size 300 width and 300
height setLayout(null);//no layout manager
setVisible(true);//now frame will be visible }

public static void main(String args[]){
First f=new First();
}
}
```

**Simple example of AWT by association**

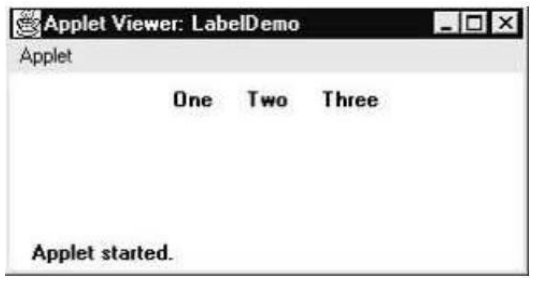
```
import java.awt.*;
class First2{
First2(){
Frame f=new Frame();
 Button b=new Button("click me");
b.setBounds(30,50,80,30);
f.add(b);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String args[]){
First2 f=new First2();
}
}
```

## AWT controls

Labels:

The easiest control to use is a label. A label is an object of type Label, and it contains a string, which it displays. Labels are passive controls that do not support any interaction with the user.

```
 // Demonstrate
    Labels import
    java.awt.*; import
    java.applet.*; /*
<applet code="LabelDemo" width=300 height=200>
</applet> */
public class LabelDemo extends Applet
{
 public void init()
 {
  Label one = new Label("One");
  Label two = new Label("Two");
  Label three = new Label("Three");
       add labels to applet window
  add(one);
  add(two);
  add(three);
}
}
```




Buttons:

The most widely used control is the push button. A push button is a component that contains a label and that generates an event when it is pressed. Push buttons are objects of type Button.

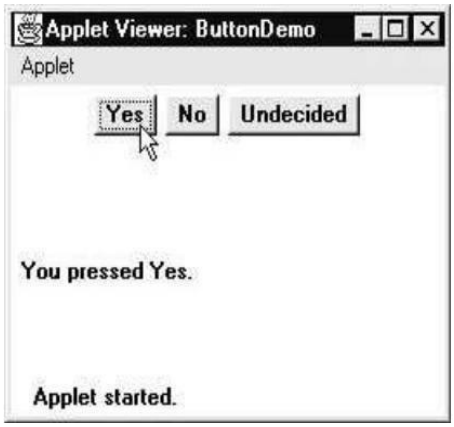
Button class is useful to create push buttons. A push button triggers a series of events.

To create push button: Button b1 =new Button("label");
To get the label of the button: String l = b1.getLabel();
To set the label of the button: b1.setLabel("label");
To get the label of the button clicked: String str = ae.getActionCommand();

- 

```
\{ Demonstrate Buttons
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/* <applet code="ButtonDemo" width=250 height=150>
</applet> */

public class ButtonDemo extends Applet implements ActionListener
  {
String msg = "";
Button yes, no, maybe;
```

```
public void init()
{
yes = new Button("Yes");
no = new Button("No");
maybe = new Button("Undecided");
add(yes);
add(no);
add(maybe);
yes.addActionListener(this);
no.addActionListener(this);
maybe.addActionListener(this);
}

public void actionPerformed(ActionEvent ae)
{
String str = ae.getActionCommand();
  if(str.equals("Yes"))
   {
     msg = "You pressed Yes.";
    }
else if(str.equals("No"))
  {
    msg = "You pressed No.";
   }
else
 {
    msg = "You pressed Undecided.";
  }
 repaint();
}
public void paint(Graphics g)
{
  g.drawString(msg, 6, 100);
}
  }
```




Check Boxes:

A check box is a control that is used to turn an option on or off. It consists of a small box that can either contain a check mark or not. There is a label associated with each check box that describes what option the box represents. You change the state of a check box by clicking on it. Check boxes can be used individually or as part of a group.

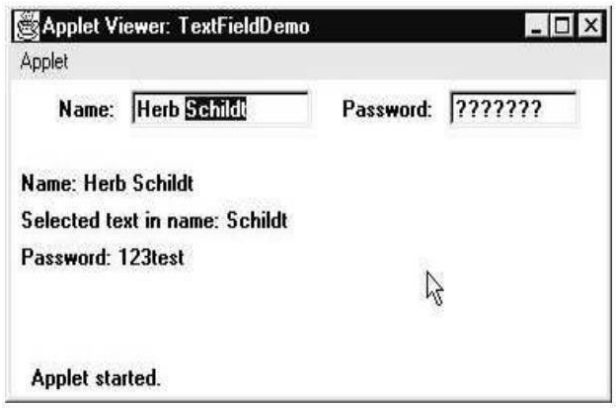
```
\{    Demonstrate check boxes.
  import java.awt.*;
   import java.awt.event.*;
   import java.applet.*;
```

```
/*
<applet code="CheckboxDemo" width=250 height=200>
</applet>
*/
 public class CheckboxDemo extends Applet implements ItemListener
{
String msg = "";
checkbox Win98, winNT, solaris, mac;

public void init()
 {
  win98 = new Checkbox("Windows 98/XP", null, true);
  winNT = new Checkbox("Windows NT/2000");
  solaris = new Checkbox("Solaris"); mac = new
  Checkbox("MacOS");
add(Win98);
add(winNT);
add(solaris);
add(mac);
Win98.addItemListener(this);
winNT.addItemListener(this);
solaris.addItemListener(this);
mac.addItemListener(this);
}
public void itemStateChanged(ItemEvent ie)
{
 repaint();
}
// Display current state of the check boxes.

public void paint(Graphics g)
{
  msg = "Current state: ";
   g.drawString(msg, 6, 80);
  msg = " Windows 98/XP: " + Win98.getState();
   g.drawString(msg, 6, 100);
  msg = " Windows NT/2000: " + winNT.getState();
   g.drawString(msg, 6, 120);
  msg = " Solaris: " + solaris.getState();
   g.drawString(msg, 6, 140);
  msg = " MacOS: " + mac.getState();
   g.drawString(msg, 6, 160);

}
}
```

TextField:

The TextField class implements a single-line text-entry area, usually called an edit control. Text fields allow the user to enter strings and to edit the text using the arrow keys, cut and paste keys, and mouse selections.

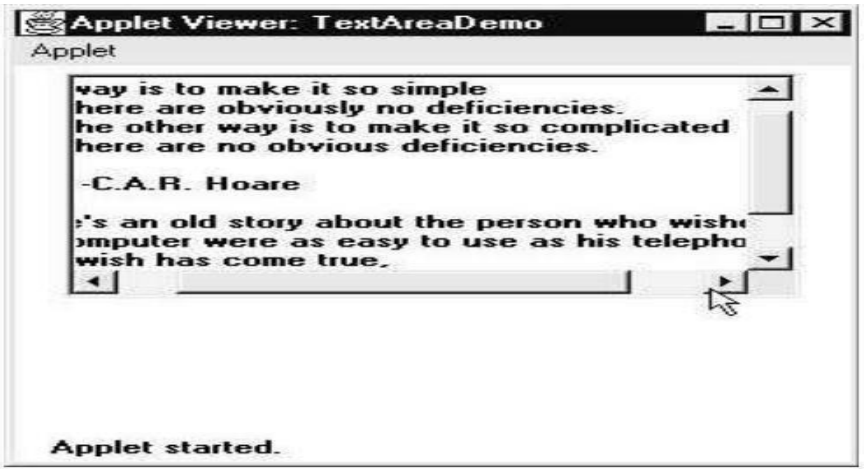
```
// Demonstrate text field.
    import java.awt.*;
    import
    java.awt.event.*;
    import java.applet.*;
/*
<applet code="TextFieldDemo" width=380 height=150>
</applet>
*/
 public class TextFieldDemo extends Applet implements ActionListener
 {
   TextField name, pass;

 public void init()
 {
 Label namep = new Label("Name: ", Label.RIGHT);
 Label passp = new Label("Password: ", Label.RIGHT);
 name = new TextField(12);
 pass = new TextField(8);
 pass.setEchoChar('?');
 add(namep);
 add(name);
 add(passp);
 add(pass);
// register to receive action events
     name.addActionListener(this);
     pass.addActionListener(this);
 }
// User pressed Enter.
 public void actionPerformed(ActionEvent ae)
 {
 repaint();
 }
 public void paint(Graphics g)
 {
  g.drawString("Name: " + name.getText(), 6, 60);
  g.drawString("Selected text in name: " + name.getSelectedText(), 6, 80);
  g.drawString("Password: " + pass.getText(), 6, 100);
 }
 }
```



TextArea:

Sometimes a single line of text input is not enough for a given task. To handle these situations, the AWT includes a simple multiline editor called TextArea .

```
\{ Demonstrate
    TextArea. import
    java.awt.*; import
    java.applet.*;
/*
<applet code="TextAreaDemo" width=300
height=250> </applet>
*/
public class TextAreaDemo extends Applet
{
  public void init()
  {
String val = "There are two ways of constructing " + "a software design.\n" + "One way is to make it so simple\n" + "that there are obviously no deficiencies.\n" + "And the other way is to make it so complicated\n" + "that there are no obvious deficiencies.\n\n" + " -C.A.R. Hoare\n\n"
+ "There's an old story about the person who wished\n" + "his computer were as easy to use as his telephone.\n" + "That wish has come true,\n" + "since I no longer know how to use my telephone.\n\n" + " -Bjarne Stroustrup, AT&T, (inventor of C++)";

TextArea text = new TextArea(val, 10, 30);
add(text);
}
}
```




CheckboxGroup

It is possible to create a set of mutually exclusive check boxes in which one and only one check box in the group can be checked at any one time. These check boxes are often called radio buttons. A Radio button represents a round shaped button such that only one can be selected from a panel. Radio button can be created using CheckboxGroup class and Checkbox classes.

· To create a radio button: CheckboxGroup cbg = new CheckboxGroup ();
Checkbox cb = new Checkbox ("label", cbg, true);
· To know the selected checkbox: Checkbox cb = cbg.getSelectedCheckbox ();
·To know the selected checkbox label: String label = cbg.getSelectedCheckbox().getLabel ();

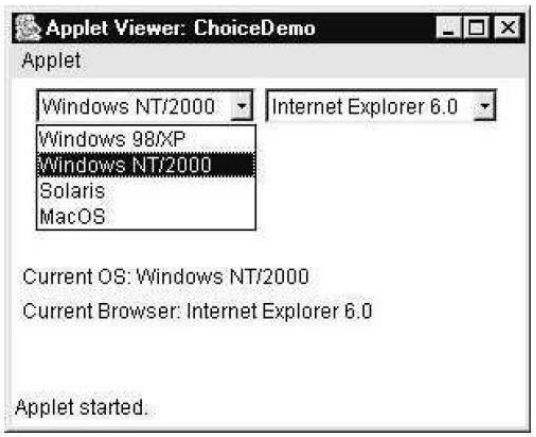
```
\{ Demonstrate check box group.
 import java.awt.*;
import java.awt.event.*;
import java.applet.*;

/*
<applet code="CBGroup" width=250 height=200>
</applet>
*/

public class CBGroup extends Applet implements ItemListener
{
String msg = "";
Checkbox Win98, winNT, solaris, mac;
CheckboxGroup cbg;
public void init() {
cbg = new CheckboxGroup();
Win98 = new Checkbox("Windows 98/XP", cbg, true);
winNT = new Checkbox("Windows NT/2000", cbg, false);
solaris = new Checkbox("Solaris", cbg, false); mac = new
Checkbox("MacOS", cbg, false);
add(Win98);
add(winNT);
add(solaris);
add(mac);
Win98.addItemListener(this);
winNT.addItemListener(this);
solaris.addItemListener(this);
mac.addItemListener(this);
}
public void itemStateChanged(ItemEvent ie) {
repaint();
}
 \{ Display current state of the check
      boxes. public void paint(Graphics g) {
msg = "Current selection: ";
msg += cbg.getSelectedCheckbox().getLabel();
g.drawString(msg, 6, 100);
}
}
```

Choice Controls

The Choice class is used to create a pop-up list of items from which the user may choose. Thus, a Choice control is a form of menu. Choice menu is a popdown list of items. Only one item can be selected.

· To create a choice menu: Choice ch = new Choice();
· To add items to the choice menu: ch.add ("text");
· To know the name of the item selected from the choice menu:
String s = ch.getSelectedItem ();
· To know the index of the currently selected item: int i = ch.getSelectedIndex();
This method returns -1, if nothing is selected.

```
//Demonstrate Choice
    lists. import
    java.awt.*; import
    java.awt.event.*;
    import java.applet.*;
/*
<applet code="ChoiceDemo" width=300
height=180> </applet>
*/
public class ChoiceDemo extends Applet implements ItemListener
{ Choice os, browser;
String msg = ""; public
void init() { os = new
Choice(); browser = new
Choice();
  //add items to os list
os.add("Windows 98/XP");
os.add("Windows NT/2000");
os.add("Solaris");
os.add("MacOS");
  \{ add items to browser list
    browser.add("Netscape 3.x");
    browser.add("Netscape 4.x");
    browser.add("Netscape 5.x");
    browser.add("Netscape 6.x");
    browser.add("Internet Explorer
    4.0"); browser.add("Internet
    Explorer 5.0");
    browser.add("Internet Explorer
    6.0"); browser.add("Lynx 2.4");
    browser.select("Netscape 4.x");
  \{ add choice lists to
    window add(os);
add(browser);
  \{ register to receive item
    events
    os.addItemListener(this);
    browser.addItemListener(
    this);
```



```
}
```

```
public void itemStateChanged(ItemEvent ie) {
repaint();
}
// Display current selections.
 public void paint(Graphics g)
 { msg = "Current OS: ";
msg += os.getSelectedItem();
g.drawString(msg, 6, 120);
msg = "Current Browser: ";
msg += browser.getSelectedItem();
g.drawString(msg, 6, 140);
}
}
```

Lists

The List class provides a compact, multiple-choice, scrolling selection list. Unlike the Choice object, which shows only the single selected item in the menu, a List object can be constructed to show any number of choices in the visible window. It can also be created to allow multiple selections. List provides these constructors: List( )

List(int numRows )

List(int numRows , boolean multipleSelect )

A List box is similar to a choice box, it allows the user to select multiple items.

· To create a list box: List lst = new List();

(or)

List lst = new List (3, true);

This list box initially displays 3 items. The next parameter true represents that the user can select more than one item from the available items. If it is false, then the user can select only one item.

= To add items to the list box: lst.add("text");

= To get the selected items: String x[] = lst.getSelectedItems();

= To get the selected indexes: int x[] = lst.getSelectedIndexes ();

```
//      Demonstrate Lists.
import java.awt.*; import
java.awt.event.*; import
java.applet.*; /*
<applet code="ListDemo" width=300 height=180>
</applet>
*/
public class ListDemo extends Applet implements ActionListener
{ List os, browser;
String msg = "";
public void init() { os
= new List(4, true);
```

```
  browser = new List(4, false);
\} add items to os list
    os.add("Windows 98/XP");
    os.add("Windows
    NT/2000");
    os.add("Solaris");
    os.add("MacOS");
\} add items to browser list
    browser.add("Netscape 3.x");
    browser.add("Netscape 4.x");
    browser.add("Netscape 5.x");
    browser.add("Netscape 6.x");
    browser.add("Internet Explorer
    4.0"); browser.add("Internet
    Explorer 5.0");
    browser.add("Internet Explorer
    6.0"); browser.add("Lynx 2.4");
    browser.select(1);
\} add lists to window
  add(os);
  add(browser);
//   register to receive action events
    os.addActionListener(this);
    browser.addActionListener(this);
  }
  public void actionPerformed(ActionEvent ae)
  { repaint();
  }
//   Display current selections.
  public void paint(Graphics g)
  {
  int idx[];
  msg = "Current OS: ";
  idx = os.getSelectedIndexes();
  for(int i=0; i<idx.length; i++)
  msg += os.getItem(idx[i]) + " ";
  g.drawString(msg, 6, 120);
  msg = "Current Browser: ";
  msg += browser.getSelectedItem();
  g.drawString(msg, 6, 140);
  }
  }
```


Scroll Bars

Scroll bars are used to select continuous values between a specified minimum and maximum. Scroll bars may be oriented horizontally or vertically. Scrollbar class is useful to create scrollbars that can be attached to a frame or text area. Scrollbars can be arranged vertically or horizontally.

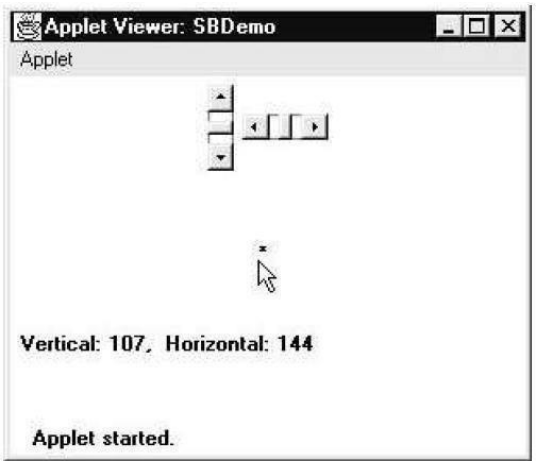
\{ To create a scrollbar : Scrollbar sb = new Scrollbar (alignment, start, step, min, max);
alignment: Scrollbar.VERTICAL, Scrollbar.HORIZONTAL
start: starting value (e.g. 0)
step: step value (e.g. 30) // represents scrollbar length
min: minimum value (e.g. 0)
max: maximum value (e.g. 300)

· To know the location of a scrollbar: int n = sb.getValue ();
· To update scrollbar position to a new position: sb.setValue (int position);
· To get the maximum value of the scrollbar: int x = sb.getMaximum ();
· To get the minimum value of the scrollbar: int x = sb.getMinimum ();
· To get the alignment of the scrollbar: int x = getOrientation ();

This method return 0 if the scrollbar is aligned HORIZONTAL, 1 if aligned VERTICAL.

```
// Demonstrate scroll bars.
  import java.awt.*;
  import java.awt.event.*;
  import java.applet.*;
 /*
 <applet code="SBDemo" width=300 height=200>
 </applet>
 */
 public class SBDemo extends Applet
 implements AdjustmentListener, MouseMotionListener
 { String msg = "";
 Scrollbar vertSB, horzSB;
 public void init() {
 int width = Integer.parseInt(getParameter("width")); int
 height = Integer.parseInt(getParameter("height"));
 vertSB = new Scrollbar(Scrollbar.VERTICAL, 0, 1, 0, height);
 horzSB = new Scrollbar(Scrollbar.HORIZONTAL, 0, 1, 0, width);
 add(vertSB);
 add(horzSB);
// register to receive adjustment events
  vertSB.addAdjustmentListener(this);
  horzSB.addAdjustmentListener(this);
  addMouseMotionListener(this);
 }
 public void adjustmentValueChanged(AdjustmentEvent ae) {
 repaint();
 }
// Update scroll bars to reflect mouse dragging.
 public void mouseDragged(MouseEvent me) {
 int x = me.getX();
 int y = me.getY();
 vertSB.setValue(y);
 horzSB.setValue(x);
```

```
    repaint();
    }
//    Necessary for MouseMotionListener
  public void mouseMoved(MouseEvent me) {
    }
//    Display current value of scroll bars.
    public void paint(Graphics g) {
    msg = "Vertical: " + vertSB.getValue();
    msg += ", Horizontal: " + horzSB.getValue();
    g.drawString(msg, 6, 160);
  \} show current mouse drag position
    g.drawString("*", horzSB.getValue(),
    vertSB.getValue());
    }
    }
```




Graphics

The AWT supports a rich assortment of graphics methods. All graphics are drawn relative to a window.

Graphics class and is obtained in two ways:

\} It is passed to an applet when one of its various methods, such as paint( ) or update( ), is called.

\} It is returned by the getGraphics( ) method of Component.

Drawing Lines

Lines are drawn by means of the drawLine( ) method, shown here:

void drawLine(int startX, int startY, int endX, int endY)

drawLine( ) displays a line in the current drawing color that begins at startX,startY and ends at endX,endY.

The following applet draws several lines:

```
//    Draw lines import
  java.awt.*; import
  java.applet.*; /*

   <applet code="Lines" width=300
   height=200> </applet>
   */
   public class Lines extends Applet
   { public void paint(Graphics g) {
   g.drawLine(0, 0, 100, 100);
   g.drawLine(0, 100, 100, 0);
   g.drawLine(40, 25, 250, 180);
   g.drawLine(75, 90, 400, 400);
   g.drawLine(20, 150, 400, 40);
   g.drawLine(5, 290, 80, 19);
   } }
```

Drawing Rectangles

The drawRect( ) and fillRect( ) methods display an outlined and filled rectangle, respectively. They are shown here:

void drawRect(int top, int left, int width, int height)
void fillRect(int top, int left, int width, int height)

The upper-left corner of the rectangle is at top,left. The dimensions of the rectangle are specified by width and height.

To draw a rounded rectangle, use drawRoundRect( ) or fillRoundRect( ), both shown here: void drawRoundRect(int top, int left, int width, int height,int xDiam, int yDiam)
void fillRoundRect(int top, int left, int width, int height, int xDiam, int yDiam)

```
//  Draw  rectangles
import  java.awt.*;
import java.applet.*;
/*
<applet code="Rectangles" width=300
height=200> </applet>
*/
public class Rectangles extends Applet {
public void paint(Graphics g) {
g.drawRect(10, 10, 60, 50); g.fillRect(100,
10, 60, 50); g.drawRoundRect(190, 10,
60, 50, 15, 15); g.fillRoundRect(70, 90,
140, 100, 30, 40);
}
}
```

Drawing Ellipses and Circles

To draw an ellipse, use drawOval( ). To fill an ellipse, use fillOval( ). These methods are shown here:

void drawOval(int top, int left, int width, int height)
void fillOval(int top, int left, int width, int height)

```
// Draw
Ellipses
import
java.awt.*
; import
java.apple
t.*; /*
<applet code="Ellipses" width=300
height=200> </applet>
*/
public class Ellipses extends Applet
{ public void paint(Graphics g) {
g.drawOval(10, 10, 50, 50);
g.fillOval(100, 10, 75, 50);
g.drawOval(190, 10, 90, 30);
g.fillOval(70, 90, 140, 100);
} }
```

## Drawing Arcs

Arcs can be drawn with drawArc( ) and fillArc( ), shown here:

```
void drawArc(int top, int left, int width, int height, int startAngle,int sweepAngle)
void fillArc(int top, int left, int width, int height, int startAngle,int sweepAngle)
```

The arc is bounded by the rectangle whose upper-left corner is specified by top,left and whose width and height are specified by width and height. The arc is drawn from startAngle through the angular distance specified by sweepAngle. Angles are specified in degrees.

Zero degrees is on the horizontal, at the three o' clock position. The arc is drawn counterclockwise if sweepAngle is positive, and clockwise if sweepAngle is negative. Therefore, to draw an arc from twelve o' clock to six o' clock, the start angle would be 90 and the sweep angle 180.

## Drawing Polygons

It is possible to draw arbitrarily shaped figures using drawPolygon( ) and fillPolygon( ), shown here:

```
void drawPolygon(int x[ ], int y[ ], int numPoints)
void fillPolygon(int x[ ], int y[ ], int numPoints)
```

The polygon' s endpoints are specified by the coordinate pairs contained within the x and y arrays. The number of points defined by x and y is specified by numPoints. There are alternative forms of these methods in which the polygon is specified by a Polygon object.

The following applet draws several arcs:

```
//Draw Arcs
import
java.awt.*;
import
java.applet.*;
/*

<applet code="Arcs"
width=300 height=200>
</applet>
*/
public class Arcs extends Applet {
public void paint(Graphics g) {
g.drawArc(10, 40, 70, 70, 0, 75);
g.fillArc(100, 40, 70, 70, 0, 75);
g.drawArc(10, 100, 70, 80, 0, 175);
g.fillArc(100, 100, 70, 90, 0, 270);
g.drawArc(200, 80, 80, 80, 0, 180);
}
}
```

The following applet draws an hourglass shape:

```
// Draw Polygon
import
java.awt.*;
import
java.applet.*;
/*
<applet code="HourGlass"
width=230 height=210>
</applet>
*/
public class HourGlass extends Applet {
public void paint(Graphics g) {
int xpoints[] = {30, 200, 30, 200, 30};
int ypoints[] = {30, 30, 200, 200, 30};
int num = 5; g.drawPolygon(xpoints,
ypoints, num);
}
```

## SWINGS

- Swing is a set of classes that provides more powerful and flexible components than are possible with the AWT. **Swing** is a GUI widget toolkit for **Java**. It is part of Oracle's **Java** Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

- In addition to the familiar components, such as buttons, check boxes, and labels, Swing supplies several exciting additions, including tabbed panes, scroll panes, trees, and tables. Even familiar components such as buttons have more capabilities in Swing. For example, a button may have both an image and a text string associated with it. Also, the image can be changed as the state of the button changes.

- Unlike AWT components, Swing components are not implemented by platform specific code. Instead, they are written entirely in Java and, therefore, are platform-independent. The term *lightweight* is used to describe such elements.

✓ The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

## Differences between AWT and Swing

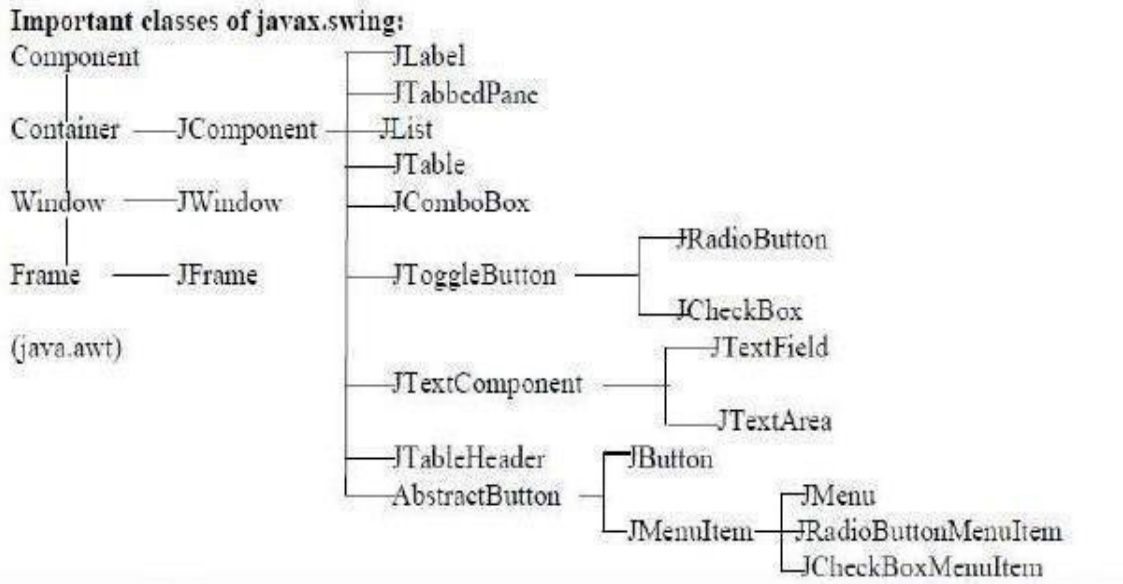
| AWT | Swing |
|---|---|
| AWT components are called Heavyweight component. | Swings are called light weight component because swing components sits on the top of AWT components and do the work. |
| AWT components are platform dependent. | Swing components are made in purely java and they are platform independent. |
| AWT components require java.awt package. | Swing components require javax.swing package. |
| AWT is a thin layer of code on top of the OS. | Swing is much larger. Swing also has very much richer functionality. |
| AWT stands for Abstract windows toolkit. | Swing is also called as JFC's (Java Foundation classes). |
| This feature is not supported in AWT. | We can have different look and feel in Swing. |
| Using AWT, you have to implement a lot of things yourself. | Swing has them built in. |

| | |
|---|---|
| This feature is not available in AWT. | Swing has many advanced features like JTabel, Jtabbed pane which is not available in AWT. Also.Swing components are called "lightweight" because they do not require a native OS object to implement their functionality. JDialog and JFrame are heavyweight, because they do have a peer. So components like JButton, JTextArea, etc., are lightweight because they do not have an OS peer. |

**The Swing component classes are:**

| Class | Description |
|---|---|
| AbstractButton | Abstract superclass for Swing buttons. |
| ButtonGroup | Encapsulates a mutually exclusive set ofbuttons. |
| ImageIcon | Encapsulates an icon. |
| JApplet | The Swing version of Applet. |
| JButton | The Swing push button class. |
| JCheckBox | The Swing check box class. |
| JComboBox | Encapsulates a combo box (combination of a drop-down list & text field). |
| JLabel | The Swing version of a label. |
| JRadioButton | The Swing version of a radio button. |
| JScrollPane | Encapsulates a scrollable window. |
| JTabbedPane | Encapsulates a tabbed window. |
| JTable | Encapsulates a table-based control. |
| JTextField | The Swing version of a text field. |
| JTree | Encapsulates a tree-based control. |

**Hierarchy for Swing components:**



**JApplet**

Fundamental to Swing is the **JApplet** class, which extends **Applet**. Applets that use Swing must be subclasses of **JApplet**. **JApplet** is rich with functionality that is not foundin **Applet**.

The content pane can be obtained via the method shown here:
Container getContentPane( )

The **add( )** method of **Container** can be used to add a component to a content pane. Its form is shown here:
void add(*comp*)

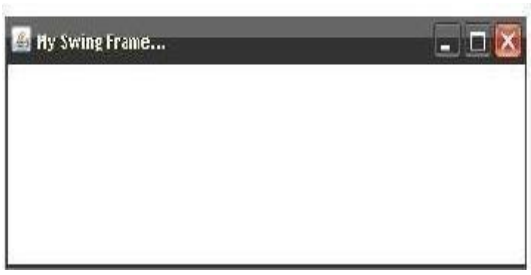
Here, *comp* is the component to be added to the content pane.

**JFrame**

Create an object to JFrame: JFrame ob = new JFrame ("title"); (or)
Create a class as subclass to JFrame class: MyFrame extends JFrame
Create an object to that class : MyFrame ob = new MyFrame ();

**Example: Write a program to create a frame by creating an object to JFrame class.**

```
//A swing Frame
import javax.swing.*;
class MyFrame
{
public static void main (String agrs[])
{ JFrame jf = new JFrame ("My Swing Frame...");
jf.setSize (400,200);
jf.setVisible (true);
jf.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
}
}
```





**Note:** To close the frame, we can take the help of getDefaultCloseOperation () method of JFrame class, as shown here: **getDefaultCloseOperation (constant);**

where the constant can be any one of the following:

- JFrame.EXIT_ON_CLOSE: This closes the application upon clicking on close button.
- JFrame.DISPOSE_ON_CLOSE: This disposes the present frame which is visible on the screen. The JVM may also terminate.
- JFrame.DO_NOTHING_ON_CLOSE: This will not perform any operation upon clicking on close button.
- JFrame.HIDE_ON_CLOSE: This hides the frame upon clicking on close button.

**Window Panes:** In swings the components are attached to the window panes only. A window pane represents a free area of a window where some text or components can be displayed. For example, we can create a frame using JFrame class in javax.swing which contains a free area inside it, this free area is called 'window pane'. Four types of window panes are available in javax.swing package.

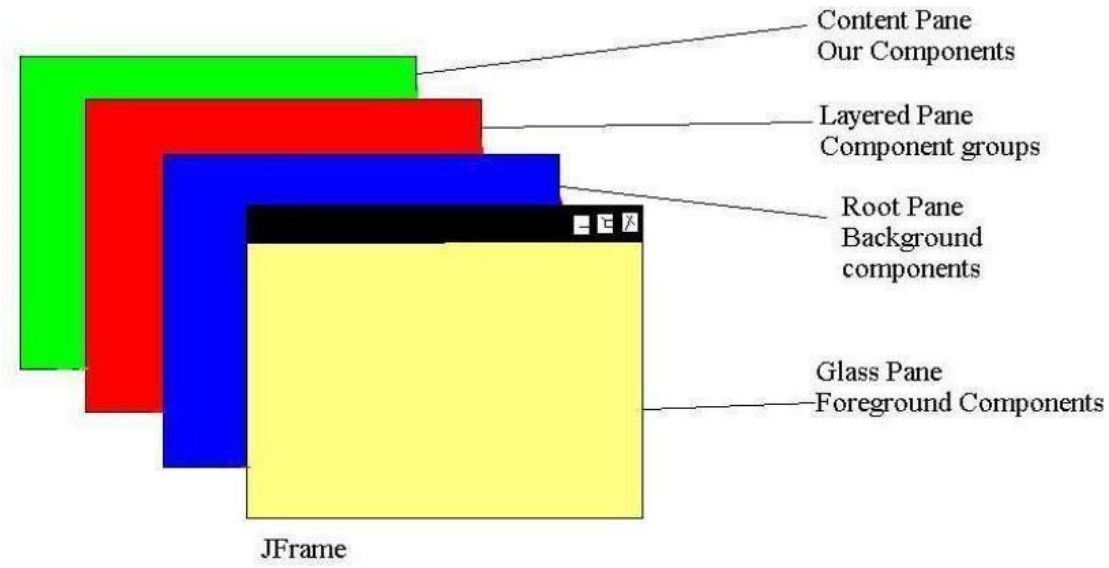
**Glass Pane:** This is the first pane and is very close to the monitors screen. Any components to be displayed in the foreground are attached to this glass pane. To reach this glass pane, we use getGlassPane () method of JFrame class.

**Root Pane:** This pane is below the glass pane. Any components to be displayed in the background are displayed in this pane. Root pane and glass pane are used in animations also.
For example, suppose we want to display a flying aeroplane in the sky. The aeroplane can be displayed as a .gif or .jpg file in the glass pane where as the blue sky can be displayed in the root pane in the background. To reach this root pane, we use getRootPane () method of JFrame class.

**Layered Pane:** This pane lies below the root pane. When we want to take several components as a group, we attach them in the layered pane. We can reach this pane by calling getLayeredPane () method of JFrame class.

**Content Pane:** This is the bottom most pane of all. Individual components are attached to this pane. To reach this pane, we can call getContentPane () method of JFrame class.

Displaying Text in the Frame:
paintComponent (Graphics g) method of JPanel class is used to paint the portion of a component in swing. We should override this method in our class. In the following example, we are writing our class MyPanel as a subclass to JPanel and override the painComponent () method.




## Write a program to display text in the frame

```
import javax.swing.*;
import java.awt.*;
class MyPanel extends JPanel
{ public void paintComponent (Graphics g)
{ super.paintComponent (g); //call JPanel's method
setBackground (Color.red);
g.setColor (Color.white);
        etFont (new Font("Courier New",Font.BOLD,30));
```

```
                rawString ("Hello Readers!", 50, 100);
}
}
class FrameDemo extends JFrame
{ FrameDemo ()
{
Container c = getContentPane ();
MyPanel mp = new MyPanel ();
c.add (mp);
setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
}
public static void main(String args[])
{ FrameDemo ob = new FrameDemo ();
ob.setSize (600, 200);
ob.setVisible (true);
}
}
```

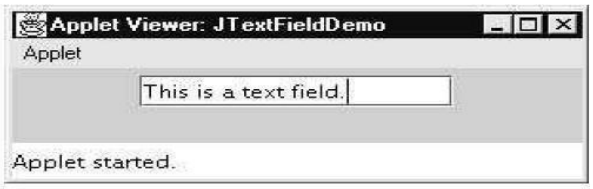

## TEXT FIELDS

The Swing text field is encapsulated by the JTextComponent class, which extends JComponent. It provides functionality that is common to Swing text components. One of its subclasses is JTextField, which allows you to edit one line of text. Some of its constructors are shown here:

```
JTextField( )
JTextField(int cols)
JTextField(String s, int cols)
JTextField(String s)
```

```
import java.awt.*;
import javax.swing.*;
/*
<applet code="JTextFieldDemo" width=300 height=50>
```

```
</applet>
*/
public class JTextFieldDemo extends JApplet {
  JTextField jtf;
public void init()
{
// Get content pane
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());
   // Add text field to content
      pane jtf = new
      JTextField(15);
      contentPane.add(jtf);
}
}
```




## BUTTONS

Swing buttons provide features that are not found in the Button class defined by the AWT. For example, you can associate an icon with a Swing button. Swing buttons are subclasses of the AbstractButton class, which extends JComponent. AbstractButton contains many methods that allow you to control the behavior of buttons, check boxes, and radio buttons.

The JButton Class

The JButton class provides the functionality of a push button. JButton allows an icon, a string, or both to be associated with the push button. Some of its constructors are shown here:

- To create a JButton with text: JButton b = new JButton (“OK”);
- To create a JButton with image: JButton b = new JButton (ImageIcon ii);
- To create a JButton with text & image: JButton b = new JButton (“OK”, ImageIcon ii);

It is possible to create components in swing with images on it. The image is specified by ImageIcon class object.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="JButtonDemo" width=250 height=300>
</applet>
*/
public class JButtonDemo extends JApplet implements ActionListener {
JTextField jtf;
public void init() {
// Get content pane
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());
// Add buttons to content pane
```

```
ImageIcon france = new ImageIcon("france.gif");
JButton jb = new JButton(france);
jb.setActionCommand("France");
jb.addActionListener(this); contentPane.add(jb);

ImageIcon germany = new ImageIcon("germany.gif");
jb = new JButton(germany);
jb.setActionCommand("Germany");
jb.addActionListener(this);
contentPane.add(jb);
ImageIcon italy = new ImageIcon("italy.gif");
jb = new JButton(italy);
jb.setActionCommand("Italy");
jb.addActionListener(this);
contentPane.add(jb);
ImageIcon japan = new ImageIcon("japan.gif");
jb = new JButton(japan);
jb.setActionCommand("Japan");
jb.addActionListener(this);
contentPane.add(jb);
      // Add text field to
           content pane jtf = new
           JTextField(15);
           contentPane.add(jtf);
}
public void actionPerformed(ActionEvent ae)
{ jtf.setText(ae.getActionCommand());
}
}
```

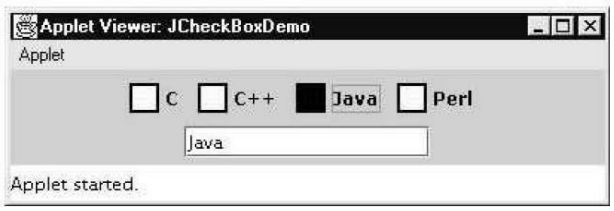
## CHECK BOXES

The JCheckBox class, which provides the functionality of a check box, is a concrete implementation of AbstractButton. Its immediate superclass is JToggleButton, which provides support for two-state buttons. Some of its constructors are shown here:

```
JCheckBox(Icon i)
JCheckBox(Icon i, boolean state)
JCheckBox(String s)
JCheckBox(String s, boolean state)
JCheckBox(String s, Icon i)
JCheckBox(String s, Icon i, boolean state)
```

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="JCheckBoxDemo" width=400 height=50>
</applet>
*/
```

```
public class JCheckBoxDemo extends JApplet implements ItemListener {
JTextField jtf;
public void init()
 {
// Get content pane
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());
// Create icons
ImageIcon normal = new ImageIcon("normal.gif");
ImageIcon rollover = new ImageIcon("rollover.gif");
ImageIcon selected = new ImageIcon("selected.gif");
// Add check boxes to the content pane
JCheckBox cb = new JCheckBox("C", normal);
cb.setRolloverIcon(rollover);
cb.setSelectedIcon(selected);
cb.addItemListener(this); contentPane.add(cb);

cb = new JCheckBox("C++",
normal); cb.setRolloverIcon(rollover);
cb.setSelectedIcon(selected);
cb.addItemListener(this);
contentPane.add(cb);
cb = new JCheckBox("Java", normal);
cb.setRolloverIcon(rollover);
cb.setSelectedIcon(selected);
cb.addItemListener(this);
contentPane.add(cb);
cb = new JCheckBox("Perl", normal);
cb.setRolloverIcon(rollover);
cb.setSelectedIcon(selected);
cb.addItemListener(this);
contentPane.add(cb);
// Add text field to the content pane
jtf = new JTextField(15);
contentPane.add(jtf);
}
public void itemStateChanged(ItemEvent ie) {
JCheckBox cb = (JCheckBox)ie.getItem();
jtf.setText(cb.getText());
}
}
```

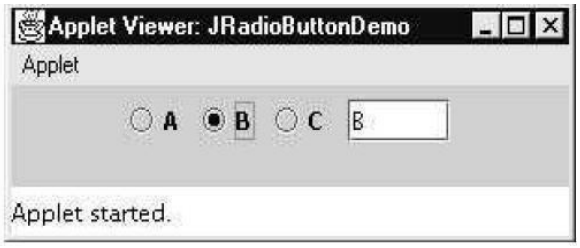



## RADIO BUTTONS

Radio buttons are supported by the JRadioButton class, which is a concrete implementation of AbstractButton. Its immediate superclass is JToggleButton, which provides support for two-state buttons. Some of its constructors are shown here:

JRadioButton(Icon i)
JRadioButton(Icon i, boolean state)
JRadioButton(String s)
JRadioButton(String s, boolean state)
JRadioButton(String s, Icon i)
JRadioButton(String s, Icon i, boolean state)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="JRadioButtonDemo" width=300 height=50>
</applet>
*/
public class JRadioButtonDemo extends JApplet implements ActionListener {
JTextField tf;
public void init() {
// Get content pane
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());
1. Add radio buttons to content pane
JRadioButton b1 = new JRadioButton("A");
b1.addActionListener(this);
contentPane.add(b1);
JRadioButton b2 = new JRadioButton("B");
b2.addActionListener(this);
contentPane.add(b2);
JRadioButton b3 = new JRadioButton("C");
b3.addActionListener(this);
contentPane.add(b3);
2. Define a button group
ButtonGroup bg = new ButtonGroup();
bg.add(b1);
bg.add(b2);
bg.add(b3);
// Create a text field and add it
// to the content pane
tf = new JTextField(5);
contentPane.add(tf);
}
public void actionPerformed(ActionEvent ae) {
tf.setText(ae.getActionCommand());
}
}
```



Limitations of AWT:

The AWT defines a basic set of controls, windows, and dialog boxes that support a usable, but limited graphical interface. One reason for the limited nature of the AWT is that it translates its various visual components into their corresponding, platform-specific equivalents or peers. This means that the look and feel of a component is defined by the platform, not by java. Because the AWT components use native code resources, they are referred to as heavy weight.

The use of native peers led to several problems.

**First**, because of variations between operating systems, a component might look, or even act, differently on different platforms. This variability threatened java's philosophy: write once, run anywhere.

**Second,** the look and feel of each component was fixed and could not be changed. Third, the use of heavyweight components caused some frustrating restrictions. Due to these limitations Swing came and was integrated to java. Swing is built on the AWT. Two key Swing features are: Swing components are light weight, Swing supports a pluggable look and feel.