**Core Java Tutorial**

**Chapter 1: Introduction**

Java is a high-level programming language originally developed by Sun Microsystems and released in 1995. Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX. This tutorial gives a complete understanding of Java. Java is an object oriented multithreaded programming language. It is designed to be small, simple and portable across different platforms as well as OS.

What is Java?

Java is a **programming language** and a **platform**. Java is a high level, robust, secured and object-oriented programming language.

**Platform**: Any hardware or software environment in which a program runs is known as a platform. Since Java has its own runtime environment (JRE) and API, it is called platform.

## Where it is used?

According to Sun, 3 billion devices run java. There are many devices where java is currently used. Some of them are as follows:

1. Desktop Applications such as acrobat reader, media player, antivirus etc.
2. Web Applications such as irctc.co.in, javatpoint.com etc.
3. Enterprise Applications such as banking applications.
4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. Games etc.

## Types of Java Applications:

There are mainly **4** types of applications that can be created using java programming:

1) **Standalone Application**

It is also known as desktop application or window-based application. An application that we need to install on every machine such as media player, antivirus etc. AWT and Swing are used in java for creating standalone applications.

2) **Web Application**

An application that runs on the server side and creates dynamic page, is called web application. Currently, servlet, jsp, struts, jsf etc. technologies are used for creating web applications in java.

3) **Enterprise Application**

An application that is distributed in nature, such as banking applications etc. It has the advantage of high level security, load balancing and clustering. In java, EJB is used for creating enterprise applications.

4) **Mobile Application**

An application that is created for mobile devices. Currently Android and Java ME are used for creating mobile applications.

### History of Java

**Java history** is interesting to know. 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, televisions etc.

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 point 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-top boxes.

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.



#### Why Oak name for java language?

5) **Why Oak?** Oak is a symbol of strength and choosen as a national tree of many countries like U.S.A., France, Germany, Romania etc.

6) In 1995, Oak was renamed as **"Java"** because it was already a trademark by Oak Technologies.

#### Why Java name for java language?

7) **Why they choosed java name for java language?** The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA" etc. They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell and fun to say.

According to James Gosling "Java was one of the top choices along with **Silk**". Since java was so unique, most of the team members preferred java.

8) Java is an island of Indonesia where first coffee was produced (called java coffee).

9) Notice that Java is just a name not an acronym.

10) Originally developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995.

11) In 1995, Time magazine called **Java one of the Ten Best Products of 1995**.

12) JDK 1.0 released in (January 23, 1996).

### 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. Platform independent
4. Secured
5. Robust
6. Architecture neutral
7. Portable
8. Dynamic
9. Interpreted
10. High Performance
11. Multithreaded
12. Distributed
13. **Simple:**

According to Sun, Java language is simple because: syntax is based on C++ (so easier for programmers to learn it after C++). Removed many confusing and/or rarely-used features e.g., explicit pointers, operator overloading etc. No need to remove unreferenced objects because there is Automatic Garbage Collection in java.

1. **Object-Oriented:**

Object-oriented programming (OOP) is a programming language model organized around objects rather than "actions" and data rather than logic. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.

Basic concepts of OOPs are:

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

### Platform Independent:

A platform is the hardware or software environment in which a program runs. There are two types of platforms software-based and hardware-based. Java provides software-based platform. The Java platform differs from most other platforms in the sense that it's a software-based platform that runs on top of other hardware-based platforms .It has two components:

1. Runtime Environment
2. API(Application Programming Interface)

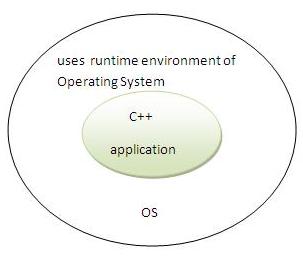
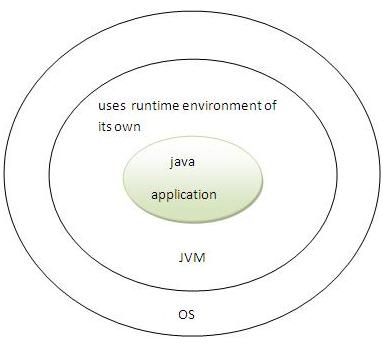
### java is platform independent

### Java code can be run on multiple platforms e.g.Windows,Linux,Sun Solaris,Mac/OS etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere (WORA).

### Secured:

### Java is secured because:

* No explicit pointer
* Programs run inside virtual machine sandbox



* **Classloader:** adds security by separating the package for the classes of the local file system from those that are imported from network sources
* **Bytecode Verifier:** checks the code fragments for illegal code that can violate access right to objects.
* **Security Manager:** determines what resources a class can access such as reading and writing to the local disk

These security are provided by java language. Some security can also be provided by application developer through SSL,JAAS,cryptography etc

### Robust:

### Robust simply means strong. Java uses strong memory management. There are lack of pointers that avoids security problem. There is automatic garbage collection in java. There is exception handling and type checking mechanism in java. All these points makes java robust

### Architecture-neutral: There is no implementation dependent features e.g. size of primitive types is set

### Portable: We may carry the java bytecode to any platform

### High Performance: Java is faster than traditional interpretation since byte code is "close" to native code still somewhat slower than a compiled language (e.g., C++)

### Distributed:

### We can create distributed applications in java. RMI and EJB are used for creating distributed applications. We may access files by calling the methods from any machine on the internet.

Java development environment has 2 Part:

1. Compiler:

When program is compiled, it is translated into machine code i.e. specific to processor but java compiler generates byte code instead of machine code. This byte code id independent on machine.

1. Interpreter:

It executes java0020program. When java program is executed, it also check byte code.

Java is an Object-Oriented Language. As a language that has the Object Oriented feature, Java supports the following fundamental concepts:

* Polymorphism
* Inheritance
* Encapsulation
* Abstraction
* Classes
* Objects
* Instance
* Method

**Features of Java:**

1. Simple and powerful:

Java prorovides a small number of clear ways to achive a given task since it exposes ineer working of machine.The programmer perform his desired action without fear.

1. Secure:

Java developed for web designing.The main problems of internet is threatening of viruses and system hackers.To overcome all these fears java provides security.Javaachive these protection by confirming a java programme to the java exicuationenviorment and by making it inexcessible to other partsofcomputer.Javadosen’t support to pointer.

1. Portable:

Java programmer’s are independent on machine and operating system.

1. Object Oriented:

Java team gave a clean useable and realistic approach to objects.Theobject model in java is simple and easy to extend.It also supports all features of object oriented programming.

1. Multithreading:

The java supports multithreaded programming which allow user to write programmers that perform many function simultenously.

1. Robust:

The most programmer in use today fail due to two reasons.

1)Memeory Management

2)Exceptional condition

The java frees the user from the worry about many of the most common causes of programming errors .Java ckecks the code at compilation time as well as runtime.

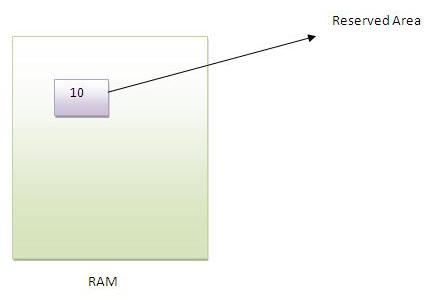
1. Distributed:

Java is designed for the distributed for the distributed enviorment of internrt causes it handles TCP/IP protocols.The original version of java include feature for intra address space messaging.The allowed the object on two different computer to execute procedure called “Remote method Invocation (RMI)”for these.

**Data Types and variable in java:** There are three types of variables: local, instance and static. There are two types of datatypes in java, primitive and non-primitive.When datatypes are required as true objects java provides classes for these primitive datatypes, known as wrapper classes.

Variable

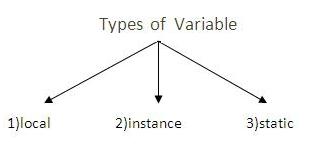
Variable is name of reserved area allocated in memory.



1. **int** data=50;//Here data is variable

Types of Variable

|  |
| --- |
| There are three types of variables in java   * local variable * instance variable * static variable |



**Local Variable**

|  |
| --- |
| A variable that is declared inside the method is called local variable. |

**Instance Variable**

|  |
| --- |
| A variable that is declared inside the class but outside the method is called instance variable . It is not declared as static. |

**Static variable**

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

Example to understand the types of variables

**class** A{

**int** data=50;//instance variable

**static** **int** m=100;//static variable

**void** method(){

**int** n=90;//local variable

}

}//end of class

**DATA TYPES IN JAVA:**

In java, there are two types of data types

* primitive data types
* Non-primitive data types



|  |  |  |
| --- | --- | --- |
| **Data Type** | **Default Value** | **Default size** |
| boolean | false | 1 bit |
| char | '\u0000' | 2 byte |
| byte | 0 | 1 byte |
| short | 0 | 2 byte |
| int | 0 | 4 byte |
| long | 0L | 8 byte |
| float | 0.0f | 4 byte |
| double | 0.0d | 8 byte |

Following are the primitive data types:

1. Byte:

It stores whole no require 1 byte memory.

Range: -128 to +127

1. Short:

It also stores whole no require 2 byte memory.

Range: -32768 to +32767

1. Int:

It stores whole no require 4 byte memory.

Range: -2147483648 to +2147483647

1. Long:

It stores whole no require 8 byte memory.

Range: -9223372036854775808 to +9223372036854775807

1. Float:

It stores decimal value having single precision. Required memory 4 byte.

Range: 3.4E^-32 to 3.4E^38

1. Double:

It stores decimal values having double precision. It requires 8 bytes memory.

Range: -1.7E^-308 to 1.7E^308

1. Char:

It stores single character. It requires 2 bytes memory.

1. Boolean:

It stores true or false values. It requires 1 byte memory.

**Operators:**

**Operator** in java is a symbol that is used to perform operations. There are many types of operators in java such as unary operator, arithmetic operator, relational operator, shift operator, bitwise operator, ternary operator and assignment operator.

|  |  |
| --- | --- |
| **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 | = += -= \*= /= %= &= ^= |= <<= >>= >>>= |

1. **Arithmetic Operators:**

It is used while performing arithmetic operation. Operators are: +,-,\*,/,%

1. **Relational Operators:**

It compares 2 values and gives Boolean result. Operators are: <,>,<=,>=,!=

1. **Logical Operators:**

It combine the result of 2 or more conditions. Operators are:

1. && (Logical AND):

When all conditions are true, result is true. If one of the condition is false, result is false.

1. || (Logical OR):

When one of the condition is true, result is true. When all conditions are false result is false.

1. ! (Logical NOT):

It reverse condition. If condition is true result is false and viceversa.

1. **Assignment Operators:**

It assign value of expression on right hand side to variable on left hand side.

=,+=,-=,\*=,/=,%=.

1. **Increment/Decrement Operators:**

1) Increment: ++

It increases value of variable by 1.

2) Decrement: --

It decreases value of variable by 1.

1. **Bitwise Operators:**

&,|,^,>>,<<,~

**Java If else Statement:**

The Java *if statement* is used to test the condition. It returns *true* or *false*. There are various types of if statement in java.

* if statement
* if-else statement
* nested if statement
* if-else-if ladder

1. **If…Else loop:**

Syntax:

If (condition 1)

Statement 1;

Else

Statement 2;

**public** **class** IfExample {

**public** **static** **void** main(String[] args) {

**int** age=20;

**if**(age>18){

System.out.print("Age is greater than 18");

}

}

}

**If else example:**

**public** **class** IfElseExample {

**public** **static** **void** main(String[] args) {

**int** number=13;

**if**(number%2==0){

         System.out.println("even number");

     }**else**{

         System.out.println("odd number");

     }

}

}

1. **Nested If…Else loop:**

Syntax:

If (condition 1)

Statement 1;

Else if(condition 2)

Statement 2;

Else if(condition 3)

Statement 3;

.

.

.

Else

Statement;

**public** **class** IfElseIfExample {

**public** **static** **void** main(String[] args) {

**int** marks=65;

**if**(marks<50){

         System.out.println("fail");

     }

**else** **if**(marks>=50 && marks<60){

         System.out.println("D grade");

     }

**else** **if**(marks>=60 && marks<70){

         System.out.println("C grade");

     }

**else** **if**(marks>=70 && marks<80){

         System.out.println("B grade");

     }

**else** **if**(marks>=80 && marks<90){

         System.out.println("A grade");

     }**else** **if**(marks>=90 && marks<100){

         System.out.println("A+ grade");

     }**else**{

         System.out.println("Invalid!");

     }

}

}

**For loop:**

The Java for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.

There are three types of for loop in java.

* Simple For Loop
* For-each or Enhanced For Loop
* Labeled For Loop

## Java Simple For Loop

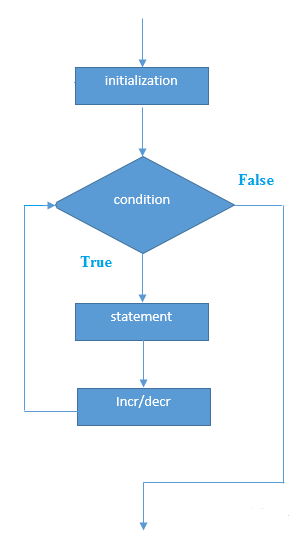
The simple for loop is same as C/C++. We can initialize variable, check condition and increment/decrement value.

**Syntax:**

**for**(initialization;condition;incr/decr){

//code to be executed

}



**Example:**

**public** **class** ForExample {

**public** **static** **void** main(String[] args) {

**for**(**int** i=1;i<=10;i++){

        System.out.println(i);

     }

}

}

## Java For-each Loop

The for-each loop is used to traverse array or collection in java. It is easier to use than simple for loop because we don't need to increment value and use subscript notation.

It works on elements basis not index. It returns element one by one in the defined variable.

**Syntax:**

**for**(Type var:array){

//code to be executed

}

**Example:**

**public** **class** ForEachExample {

**public** **static** **void** main(String[] args) {

**int** arr[]={12,23,44,56,78};

**for**(**int** i:arr){

        System.out.println(i);

    }

}

}

**Java Labeled For Loop**

We can have name of each for loop. To do so, we use label before the for loop. It is useful if we have nested for loop so that we can break/continue specific for loop.

Normally, break and continue keywords breaks/continues the inner most for loop only.

**Syntax:**

labelname:

**for**(initialization;condition;incr/decr){

//code to be executed

}

**Example:**

**public** **class** LabeledForExample {

**public** **static** **void** main(String[] args) {

    aa:

**for**(**int** i=1;i<=3;i++){

            bb:

**for**(**int** j=1;j<=3;j++){

**if**(i==2&&j==2){

**break** aa;

                    }

                    System.out.println(i+" "+j);

                }

        }

}

}

If you use **break bb;**, it will break inner loop only which is the default behavior of any loop.

**public** **class** LabeledForExample {

**public** **static** **void** main(String[] args) {

    aa:

**for**(**int** i=1;i<=3;i++){

            bb:

**for**(**int** j=1;j<=3;j++){

**if**(i==2&&j==2){

**break** bb;

                    }

                    System.out.println(i+" "+j);

                }

        }

}

}

**Java Infinitive For Loop**

If you use two semicolons ;; in the for loop, it will be infinitive for loop.

**Syntax:**

**for**(;;){

//code to be executed

}

**Example:**

**public** **class** ForExample {

**public** **static** **void** main(String[] args) {

**for**(;;){

        System.out.println("infinitive loop");

    }

}

}

**While loop:**

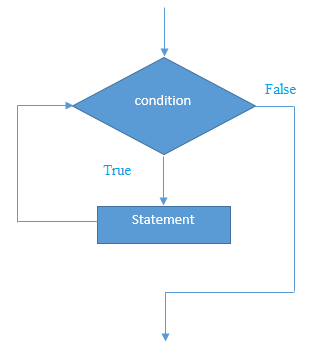
The Java while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

**Syntax:**

**while**(condition){

//code to be executed

}



**Example:**

**public** **class** WhileExample {

**public** **static** **void** main(String[] args) {

**int** i=1;

**while**(i<=10){

        System.out.println(i);

    i++;

    }

}

}

**Java Infinitive While Loop**

If you pass **true** in the while loop, it will be infinitive while loop.

**Syntax:**

**while**(**true**){

//code to be executed

}

**Example:**

**public** **class** WhileExample2 {

**public** **static** **void** main(String[] args) {

**while**(**true**){

        System.out.println("infinitive while loop");

    }

}

}

**Do….While loop:**

The Java do-while loop is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use while loop.

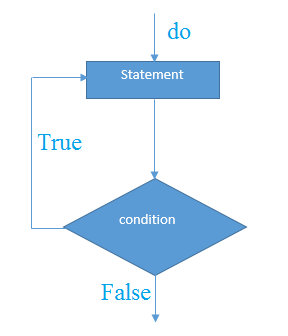
It is executed at least once because condition is checked after loop body.

**Syntax:**

**do**{

//code to be executed

}**while**(condition);



**Example:**

**public** **class** DoWhileExample {

**public** **static** **void** main(String[] args) {

**int** i=1;

**do**{

        System.out.println(i);

    i++;

    }**while**(i<=10);

}

}

## Java Infinitive do-while Loop

If you pass **true** in the do-while loop, it will be infinitive do-while loop.

**Syntax:**

**while**(**true**){

//code to be executed

}

**Example:**

**public** **class** DoWhileExample2 {

**public** **static** **void** main(String[] args) {

**do**{

        System.out.println("infinitive do while loop");

    }**while**(**true**);

}

}

1. **Switch loop:**

Syntax:

Switch ( expression)

{

Case 0:

Statement 1;

Break;

Case 1:

Statement 2;

Break;

.

.

Case n:

Statement n;

Break;

Default:

Statement;

}

**Chapter 2: Classes and Objects**

1. **Class :**

A class is a blue print from which individual objects are created.

A sample of a class is given below:

Public class Car

{

Int speed;

String color;

Void cal\_speed()

{

}

Public static void main(String args[])

{

Car object =new Car();

Object.cal\_speed();

}

}

A class can contain any of the following variable types.

* **Local variables:**Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
* **Instance variables:**Instance variables are variables within a class but outside any method. These variables are initialized when the class is instantiated. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.
* **Class variables:**Class variables are variables declared with in a class, outside any method, with the static keyword.

1. **Methods :**

Syntax :

Public void get(int a)

{

}

Public : Access specifier

Void : return type

get : name of the method

a : formal parameters

1. **Objects :**

Class Demo

{

Public static void main(String args[])

{

Demo d=new Demo(); //d is the object of class Demo

}

}

1. **Constructor :**

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

**Rules for creating java constructor**

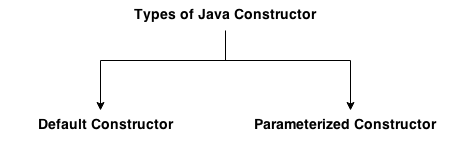
There are basically two rules defined for the constructor.

1. Constructor name must be same as its class name
2. Constructor must have no explicit return type

**Types of java constructors**

There are two types of constructors:

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



1. **Default constructor :**

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

Class Bike

{

Bike()

{

System.out.println(“Bike constructor get invoked”);

}

Public static void main(String ar[])

{

Bike b=new Bike();

}

}

OUTPUT :

Bike constructor get invoked

1. **Parameterized constructor :**

Constructors that have parameters are known as parameterized constructors.

**Why use parameterized constructor?**

Parameterized constructor is used to provide different values to the distinct objects.

Class Bike

{

Bike(int i)

{

Int speed;

Speed=i;

System.out.println(“Bike constructor get invoked having speed” +speed);

}

Public static void main(String ar[])

{

Bike b=new Bike(100);

}

}

OUTPUT :

Bike constructor get invoked having speed 100

1. **Overloading Methods:**

If a class have multiple methods by same name but different parameters, it is known as Method Overloading. Method overloading increases the readability of the program.

Example:

Class Calculate

{

Void add()

{

int a=10, b=30,c;

C=a+b;

System.out.println(“Addition is : “+c);

}

Void add(inta,int b)

{

int c;

C=a+b;

System.out.println(“Addition is : “+c);

}

Public static void main(String ar[])

{

Calculate cal=new Calculate();

Cal.add();

Cal.add(50,60);

}}

1. **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 the unreferenced objects from heap memory.
* It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

**Finalize() method :**

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:

protected void finalize(){ }

**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(){ }

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();

  }

}

1. **Nested Classes :**

In Java, just like methods, variables of a class too can have another class as its member. Writing a class within another is allowed in Java. The class written within is called the **nested class**, and the class that holds the inner class is called the **outer class**.

Example :

classOuter\_Demo

{

classNested\_Demo

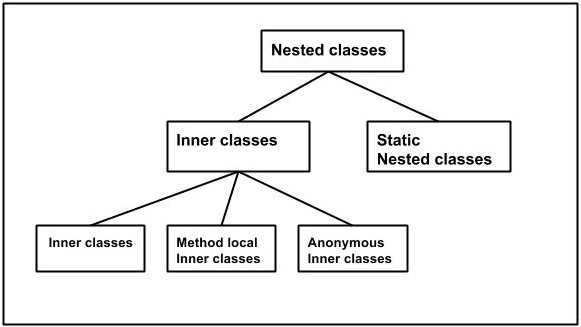
{

}

}

Nested classes are divided into two types:

* **Non-static nested classes:** These are the non-static members of a class.
* **Static nested classes:** These are the static members of a class.



**Chapter 3 : Inheritance**

1. **Inheritance :**

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.

The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

**Extends Keyword :**

**extends** is the keyword used to inherit the properties of a class. Below given is the syntax of extends keyword.

**Example :**

classSuper

{

.....

.....

}

classSub**extends**Super

{

.....

.....

}

1. **Overriding :**

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

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

**Rules :**

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

class Vehicle

{

void run()

{

System.out.println("Vehicle is running");

}

}

class Bike2 extends Vehicle

{

void run()

{

System.out.println("Bike is running safely");

}

  public static void main(String args[])

{

Bike2 obj = new Bike2();

obj.run();

}

}

Output :

Bike is running safely

It doesn’t call super/base class method. So to avoid this super keyword is used.

**Super Keyword :**

The **super** keyword is similar to **this** keyword following are the scenarios where the super keyword is used.

* It is used to **differentiate the members** of superclass from the members of subclass, if they have same names.
* It is used to **invoke the superclass** constructor from subclass.

class Vehicle

{

void run()

{

System.out.println("Vehicle is running");

}

}

class Bike2 extends Vehicle

{

void run()

{

Super.run();

System.out.println("Bike is running safely");

}

   public static void main(String args[])

{

Bike2 obj = new Bike2();

obj.run();

}

}

Output :

Vehicle is running

Bike is running safely

1. **Polymorphism :**

**Polymorphism in java** is a concept by which we can perform a single action by different ways. Polymorphism is derived from 2 greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

There are two types of polymorphism in java: compile time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

If you overload static method in java, it is the example of compile time polymorphism. Here, we will focus on runtime polymorphism in java.

Polymorphism also **represents HAS-A relationship**.

**Run time polymorphism :**

**Runtime polymorphism** or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

Let's first understand the upcasting before Runtime Polymorphism.

Example :

class Animal

{

void eat()

{

System.out.println("eating");

}

}

 class Dog extends Animal

{

void eat()

{

System.out.println("eating fruits");

}

}

 class BabyDog extends Dog

{

void eat()

{

System.out.println("drinking milk");

}

   public static void main(String args[])

{

   Animal a1,a2,a3;

a1=new Animal();

a2=new Dog();

a3=new BabyDog();

   a1.eat();

a2.eat();

a3.eat();

}

}

Output :

eating

eating fruits

drinking Milk

1. **Final Keyword :**

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.

1. **Final variable** :

There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

Example:

class Bike9

{

 final int speedlimit=90;

 void run()

{

  speedlimit=400;

  }

 public static void main(String args[])

{

 Bike9 obj=new  Bike9();

 obj.run();

}

}

Output:

Compile time error.

1. **Final Method** :

If you make any method as final, you cannot override it

class Bike

{

  final void run()

{

System.out.println("running");

}

}

  class Honda extends Bike

{

   void run()

{

System.out.println("running safely with 100kmph");

}

        public static void main(String args[])

{

   Honda honda= new Honda();

   honda.run();

    }

}

Output:

Compile time error.

1. **Final class :**

If you make any class as final, you cannot extend it.

Example :

final class Bike{ }

class Honda1 extends Bike

{

 void run()

{

System.out.println("running safely with 100kmph");

}

      public static void main(String args[])

{

  Honda1 honda= new Honda();

  honda.run();

   }

}

Output:

Compile time error.

1. **Abstract Keyword :**
   1. **Abstract class:**

* 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).
* A class that is declared as abstract is known as **abstract class**. It needs to be extended and its method implemented. It cannot be instantiated.
* Abstract class contain only abstract methods

Example :

Abstract class Car() { }

* 1. **Abstract Method :**

A method that is declared as abstract and does not have implementation is known as abstract method.

Example :

abstract void printStatus();

abstract class Bike

{

  abstract void run();

}

class Honda4 extends Bike

{

void run()

{

System.out.println("running safely..");

}

   public static void main(String args[])

{

 Bike obj = new Honda4();

  obj.run();

}

}

1. **Interfaces :**

* An **interface in java** is a blueprint of a class. It has static constants and abstract methods only.
* The interface in java is **a mechanism to achieve fully abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve fully abstraction and multiple inheritance in Java.
* Java Interface also **represents IS-A relationship**.
* It cannot be instantiated just like abstract class.
* Java does not support multiple inheritance so interfaces are implemented.

Example :

interface printable

{

void print();

}

class A6 implements printable

{

public void print()

{

System.out.println("Hello");

}

   public static void main(String args[])

{

A6 obj = new A6();

obj.print();

  }  }

**Chapter 4 : Exception Handling**

**Exception handling:**

Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc. The core advantage of exception handling is **to maintain the normal flow of the application**.

An exception (or exceptional event) is a problem that arises during the **execution** of a program. When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore these exceptions are to be handled.

An exception can occur for many different reasons, below given are some scenarios where exception occurs.

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

**Example:**

publicclassUnchecked\_Demo

{

publicstaticvoid main(Stringargs[])

{

intnum[]={1,2,3,4};

System.out.println(num[5]);

}

}

Output :

Exceptionin thread "main"java.lang.ArrayIndexOutOfBoundsException:5

atExceptions.Unchecked\_Demo.main(Unchecked\_Demo.java:8)

1. **Try…..catch…..finally**

* **Try block** : The code which has chances to generate an exception i.e. write into try block. When exception occurtey block throw that exception and stop execution.
* **Catch block** : The exception thrown by try block is catched by catch block. It matches type of exception. When exception is catched bye catch block, it is executed.
* **Finally block** : Finally block will be executed after try or catch block i.e finally bock executed whether exception occur or not.
* Try block has catch block and finally block. One try block has multiple catch blocks.

Example :

Class Div

{

Public static void main(String args[])

{

inta,b,c;

try

{

DataInputStream din=new DataInputStream(System.in);

System.out.println(“Enter 1st No”);

a=Integer.parseInt(din.readLine());

System.out.println(“Enter 2nd No”);

b=Integer.parseInt(din.readLine());

c=a/b;

System.out.println(“Division is : “,c);

}

Catch(IOException e)

{

System.out.println(e);

}

Catch(ArithamaticException e1)

{

System.out.println(e1);

}

Finally

{

System.out.println(“I am always execute”);

}

}

1. **Throws Keyword**

Instead of try…catch block we can use throws keyword to handle exception.

Example:

Class Div

{

Public static void main(String args[]) throws Exception

{

inta,b,c;

DataInputStream din=new DataInputStream(System.in);

System.out.println(“Enter 1st No”);

a=Integer.parseInt(din.readLine());

System.out.println(“Enter 2nd No”);

b=Integer.parseInt(din.readLine());

c=a/b;

System.out.println(“Division is : “,c);

}

}

1. **Throw Keyword :**

The java runtime system throw exception bydefault. It is possible to throw an exception explicitly using throw keyword.

Syntax:

Throw throwableinstance(object of exception class)

Example :

classMyexp extends Exception

{

Public Myexp()

{

Public static String toString()

{

Return(“Plz Enter Correct value”);

}

}

}

classexp

{

int a;

public static void main(String args[])throws Exception

{

DataInputStream din=new DataInputStream(System.in);

System.out.println(“Enter No”);

a=Integer.parseInt(din.readLine());

if(a>10)

throw new Myexp();

System.out.println(“Value of A is : “ ,a);

}

}

**Chapter 5: Wrapper Classes**

**Wrapper class in java** provides the mechanism to convert primitive into object and object into primitive*.* One of the eight classes of java.lang package are known as wrapper class in java. The list of eight wrapper classes are given below:

|  |  |
| --- | --- |
| **Primitive Type** | **Wrapper class** |
| Boolean | Boolean |
| Char | Character |
| Byte | Byte |
| Short | Short |
| Int | Integer |
| Long | Long |
| Float | Float |
| Double | Double |

**Example:**

public class WrapperExample1

{

public static void main(String args[])

{

int a=20;

Integer i=Integer.valueOf(a);

Integer j=a;

System.out.println(a+" "+i+" "+j);  } }

**6 : Packages**

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

A Package can be defined as a grouping of related types (classes, interfaces, enumerations and annotations ) providing access protection and name space management.

Some of the existing packages in Java are::

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

Programmers can define their own packages to bundle group of classes/interfaces, etc. It is a good practice to group related classes implemented by you so that a programmer can easily determine that the classes, interfaces, enumerations, annotations are related.

Since the package creates a new namespace there won't be any name conflicts with names in other packages. Using packages, it is easier to provide access control and it is also easier to locate the related classes.

Creating Packages:

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

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

**Example :**

package animals;

interfaceAnimal

{

publicvoid eat();

publicvoid travel();

}

package animals;

public class Cat implements Animal

{

Public void eat()

{

System.out.println("Cat eats");

}

Public void travel()

{

System.out.println("Cat travels");

}

Public int noOfLegs()

{

return0;

}

Public static void main(Stringargs[])

{

Cat c =new Cat();

c.eat();

c.travel();

c.noOfLegs();

}

}

**Chapter 7: Applets**

**Introduction to Applet:**

All Applets are subclasses of Applet class. Applet is dynamic and interactive program i.e. run in java compatible browser. The applet window is create using applet tag. Applets are event driven, it provides user initiates interaction with an applet. Some restriction on Applet:

* Applet doesn’t read and write an file system.
* Applets are run in same server in which they are created.

**Applet tag:**

<applet code = “.class” codebase = “path” width= height= >

</applet>

**Applet LifeCycle:**

1. Init() :

This method get called when applet is started.

1. Start() :

This method is executed after init(). In this case a java enabled browser is used to run the applet, any time it is reloaded and execution begins from start(). Init() called only one time when applet is loaded 1st time and start() called when applet document display on screen.

1. Paint() :

This method help in drawing, writing and creating color background or image on to applet. It takes argument of graphics class.

1. Stop() :

Used to halt the running of an applet.

1. Destroy() :

Used to free memory occupied variables and objects initialized in applet.

1. Repaint() :

Used incase when applet is to be repainted. Repaint() calls update() to clear screen of existing content.

1. Update() :

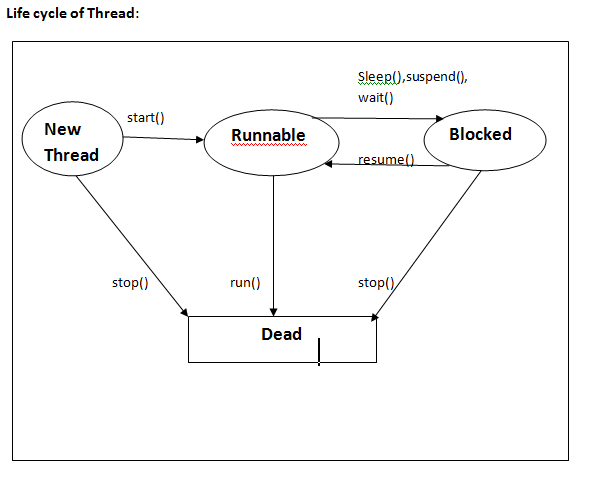
update() used to clear screen of existing content.

**Chapter 8: Swing**

**Chapter 9 : Thread**

* A thread is a line of execution. It is the smallest unit of code that is dispatched by scheduler.
* In single threaded system, there is only one execution line i.e only one part of program is in process of execution at any one time.
* In multithreaded system, a process can contain multiple threads to execute its different section i.e. 2 or more threads run concurrently.
* **Advantages of thread over process:**
  1. Can be created faster.
  2. Require less overhead.
  3. Intercrosses communication is faster.
  4. Context switching is faster.
  5. Maximum use of CPU time.

**Life cycle of Thread**:



**Methods of thread :**

1. Start() :

It starts the execution of invoking object. It can throw an illegal thread state if thread was already executed.

1. Stop() :

This method terminate the invoking object.

1. Suspend() :

This suspend the invoking object. The thread will become runnable again if its get the resume().

1. Sleep() :

This method suspend the execution of executing thread for specified no of milliseconds. Ex : sleep( long milisec)

1. Resume() :

This method restart the suspended thread at the point at which it was halted. This method is called by some other thread outside the suspended one.

**Constructor of thread:**

1. Thread(runnable) :

It create thread object for given class.

1. Thread(runnable , string) :

It create thread object for given class with given menu.

Example:

Create thread to display even no between 1 to 10 and odd numbers between 1 to 10.

Class Even extends Thread

{

Thread t=null;

Even()

{

t=new Thread(this);

t.start();

}

Public void run() throws Exception

{

For(int i=0 ; i<=10 ; i+=2)

{

System.out.println(“Even no= “ +i);

Thread.sleep(500);

}

}

}

Class Odd extends Thread

{

Thread t1=null;

Odd()

{

t1=new Thread(this);

t1.start();

}

Public void run() throws Exception

{

For(int j=1 ; j<=10 ; j+=2)

{

System.out.println(“Odd no= “ +j);

Thread.sleep(500);

}

}

}

Class ThreadDemo

{

Public static void main(String ar[])

{

Even e=new Even();

Odd o=new Odd();

}

}

Synchronization :

Two or more threads accessing the same data simultaneously may need the loss of data integrity. Java uses concept of monitor to enable this monitor is an object used as mutually exclusive lock. Hence, at a time only one thread can access monitor.2nd thread can not enter the monitor until the first comes out.

Example:

Class Book

{

Intbook\_id,cnt;

String book\_name;

Public void get()

{ }

Public void display()

{ }

}

Class Student implements Runnable

{

Book b;

Thread t=NULL;

Student(Book x)

{

b=x;

t=new Thread(this);

t.start();

}

Synchronized void issue(Book b)

{

If(b.cnt> 0 )

cnt --;

}

public void run() throws Exception

{

t.sleep(500);

}

issue(b);

}

Class Demo

{

Public static void main(String ar[])

{

Book b=new Book();

b.get();

Student s1 = new Student(b);

Student s2 = new Student(b);

}

}

**Chapter 10 : I/O packages**

**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.If you have to write primitive values then use FileOutputStream.Instead, for character-oriented data, prefer FileWriter.But you can write byte-oriented as well as character-oriented data.

**Example of Java FileOutputStream class**

**import** java.io.\*;

**class** Test{

**public** **static** **void** main(String args[]){

**try**{

FileOutputstream fout=**new** FileOutputStream("abc.txt");

String s="Sachin Tendulkar is my favourite player";

**byte** b[]=s.getBytes();//converting string into byte array

fout.write(b);

fout.close();

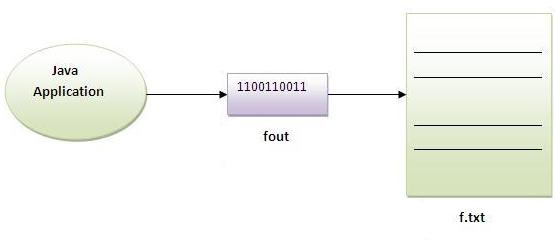
System.out.println("success...");

}**catch**(Exception e){system.out.println(e);}

}

}

Output:success...



**Java FileInputStream class**

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

Example of FileInputStream class

**import** java.io.\*;

**class** SimpleRead{

**public** **static** **void** main(String args[]){

**try**{

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

**int** i=0;

**while**((i=fin.read())!=-1){

System.out.println((**char**)i);

}

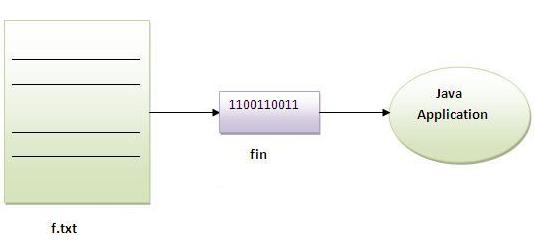
fin.close();

}**catch**(Exception e){system.out.println(e);}

}

}

Output:Sachin is my favourite player.



Example of Reading the data of current java file and writing it into another file

We can read the data of any file using the FileInputStream class whether it is java file, image file, video file etc. In this example, we are reading the data of C.java file and writing it into another file M.java.

**import** java.io.\*;

**class** C{

**public** **static** **void** main(String args[])**throws** Exception{

FileInputStream fin=**new** FileInputStream("C.java");

FileOutputStream fout=**new** FileOutputStream("M.java");

**int** i=0;

**while**((i=fin.read())!=-1){

fout.write((**byte**)i);

}

fin.close();

}

}

# Java ByteArrayOutputStream class

Java ByteArrayOutputStream class is used to write data into multiple files. In this stream, the data is written into a byte array that can be written to multiple stream. The ByteArrayOutputStream holds a copy of data and forwards it to multiple streams. The buffer of ByteArrayOutputStream automatically grows according to data.

#### Closing the ByteArrayOutputStream has no effect.

### Constructors of ByteArrayOutputStream class

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| ByteArrayOutputStream() | creates a new byte array output stream with the initial capacity of 32 bytes, though its size increases if necessary. |
| ByteArrayOutputStream(int size) | creates a new byte array output stream, with a buffer capacity of the specified size, in bytes. |

### Methods of ByteArrayOutputStream class

|  |  |
| --- | --- |
| **Method** | **Description** |
| 1) public synchronized void writeTo(OutputStream out) throws IOException | writes the complete contents of this byte array output stream to the specified output stream. |
| 2) public void write(byte b) throws IOException | writes byte into this stream. |
| 3) public void write(byte[] b) throws IOException | writes byte array into this stream. |
| 4) public void flush() | flushes this stream. |
| 5) public void close() | has no affect, it doesn't closes the bytearrayoutputstream. |

### Java ByteArrayOutputStream Example

Let's see a simple example of java ByteArrayOutputStream class to write data into 2 files.

**import** java.io.\*;

**class** S{

**public** **static** **void** main(String args[])**throws** Exception{

FileOutputStream fout1=**new** FileOutputStream("f1.txt");

FileOutputStream fout2=**new** FileOutputStream("f2.txt");

ByteArrayOutputStream bout=**new** ByteArrayOutputStream();

bout.write(139);

bout.writeTo(fout1);

bout.writeTo(fout2);

bout.flush();

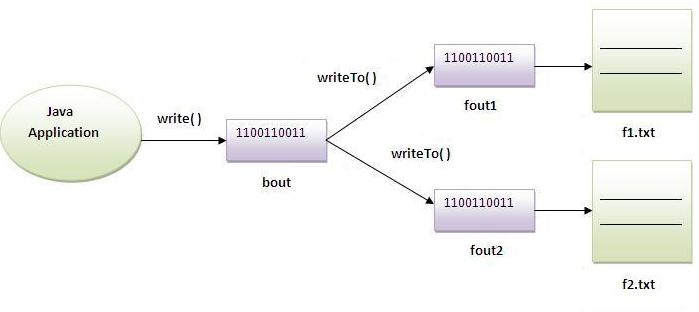
bout.close();//has no effect

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

}

}

success...



**Java BufferedOutputStream and BufferedInputStream**

**Java BufferedOutputStream class**

Java BufferedOutputStream class uses an internal buffer to store data. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast.

**Example of BufferedOutputStream class:**

In this example, we are writing the textual information in the BufferedOutputStream object which is connected to the FileOutputStream object. The flush() flushes the data of one stream and send it into another. It is required if you have connected the one stream with another.

**import** java.io.\*;

**class** Test{

**public** **static** **void** main(String args[])**throws** Exception{

FileOutputStream fout=**new** FileOutputStream("f1.txt");

BufferedOutputStream bout=**new** BufferedOutputStream(fout);

String s="Sachin is my favourite player";

**byte** b[]=s.getBytes();

bout.write(b);

bout.flush();

bout.close();

fout.close();

System.out.println("success");

}

}

Output:

success...

**Java BufferedInputStream class**

Java BufferedInputStream class is used to read information from stream. It internally uses buffer mechanism to make the performance fast.

**Example of Java BufferedInputStream**

Let's see the simple example to read data of file using BufferedInputStream.

**import** java.io.\*;

**class** SimpleRead{

**public** **static** **void** main(String args[]){

**try**{

FileInputStream fin=**new** FileInputStream("f1.txt");

BufferedInputStream bin=**new** BufferedInputStream(fin);

**int** i;

**while**((i=bin.read())!=-1){

System.out.println((**char**)i);

}

bin.close();

fin.close();

}**catch**(Exception e){system.out.println(e);}

}

}

Output:

Sachin is my favourite player

**Java FileWriter and FileReader (File Handling in java)**

Java FileWriter and FileReader classes are used to write and read data from text files. These are character-oriented classes, used for file handling in java. Java has suggested not to use the FileInputStream and FileOutputStream classes if you have to read and write the textual information.

**Java FileWriter class**

Java FileWriter class is used to write character-oriented data to the file. Constructors of FileWriter class

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| FileWriter(String file) | creates a new file. It gets file name in string. |
| FileWriter(File file) | creates a new file. It gets file name in File object. |

Methods of FileWriter class

|  |  |
| --- | --- |
| **Method** | **Description** |
| 1) public void write(String text) | writes the string into FileWriter. |
| 2) public void write(char c) | writes the char into FileWriter. |
| 3) public void write(char[] c) | writes char array into FileWriter. |
| 4) public void flush() | flushes the data of FileWriter. |
| 5) public void close() | closes FileWriter. |

**Java FileWriter Example**

In this example, we are writing the data in the file abc.txt.

**import** java.io.\*;

**class** Simple{

**public** **static** **void** main(String args[]){

**try**{

FileWriter fw=**new** FileWriter("abc.txt");

fw.write("my name is sachin");

fw.close();

}**catch**(Exception e){System.out.println(e);}

System.out.println("success");

}

}

Output:

success...

**Java FileReader class**

Java FileReader class is used to read data from the file. It returns data in byte format like FileInputStream class.

Constructors of FileWriter class

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| FileReader(String file) | It gets filename in string. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException. |
| FileReader(File file) | It gets filename in file instance. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException. |

Methods of FileReader class

|  |  |
| --- | --- |
| **Method** | **Description** |
| 1) public int read() | returns a character in ASCII form. It returns -1 at the end of file. |
| 2) public void close() | closes FileReader. |

Java FileReader Example

In this example, we are reading the data from the file abc.txt file.

**import** java.io.\*;

**class** Simple{

**public** **static** **void** main(String args[])**throws** Exception{

FileReader fr=**new** FileReader("abc.txt");

**int** i;

**while**((i=fr.read())!=-1)

System.out.println((**char**)i);

fr.close();

}

}

Output: my name is sachin

**Reading data from keyboard**

There are many ways to read data from the keyboard. For example:

* InputStreamReader
* Console
* Scanner
* DataInputStream etc.

**InputStreamReader class**

InputStreamReader class can be used to read data from keyboard.It performs two tasks:

* connects to input stream of keyboard
* converts the byte-oriented stream into character-oriented stream

**BufferedReader class**

BufferedReader class can be used to read data line by line by readLine() method.

Example of reading data from keyboard by InputStreamReader and BufferdReader class

In this example, we are connecting the BufferedReader stream with the InputStreamReader stream for reading the line by line data from the keyboard.

**import** java.io.\*;

**class** G5{

**public** **static** **void** main(String args[])**throws** Exception{

InputStreamReader r=**new** InputStreamReader(System.in);

BufferedReader br=**new** BufferedReader(r);

System.out.println("Enter your name");

String name=br.readLine();

System.out.println("Welcome "+name);

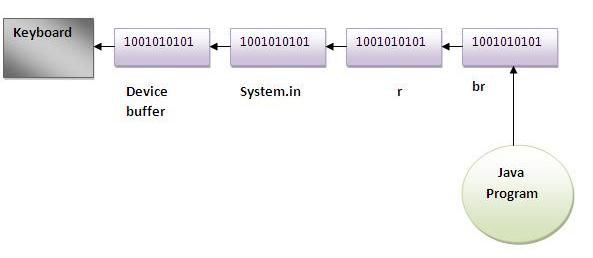
}

}

Output:Enter your name

Amit

Welcome Amit



Another Example of reading data from keyboard by InputStreamReader and BufferdReader class until the user writes stop

In this example, we are reading and printing the data until the user prints stop.

**import** java.io.\*;

**class** G5{

**public** **static** **void** main(String args[])**throws** Exception{

InputStreamReader r=**new** InputStreamReader(System.in);

BufferedReader br=**new** BufferedReader(r);

String name="";

**while**(!name.equals("stop")){

System.out.println("Enter data: ");

name=br.readLine();

System.out.println("data is: "+name);

}

br.close();

r.close();

}

}

Output:Enter data: Amit

data is: Amit

Enter data: 10

data is: 10

Enter data: stop

data is: stop

**Java Console class**

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

**String text=System.console().readLine();**

**System.out.println("Text is: "+text);**

Methods of Console class

Let's see the commonly used methods of Console class.

|  |  |
| --- | --- |
| **Method** | **Description** |
| 1) public String readLine() | is used to read a single line of text from the console. |
| 2) public String readLine(String fmt,Object... args) | it provides a formatted prompt then reads the single line of text from the console. |
| 3) public char[] readPassword() | is used to read password that is not being displayed on the console. |
| 4) public char[] readPassword(String fmt,Object... args) | it provides a formatted prompt then reads the password that is not being displayed on the console. |

How to get the object of Console

System class provides a static method console() that returns the unique instance of Console class.

1. **public** **static** Console console(){}

Let's see the code to get the instance of Console class.

1. Console c=System.console();

Java Console Example

**import** java.io.\*;

**class** ReadStringTest{

**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: james gosling

Welcome james gosling

Java Console Example to read password

**import** java.io.\*;

**class** ReadPasswordTest{

**public** **static** **void** main(String args[]){

Console c=System.console();

System.out.println("Enter password: ");

**char**[] ch=c.readPassword();

String pass=String.valueOf(ch);//converting char array into string

System.out.println("Password is: "+pass);

}

}

Output:

Enter password:

Password is: sonoo

**Java Scanner class**

There are various ways to read input from the keyboard, the java.util.Scanner class is one of them. The **Java Scanner** class breaks the input into tokens using a delimiter that is whitespace bydefault. It provides many methods to read and parse various primitive values. Java Scanner class is widely used to parse text for string and primitive types using regular expression. Java Scanner class extends Object class and implements Iterator and Closeable interfaces.

Commonly used methods of Scanner class

There is a list of commonly used Scanner class methods:

|  |  |
| --- | --- |
| **Method** | **Description** |
| public String next() | it returns the next token from the scanner. |
| public String nextLine() | it moves the scanner position to the next line and returns the value as a string. |
| public byte nextByte() | it scans the next token as a byte. |
| public short nextShort() | it scans the next token as a short value. |
| public int nextInt() | it scans the next token as an int value. |
| public long nextLong() | it scans the next token as a long value. |
| public float nextFloat() | it scans the next token as a float value. |
| public double nextDouble() | it scans the next token as a double value. |

**Java Scanner Example to get input from console**

Let's see the simple example of the Java Scanner class which reads the int, string and double value as an input:

**import** java.util.Scanner;

**class** ScannerTest{

**public** **static** **void** main(String args[]){

Scanner sc=**new** Scanner(System.in);

System.out.println("Enter your rollno");

**int** rollno=sc.nextInt();

System.out.println("Enter your name");

String name=sc.next();

System.out.println("Enter your fee");

**double** fee=sc.nextDouble();

System.out.println("Rollno:"+rollno+" name:"+name+" fee:"+fee);

sc.close();

}

}

[download this scanner example](http://www.javatpoint.com/src/io/scanner.zip)

Output:

Enter your rollno

111

Enter your name

Ratan

Enter

450000

Rollno:111 name:Ratan fee:450000

**Java Scanner Example with delimiter**

Let's see the example of Scanner class with delimiter. The \s represents whitespace.

**import** java.util.\*;

**public** **class** ScannerTest2{

**public** **static** **void** main(String args[]){

String input = "10 tea 20 coffee 30 tea buiscuits";

Scanner s = **new** Scanner(input).useDelimiter("\\s");

System.out.println(s.nextInt());

System.out.println(s.next());

System.out.println(s.nextInt());

System.out.println(s.next());

s.close();}}

**Java AWT**

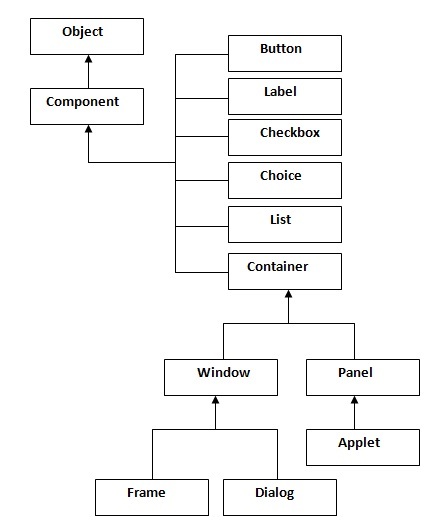
**Java AWT** (Abstract Windowing Toolkit) is *an API to develop GUI or window-based application in java*.

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

Java AWT Hierarchy

The hierarchy of Java AWT classes are given below.



Container

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

Window

The window is the container that have 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. |

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)
* 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, by default not visible

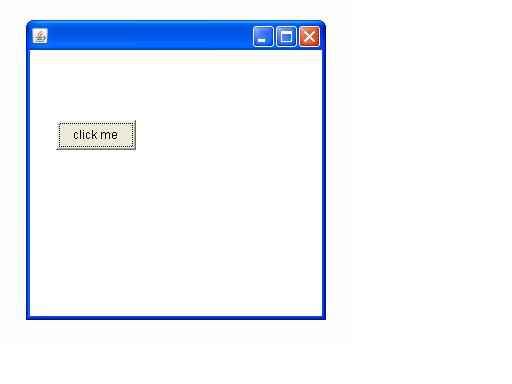
}

**public** **static** **void** main(String args[]){

First f=**new** First();

}}

The setBounds(int xaxis, int yaxis, int width, int height) method is used in the above example that sets the position of the awt button.



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();

}}

Event and Listener (Java Event Handling)

|  |
| --- |
| Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling. |

Event classes and Listener interfaces:

|  |  |
| --- | --- |
| **Event Classes** | **Listener Interfaces** |
| ActionEvent | ActionListener |
| MouseEvent | MouseListener and MouseMotionListener |
| MouseWheelEvent | MouseWheelListener |
| KeyEvent | KeyListener |
| ItemEvent | ItemListener |
| TextEvent | TextListener |
| AdjustmentEvent | AdjustmentListener |
| WindowEvent | WindowListener |
| ComponentEvent | ComponentListener |
| ContainerEvent | ContainerListener |
| FocusEvent | FocusListener |

Steps to perform Event Handling

Following steps are required to perform event handling:

1. Implement the Listener interface and overrides its methods
2. Register the component with the Listener

For registering the component with the Listener, many classes provide the registration methods. For example:

* **Button**
  + public void addActionListener(ActionListener a){}
* **MenuItem**
  + public void addActionListener(ActionListener a){}
* **TextField**
  + public void addActionListener(ActionListener a){}
  + public void addTextListener(TextListener a){}
* **TextArea**
  + public void addTextListener(TextListener a){}
* **Checkbox**
  + public void addItemListener(ItemListener a){}
* **Choice**
  + public void addItemListener(ItemListener a){}
* **List**
  + public void addActionListener(ActionListener a){}
  + public void addItemListener(ItemListener a){}

EventHandling Codes:

|  |
| --- |
| We can put the event handling code into one of the following places:   1. Same class 2. Other class 3. Annonymous class |

Example of event handling within class:

**import** java.awt.\*;

**import** java.awt.event.\*;

**class** AEvent **extends** Frame **implements** ActionListener{

TextField tf;

AEvent(){

tf=**new** TextField();

tf.setBounds(60,50,170,20);

Button b=**new** Button("click me");

b.setBounds(100,120,80,30);

b.addActionListener(**this**);

add(b);add(tf);

setSize(300,300);

setLayout(**null**);

setVisible(**true**);

}

**public** **void** actionPerformed(ActionEvent e){

tf.setText("Welcome");

}

**public** **static** **void** main(String args[]){

**new** AEvent();

}

}

|  |
| --- |
| **public void setBounds(int xaxis, int yaxis, int width, int height);** have been used in the above example that sets the position of the component it may be button, textfield etc. |



2) Example of event handling by Outer class:

**import** java.awt.\*;

**import** java.awt.event.\*;

**class** AEvent2 **extends** Frame{

TextField tf;

AEvent2(){

tf=**new** TextField();

tf.setBounds(60,50,170,20);

Button b=**new** Button("click me");

b.setBounds(100,120,80,30);

Outer o=**new** Outer(**this**);

b.addActionListener(o);//passing outer class instance

add(b);add(tf);

setSize(300,300);

setLayout(**null**);

setVisible(**true**);

}

**public** **static** **void** main(String args[]){

**new** AEvent2();

}

}

**import** java.awt.event.\*;

**class** Outer **implements** ActionListener{

AEvent2 obj;

Outer(AEvent2 obj){

**this**.obj=obj;

}

**public** **void** actionPerformed(ActionEvent e){

obj.tf.setText("welcome");

}

}

3) Example of event handling by Annonymous class:

**import** java.awt.\*;

**import** java.awt.event.\*;

**class** AEvent3 **extends** Frame{

TextField tf;

AEvent3(){

tf=**new** TextField();

tf.setBounds(60,50,170,20);

Button b=**new** Button("click me");

b.setBounds(50,120,80,30);

b.addActionListener(**new** ActionListener(){

**public** **void** actionPerformed(){

tf.setText("hello");

}

});

add(b);add(tf);

setSize(300,300);

setLayout(**null**);

setVisible(**true**);

}

**public** **static** **void** main(String args[]){

**new** AEvent3();

}

}

## Graphical User Interface

Graphical User Interface (GUI) offers user interaction via some graphical components. For example our underlying Operating System also offers GUI via window,frame,Panel, Button, Textfield, TextArea, Listbox, Combobox, Label, Checkbox etc. These all are known as components. Using these components we can create an interactive user interface for an application.

GUI provides result to end user in response to raised events.GUI is entirely based events. For example clicking over a button, closing a window, opening a window, typing something in a textarea etc. These activities are known as events.GUI makes it easier for the end user to use an application. It also makes them interesting.

## Basic Terminologies

|  |  |
| --- | --- |
| **Term** | **Description** |
| Component | Component is an object having a graphical representation that can be displayed on the screen and that can interact with the user. For examples buttons, checkboxes, list and scrollbars of a graphical user interface. |
| Container | Container object is a component that can contain other components.Components added to a container are tracked in a list. The order of the list will define the components' front-to-back stacking order within the container. If no index is specified when adding a component to a container, it will be added to the end of the list. |
| Panel | Panel provides space in which an application can attach any other components, including other panels. |
| Window | Window is a rectangular area which is displayed on the screen. In different window we can execute different program and display different data. Window provide us with multitasking environment. A window must have either a frame, dialog, or another window defined as its owner when it's constructed. |
| Frame | A Frame is a top-level window with a title and a border. The size of the frame includes any area designated for the border. Frame encapsulates **window**. It and has a title bar, menu bar, borders, and resizing corners. |
| Canvas | Canvas component represents a blank rectangular area of the screen onto which the application can draw. Application can also trap input events from the use from that blank area of Canvas component. |

## Examples of GUI based Applications

Following are some of the examples for GUI based applications.

* Automated Teller Machine (ATM)
* Airline Ticketing System
* Information Kiosks at railway stations
* Mobile Applications
* Navigation Systems

## Advantages of GUI over CUI

* GUI provides graphical icons to interact while the CUI (Character User Interface) offers the simple text-based interfaces.
* GUI makes the application more entertaining and interesting on the other hand CUI does not.
* GUI offers click and execute environment while in CUI every time we have to enter the command for a task.
* New user can easily interact with graphical user interface by the visual indicators but it is difficult in Character user interface.
* GUI offers a lot of controls of file system and the operating system while in CUI you have to use commands which is difficult to remember.
* Windows concept in GUI allow the user to view, manipulate and control the multiple applications at once while in CUI user can control one task at a time.
* GUI provides multitasking environment so as the CUI also does but CUI does not provide same ease as the GUI do.Using GUI it is easier to control and navigate the operating system which becomes very slow in command user interface. GUI can be easily customized.

# AWT Label Class

## Introduction

Label is a passive control because it does not create any event when accessed by the user. The label control is an object of Label. A label displays a single line of read-only text. However the text can be changed by the application programmer but cannot be changed by the end user in any way.

## Class declaration

Following is the declaration for **java.awt.Label** class:

public class Label

extends Component

implements Accessible

## Field

Following are the fields for **java.awt.Component** class:

* **static int CENTER** -- Indicates that the label should be centered.
* **static int LEFT** -- Indicates that the label should be left justified.
* **static int RIGHT** -- Indicates that the label should be right justified.

## Class constructors

|  |  |
| --- | --- |
| **S.N.** | **Constructor & Description** |
| 1 | **Label()**  Constructs an empty label. |
| 2 | **Label(String text)**  Constructs a new label with the specified string of text, left justified. |
| 3 | **Label(String text, int alignment)**  Constructs a new label that presents the specified string of text with the specified alignment. |

## Class methods

|  |  |
| --- | --- |
| **S.N.** | **Method & Description** |
| 1 | **void addNotify()**  Creates the peer for this label. |
| 2 | **AccessibleContext getAccessibleContext()**  Gets the AccessibleContext associated with this Label. |
| 3 | **int getAlignment()**  Gets the current alignment of this label. |
| 4 | **String getText()**  Gets the text of this label. |
| 5 | **protected String paramString()**  Returns a string representing the state of this Label. |
| 6 | **void setAlignment(int alignment)**  Sets the alignment for this label to the specified alignment. |
| 7 | **void setText(String text)**  Sets the text for this label to the specified text. |

## Methods inherited

This class inherits methods from the following classes:

* java.awt.Component
* java.lang.Object

**AwtControlDemo.java**

import java.awt.\*;

import java.awt.event.\*;

public class AwtControlDemo {

private Frame mainFrame;

private Label headerLabel;

private Label statusLabel;

private Panel controlPanel;

public AwtControlDemo(){

prepareGUI();

}

public static void main(String[] args){

AwtControlDemo awtControlDemo = new AwtControlDemo();

awtControlDemo.showLabelDemo();

}

private void prepareGUI(){

mainFrame = new Frame("Java AWT Examples");

mainFrame.setSize(400,400);

mainFrame.setLayout(new GridLayout(3, 1));

mainFrame.addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent windowEvent){

System.exit(0);

}

});

headerLabel = new Label();

headerLabel.setAlignment(Label.CENTER);

statusLabel = new Label();

statusLabel.setAlignment(Label.CENTER);

statusLabel.setSize(350,100);

controlPanel = new Panel();

controlPanel.setLayout(new FlowLayout());

mainFrame.add(headerLabel);

mainFrame.add(controlPanel);

mainFrame.add(statusLabel);

mainFrame.setVisible(true);

}

private void showLabelDemo(){

headerLabel.setText("Control in action: Label");

Label label = new Label();

label.setText("Welcome to TutorialsPoint AWT Tutorial.");

label.setAlignment(Label.CENTER);

label.setBackground(Color.GRAY);

label.setForeground(Color.WHITE);

controlPanel.add(label);

mainFrame.setVisible(true);

}

}

## Introduction

Button is a control component that has a label and generates an event when pressed. When a button is pressed and released, AWT sends an instance of ActionEvent to the button, by calling processEvent on the button. The button's processEvent method receives all events for the button; it passes an action event along by calling its own processActionEvent method. The latter method passes the action event on to any action listeners that have registered an interest in action events generated by this button.

If an application wants to perform some action based on a button being pressed and released, it should implement ActionListener and register the new listener to receive events from this button, by calling the button's addActionListener method. The application can make use of the button's action command as a messaging protocol.

## Class declaration

Following is the declaration for **java.awt.Button** class:

public class Button

extends Component

implements Accessible

## Class constructors

|  |  |
| --- | --- |
| **S.N.** | **Constructor & Description** |
| 1 | **Button()**  Constructs a button with an empty string for its label. |
| 2 | **Button(String text)**  Constructs a new button with specified label. |

## Class methods

|  |  |
| --- | --- |
| **S.N.** | **Method & Description** |
| 1 | **void addActionListener(ActionListener l)**  Adds the specified action listener to receive action events from this button. |
| 2 | **void addNotify()**  Creates the peer of the button. |
| 3 | **AccessibleContext getAccessibleContext()**  Gets the AccessibleContext associated with this Button. |
| 4 | **String getActionCommand()**  Returns the command name of the action event fired by this button. |
| 5 | **ActionListener[] getActionListeners()**  Returns an array of all the action listeners registered on this button. |
| 6 | **String getLabel()**  Gets the label of this button. |
| 7 | **<T extends EventListener> T[] getListeners(Class<T> listenerType)**  Returns an array of all the objects currently registered as FooListeners upon this Button. |
| 8 | **protected String paramString()**  Returns a string representing the state of this Button. |
| 9 | **protected void processActionEvent(ActionEvent e)**  Processes action events occurring on this button by dispatching them to any registered ActionListener objects. |
| 10 | **protected void processEvent(AWTEvent e)**  Processes events on this button. |
| 11 | **void removeActionListener(ActionListener l)**  Removes the specified action listener so that it no longer receives action events from this button. |
| 12 | **void setActionCommand(String command)**  Sets the command name for the action event fired by this button. |
| 13 | **void setLabel(String label)**  Sets the button's label to be the specified string. |

## Methods inherited

This class inherits methods from the following classes:

* java.awt.Component
* java.lang.Object

**AwtControlDemo.java**

import java.awt.\*;

import java.awt.event.\*;

public class AwtControlDemo {

private Frame mainFrame;

private Label headerLabel;

private Label statusLabel;

private Panel controlPanel;

public AwtControlDemo(){

prepareGUI();

}

public static void main(String[] args){

AwtControlDemo awtControlDemo = new AwtControlDemo();

awtControlDemo.showButtonDemo();

}

private void prepareGUI(){

mainFrame = new Frame("Java AWT Examples");

mainFrame.setSize(400,400);

mainFrame.setLayout(new GridLayout(3, 1));

mainFrame.addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent windowEvent){

System.exit(0);

}

});

headerLabel = new Label();

headerLabel.setAlignment(Label.CENTER);

statusLabel = new Label();

statusLabel.setAlignment(Label.CENTER);

statusLabel.setSize(350,100);

controlPanel = new Panel();

controlPanel.setLayout(new FlowLayout());

mainFrame.add(headerLabel);

mainFrame.add(controlPanel);

mainFrame.add(statusLabel);

mainFrame.setVisible(true);

}

private void showButtonDemo(){

headerLabel.setText("Control in action: Button");

Button okButton = new Button("OK");

Button submitButton = new Button("Submit");

Button cancelButton = new Button("Cancel");

okButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

statusLabel.setText("Ok Button clicked.");

}

});

submitButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

statusLabel.setText("Submit Button clicked.");

}

});

cancelButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

statusLabel.setText("Cancel Button clicked.");

}

});

controlPanel.add(okButton);

controlPanel.add(submitButton);

controlPanel.add(cancelButton);

mainFrame.setVisible(true);

}

}

# AWT CheckBox Class

## Introduction

Checkbox control is used to turn an option on(true) or off(false). There is label for each checkbox representing what the checkbox does.The state of a checkbox can be changed by clicking on it.

## Class declaration

Following is the declaration for **java.awt.Checkbox** class:

public class Checkbox

extends Component

implements ItemSelectable,Accessible

## Class constructors

|  |  |
| --- | --- |
| **S.N.** | **Constructor & Description** |
| 1 | **Checkbox()**  Creates a check box with an empty string for its label. |
| 2 | **Checkbox(String label)**  Creates a check box with the specified label. |
| 3 | **Checkbox(String label, boolean state)**  Creates a check box with the specified label and sets the specified state. |
| 4 | **Checkbox(String label, boolean state, CheckboxGroup group)**  Constructs a Checkbox with the specified label, set to the specified state, and in the specified check box group. |
| 5 | **Checkbox(String label, CheckboxGroup group, boolean state)**  Creates a check box with the specified label, in the specified check box group, and set to the specified state. |

## Class methods

|  |  |
| --- | --- |
| **S.N.** | **Method & Description** |
| 1 | **void addItemListener(ItemListener l)**  Adds the specified item listener to receive item events from this check box. |
| 2 | **void addNotify()**  Creates the peer of the Checkbox. |
| 3 | **AccessibleContext getAccessibleContext()**  Gets the AccessibleContext associated with this Checkbox. |
| 4 | **CheckboxGroup getCheckboxGroup()**  Determines this check box's group. |
| 5 | **ItemListener[] getItemListeners()**  Returns an array of all the item listeners registered on this checkbox. |
| 6 | **String getLabel()**  Gets the label of this check box. |
| 7 | **<T extends EventListener>T[] getListeners(Class<T> listenerType)**  Returns an array of all the objects currently registered as FooListeners upon this Checkbox. |
| 8 | **Object[] getSelectedObjects()**  Returns an array (length 1) containing the checkbox label or null if the checkbox is not selected. |
| 9 | **boolean getState()**  Determines whether this check box is in the **on** or **off** state. |
| 10 | **protected String paramString()**  Returns a string representing the state of this Checkbox. |
| 11 | **protected void processEvent(AWTEvent e)**  Processes events on this check box. |
| 12 | **protected void processItemEvent(ItemEvent e)**  Processes item events occurring on this check box by dispatching them to any registered ItemListener objects. |
| 13 | **void removeItemListener(ItemListener l)**  Removes the specified item listener so that the item listener no longer receives item events from this check box. |
| 14 | **void setCheckboxGroup(CheckboxGroup g)**  Sets this check box's group to the specified check box group. |
| 15 | **void setLabel(String label)**  Sets this check box's label to be the string argument. |
| 16 | **void setState(boolean state)**  Sets the state of this check box to the specified state. |

## Methods inherited

This class inherits methods from the following classes:

* java.awt.Component
* java.lang.Object

**AwtControlDemo.java**

import java.awt.\*;

import java.awt.event.\*;

public class AwtControlDemo {

private Frame mainFrame;

private Label headerLabel;

private Label statusLabel;

private Panel controlPanel;

public AwtControlDemo(){

prepareGUI();

}

public static void main(String[] args){

AwtControlDemo awtControlDemo = new AwtControlDemo();

awtControlDemo.showCheckBoxDemo();

}

private void prepareGUI(){

mainFrame = new Frame("Java AWT Examples");

mainFrame.setSize(400,400);

mainFrame.setLayout(new GridLayout(3, 1));

mainFrame.addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent windowEvent){

System.exit(0);

}

});

headerLabel = new Label();

headerLabel.setAlignment(Label.CENTER);

statusLabel = new Label();

statusLabel.setAlignment(Label.CENTER);

statusLabel.setSize(350,100);

controlPanel = new Panel();

controlPanel.setLayout(new FlowLayout());

mainFrame.add(headerLabel);

mainFrame.add(controlPanel);

mainFrame.add(statusLabel);

mainFrame.setVisible(true);

}

private void showCheckBoxDemo(){

headerLabel.setText("Control in action: CheckBox");

Checkbox chkApple = new Checkbox("Apple");

Checkbox chkMango = new Checkbox("Mango");

Checkbox chkPeer = new Checkbox("Peer");

chkApple.addItemListener(new ItemListener() {

public void itemStateChanged(ItemEvent e) {

statusLabel.setText("Apple Checkbox: "

+ (e.getStateChange()==1?"checked":"unchecked"));

}

});

chkMango.addItemListener(new ItemListener() {

public void itemStateChanged(ItemEvent e) {

statusLabel.setText("Mango Checkbox: "

+ (e.getStateChange()==1?"checked":"unchecked"));

}

});

chkPeer.addItemListener(new ItemListener() {

public void itemStateChanged(ItemEvent e) {

statusLabel.setText("Peer Checkbox: "

+ (e.getStateChange()==1?"checked":"unchecked"));

}

});

controlPanel.add(chkApple);

controlPanel.add(chkMango);

controlPanel.add(chkPeer);

mainFrame.setVisible(true);

}

}

# AWT List Class

## Introduction

The List represents a list of text items. The list can be configured to that user can choose either one item or multiple items.

## Class declaration

Following is the declaration for **java.awt.List** class:

public class List

extends Component

implements ItemSelectable, Accessible

## Class constructors

|  |  |
| --- | --- |
| **S.N.** | **Constructor & Description** |
| 1 | **List()**  Creates a new scrolling list. |
| 2 | **List(int rows)**  Creates a new scrolling list initialized with the specified number of visible lines. |
| 3 | **List(int rows, boolean multipleMode)**  Creates a new scrolling list initialized to display the specified number of rows. |

## Class methods

<T extends EventListener> T[] getListeners(Class<T> listenerType)

Returns an array of all the objects currently registered as FooListeners upon this List.

|  |  |
| --- | --- |
| **S.N.** | **Method & Description** |
| 1 | **void add(String item)**  Adds the specified item to the end of scrolling list. |
| 2 | **void add(String item, int index)**  Adds the specified item to the the scrolling list at the position indicated by the index. |
| 3 | **void addActionListener(ActionListener l)**  Adds the specified action listener to receive action events from this list. |
| 4 | **void addItem(String item)**  Deprecated. replaced by add(String). |
| 5 | **void addItem(String item, int index)**  Deprecated. replaced by add(String, int). |
| 6 | **void addItemListener(ItemListener l)**  Adds the specified item listener to receive item events from this list. |
| 7 | **void addNotify()**  Creates the peer for the list. |
| 8 | **boolean allowsMultipleSelections()**  Deprecated. As of JDK version 1.1, replaced by isMultipleMode(). |
| 9 | **void clear()**  Deprecated. As of JDK version 1.1, replaced by removeAll(). |
| 10 | **int countItems()**  Deprecated. As of JDK version 1.1, replaced by getItemCount(). |
| 11 | **void delItem(int position)**  Deprecated. replaced by remove(String) and remove(int). |
| 12 | **void delItems(int start, int end)**  Deprecated. As of JDK version 1.1, Not for public use in the future. This method is expected to be retained only as a package private method. |
| 13 | **void deselect(int index)**  Deselects the item at the specified index. |
| 14 | **AccessibleContext getAccessibleContext()**  Gets the AccessibleContext associated with this List. |
| 15 | **ActionListener[] getActionListeners()**  Returns an array of all the action listeners registered on this list. |
| 16 | **String getItem(int index)**  Gets the item associated with the specified index. |
| 17 | **int getItemCount()**  Gets the number of items in the list. |
| 18 | **ItemListener[] getItemListeners()**  Returns an array of all the item listeners registered on this list. |
| 19 | **String[] getItems()**  Gets the items in the list. |
| 20 | **Dimension getMinimumSize()**  Determines the minimum size of this scrolling list. |
| 21 | **Dimension getMinimumSize(int rows)**  Gets the minumum dimensions for a list with the specified number of rows. |
| 22 | **Dimension getPreferredSize()**  Gets the preferred size of this scrolling list. |
| 23 | **Dimension getPreferredSize(int rows)**  Gets the preferred dimensions for a list with the specified number of rows. |
| 24 | **int getRows()**  Gets the number of visible lines in this list. |
| 25 | **int getSelectedIndex()**  Gets the index of the selected item on the list, |
| 26 | **int[] getSelectedIndexes()**  Gets the selected indexes on the list. |
| 27 | **String getSelectedItem()**  Gets the selected item on this scrolling list. |
| 28 | **String[] getSelectedItems()**  Gets the selected items on this scrolling list. |
| 29 | **Object[] getSelectedObjects()**  Gets the selected items on this scrolling list in an array of Objects. |
| 30 | **int getVisibleIndex()**  Gets the index of the item that was last made visible by the method makeVisible. |
| 31 | **boolean isIndexSelected(int index)**  Determines if the specified item in this scrolling list is selected. |
| 32 | **boolean isMultipleMode()**  Determines whether this list allows multiple selections. |
| 33 | **boolean isSelected(int index)**  Deprecated. As of JDK version 1.1, replaced by isIndexSelected(int). |
| 34 | **void makeVisible(int index)**  Makes the item at the specified index visible. |
| 35 | **Dimension minimumSize()**  Deprecated. As of JDK version 1.1, replaced by getMinimumSize(). |
| 36 | **Dimension minimumSize(int rows)**  Deprecated. As of JDK version 1.1, replaced by getMinimumSize(int). |
| 37 | **protected String paramString()**  Returns the parameter string representing the state of this scrolling list. |
| 38 | **Dimension preferredSize()**  Deprecated. As of JDK version 1.1, replaced by getPreferredSize(). |
| 39 | **Dimension preferredSize(int rows)**  Deprecated. As of JDK version 1.1, replaced by getPreferredSize(int). |
| 40 | **protected void processActionEvent(ActionEvent e)**  Processes action events occurring on this component by dispatching them to any registered ActionListener objects. |
| 41 | **protected void processEvent(AWTEvent e)**  Processes events on this scrolling list. |
| 42 | **protected void processItemEvent(ItemEvent e)**  Processes item events occurring on this list by dispatching them to any registered ItemListener objects. |
| 43 | **void remove(int position)**  Removes the item at the specified position from this scrolling list. |
| 44 | **void remove(String item)**  Removes the first occurrence of an item from the list. |
| 45 | **void removeActionListener(ActionListener l)**  Removes the specified action listener so that it no longer receives action events from this list. |
| 46 | **void removeAll()**  Removes all items from this list. |
| 47 | **void removeItemListener(ItemListener l)**  Removes the specified item listener so that it no longer receives item events from this list. |
| 48 | **void removeNotify()**  Removes the peer for this list. |
| 49 | **void replaceItem(String newValue, int index)**  Replaces the item at the specified index in the scrolling list with the new string. |
| 50 | **void select(int index)**  Selects the item at the specified index in the scrolling list. |
| 51 | **void setMultipleMode(boolean b)**  Sets the flag that determines whether this list allows multiple selections. |
| 52 | **void setMultipleSelections(boolean b)**  Deprecated. As of JDK version 1.1, replaced by setMultipleMode(boolean). |

## Methods inherited

This class inherits methods from the following classes:

* java.awt.Component
* java.lang.Object

**AwtControlDemo.java**

import java.awt.\*;

import java.awt.event.\*;

public class AwtControlDemo {

private Frame mainFrame;

private Label headerLabel;

private Label statusLabel;

private Panel controlPanel;

public AwtControlDemo(){

prepareGUI();

}

public static void main(String[] args){

AwtControlDemo awtControlDemo = new AwtControlDemo();

awtControlDemo.showListDemo();

}

private void prepareGUI(){

mainFrame = new Frame("Java AWT Examples");

mainFrame.setSize(400,400);

mainFrame.setLayout(new GridLayout(3, 1));

mainFrame.addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent windowEvent){

System.exit(0);

}

});

headerLabel = new Label();

headerLabel.setAlignment(Label.CENTER);

statusLabel = new Label();

statusLabel.setAlignment(Label.CENTER);

statusLabel.setSize(350,100);

controlPanel = new Panel();

controlPanel.setLayout(new FlowLayout());

mainFrame.add(headerLabel);

mainFrame.add(controlPanel);

mainFrame.add(statusLabel);

mainFrame.setVisible(true);

}

private void showListDemo(){

headerLabel.setText("Control in action: List");

final List fruitList = new List(4,false);

fruitList.add("Apple");

fruitList.add("Grapes");

fruitList.add("Mango");

fruitList.add("Peer");

final List vegetableList = new List(4,true);

vegetableList.add("Lady Finger");

vegetableList.add("Onion");

vegetableList.add("Potato");

vegetableList.add("Tomato");

Button showButton = new Button("Show");

showButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

String data = "Fruits Selected: "

+ fruitList.getItem(fruitList.getSelectedIndex());

data += ", Vegetables selected: ";

for(String vegetable:vegetableList.getSelectedItems()){

data += vegetable + " ";

}

statusLabel.setText(data);

}

});

controlPanel.add(fruitList);

controlPanel.add(vegetableList);

controlPanel.add(showButton);

mainFrame.setVisible(true);

}

}

# AWT Image Class

## Introduction

Image control is superclass for all image classes representing graphical images.

## Class declaration

Following is the declaration for **java.awt.Image** class:

public abstract class Image

extends Object

## Field

Following are the fields for **java.awt.Image** class:

* **protected float accelerationPriority**-- Priority for accelerating this image.
* **static int SCALE\_AREA\_AVERAGING**-- Use the Area Averaging image scaling algorithm.
* **static int SCALE\_DEFAULT**-- Use the default image-scaling algorithm.
* **static int SCALE\_FAST**-- Choose an image-scaling algorithm that gives higher priority to scaling speed than smoothness of the scaled image.
* **static int SCALE\_REPLICATE**-- Use the image scaling algorithm embodied in the ReplicateScaleFilter class.
* **static int SCALE\_SMOOTH**-- Choose an image-scaling algorithm that gives higher priority to image smoothness than scaling speed.
* **static Object UndefinedProperty**-- The UndefinedProperty object should be returned whenever a property which was not defined for a particular image is fetched.

## Class constructors

|  |  |
| --- | --- |
| **S.N.** | **Constructor & Description** |
| 1 | **Image()** |

## Class methods

|  |  |
| --- | --- |
| **S.N.** | **Method & Description** |
| 1 | **void flush()**  Flushes all reconstructable resources being used by this Image object. |
| 2 | **float getAccelerationPriority()**  Returns the current value of the acceleration priority hint. |
| 3 | **ImageCapabilities getCapabilities(GraphicsConfiguration gc)**  Returns an ImageCapabilities object which can be inquired as to the capabilities of this Image on the specified GraphicsConfiguration. |
| 4 | **abstract Graphics getGraphics()**  Creates a graphics context for drawing to an off-screen image. |
| 5 | **abstract int getHeight(ImageObserver observer)**  Determines the height of the image. |
| 6 | **abstract Object getProperty(String name, ImageObserver observer)**  Gets a property of this image by name. |
| 7 | **Image getScaledInstance(int width, int height, int hints)**  Creates a scaled version of this image. |
| 8 | **abstract ImageProducer getSource()**  Gets the object that produces the pixels for the image. |
| 9 | **abstract int getWidth(ImageObserver observer)**  Determines the width of the image. |
| 10 | **void setAccelerationPriority(float priority)**  Sets a hint for this image about how important acceleration is. |

## Methods inherited

This class inherits methods from the following classes:

* java.lang.Object

**AwtControlDemo.java**

import java.awt.\*;

import java.awt.event.\*;

public class AwtControlDemo {

private Frame mainFrame;

private Label headerLabel;

private Label statusLabel;

private Panel controlPanel;

public AwtControlDemo(){

prepareGUI();

}

public static void main(String[] args){

AwtControlDemo awtControlDemo = new AwtControlDemo();

awtControlDemo.showImageDemo();

}

private void prepareGUI(){

mainFrame = new Frame("Java AWT Examples");

mainFrame.setSize(400,400);

mainFrame.setLayout(new GridLayout(3, 1));

mainFrame.addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent windowEvent){

System.exit(0);

}

});

headerLabel = new Label();

headerLabel.setAlignment(Label.CENTER);

statusLabel = new Label();

statusLabel.setAlignment(Label.CENTER);

statusLabel.setSize(350,100);

controlPanel = new Panel();

controlPanel.setLayout(new FlowLayout());

mainFrame.add(headerLabel);

mainFrame.add(controlPanel);

mainFrame.add(statusLabel);

mainFrame.setVisible(true);

}

private void showImageDemo(){

headerLabel.setText("Control in action: Image");

controlPanel.add(new ImageComponent("resources/java.jpg"));

mainFrame.setVisible(true);

}

class ImageComponent extends Component {

BufferedImage img;

public void paint(Graphics g) {

g.drawImage(img, 0, 0, null);

}

public ImageComponent(String path) {

try {

img = ImageIO.read(new File(path));

} catch (IOException e) {

e.printStackTrace();

}

}

public Dimension getPreferredSize() {

if (img == null) {

return new Dimension(100,100);

} else {

return new Dimension(img.getWidth(), img.getHeight());

}

}

}

}

# AWT Dialog Class

## Introduction

Dialog control represents a top-level window with a title and a border used to take some form of input from the user.

## Class declaration

Following is the declaration for **java.awt.Dialog** class:

public class Dialog

extends Window

## Field

Following are the fields for **java.awt.Image** class:

* **static Dialog.ModalityType DEFAULT\_MODALITY\_TYPE**-- Default modality type for modal dialogs.

## Class constructors

|  |  |
| --- | --- |
| **S.N.** | **Constructor & Description** |
| 1 | **Dialog(Dialog owner)**  Constructs an initially invisible, modeless Dialog with the specified owner Dialog and an empty title. |
| 2 | **Dialog(Dialog owner, String title)**  Constructs an initially invisible, modeless Dialog with the specified owner Dialog and title. |
| 3 | **Dialog(Dialog owner, String title, boolean modal)**  Constructs an initially invisible Dialog with the specified owner Dialog, title, and modality. |
| 4 | **Dialog(Dialog owner, String title, boolean modal, GraphicsConfiguration gc)**  Constructs an initially invisible Dialog with the specified owner Dialog, title, modality and GraphicsConfiguration. |
| 5 | **Dialog(Frame owner)**  Constructs an initially invisible, modeless Dialog with the specified owner Frame and an empty title. |
| 6 | **Dialog(Frame owner, boolean modal)**  Constructs an initially invisible Dialog with the specified owner Frame and modality and an empty title. |
| 7 | **Dialog(Frame owner, String title)**  Constructs an initially invisible, modeless Dialog with the specified owner Frame and title. |
| 8 | **Dialog(Frame owner, String title, boolean modal)**  Constructs an initially invisible Dialog with the specified owner Frame, title and modality. |
| 9 | **Dialog(Frame owner, String title, boolean modal, GraphicsConfiguration gc)**  Constructs an initially invisible Dialog with the specified owner Frame, title, modality, and GraphicsConfiguration. |
| 10 | **Dialog(Window owner)**  Constructs an initially invisible, modeless Dialog with the specified owner Window and an empty title. |
| 11 | **Dialog(Window owner, Dialog.ModalityType modalityType)**  Constructs an initially invisible Dialog with the specified owner Window and modality and an empty title. |
| 12 | **Dialog(Window owner, String title)**  Constructs an initially invisible, modeless Dialog with the specified owner Window and title. |
| 13 | **Dialog(Window owner, String title, Dialog.ModalityType modalityType)**  Constructs an initially invisible Dialog with the specified owner Window, title and modality. |
| 14 | **Dialog(Window owner, String title, Dialog.ModalityType modalityType, GraphicsConfiguration gc)**  Constructs an initially invisible Dialog with the specified owner Window, title, modality and GraphicsConfiguration |

## Class methods

|  |  |
| --- | --- |
| **S.N.** | **Method & Description** |
| 1 | **void addNotify()**  Makes this Dialog displayable by connecting it to a native screen resource. |
| 2 | **AccessibleContext getAccessibleContext()**  Gets the AccessibleContext associated with this Dialog. |
| 3 | **Dialog.ModalityType getModalityType()**  Returns the modality type of this dialog. |
| 4 | **String getTitle()**  Gets the title of the dialog. |
| 5 | **void hide()**  Deprecated. As of JDK version 1.5, replaced by setVisible(boolean). |
| 6 | **boolean isModal()**  Indicates whether the dialog is modal. |
| 7 | **boolean isResizable()**  Indicates whether this dialog is resizable by the user. |
| 8 | **boolean isUndecorated()**  Indicates whether this dialog is undecorated. |
| 9 | **protected String paramString()**  Returns a string representing the state of this dialog. |
| 10 | **void setModal(boolean modal)**  Specifies whether this dialog should be modal. |
| 11 | **void setModalityType(Dialog.ModalityType type)**  Sets the modality type for this dialog. |
| 12 | **void setResizable(boolean resizable)**  Sets whether this dialog is resizable by the user. |
| 13 | **void setTitle(String title)**  Sets the title of the Dialog. |
| 14 | **void setUndecorated(boolean undecorated)**  Disables or enables decorations for this dialog. |
| 15 | **void setVisible(boolean b)**  Shows or hides this Dialog depending on the value of parameter b. |
| 16 | **void show()**  Deprecated. As of JDK version 1.5, replaced by setVisible(boolean). |
| 17 | **void toBack()**  If this Window is visible, sends this Window to the back and may cause it to lose focus or activation if it is the focused or active Window. |

## Methods inherited

This class inherits methods from the following classes:

* java.awt.Window
* java.awt.Component
* java.lang.Object

**AwtControlDemo.java**

import java.awt.\*;

import java.awt.event.\*;

public class AwtControlDemo {

private Frame mainFrame;

private Label headerLabel;

private Label statusLabel;

private Panel controlPanel;

public AwtControlDemo(){

prepareGUI();

}

public static void main(String[] args){

AwtControlDemo awtControlDemo = new AwtControlDemo();

awtControlDemo.showDialogDemo();

}

private void prepareGUI(){

mainFrame = new Frame("Java AWT Examples");

mainFrame.setSize(400,400);

mainFrame.setLayout(new GridLayout(3, 1));

mainFrame.addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent windowEvent){

System.exit(0);

}

});

headerLabel = new Label();

headerLabel.setAlignment(Label.CENTER);

statusLabel = new Label();

statusLabel.setAlignment(Label.CENTER);

statusLabel.setSize(350,100);

controlPanel = new Panel();

controlPanel.setLayout(new FlowLayout());

mainFrame.add(headerLabel);

mainFrame.add(controlPanel);

mainFrame.add(statusLabel);

mainFrame.setVisible(true);

}

private void showDialogDemo(){

headerLabel.setText("Control in action: Dialog");

Button showAboutDialogButton = new Button("Show About Dialog");

showAboutDialogButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

AboutDialog aboutDialog = new AboutDialog(mainFrame);

aboutDialog.setVisible(true);

}

});

controlPanel.add(showAboutDialogButton);

mainFrame.setVisible(true);

}

class AboutDialog extends Dialog {

public AboutDialog(Frame parent){

super(parent, true);

setBackground(Color.gray);

setLayout(new BorderLayout());

Panel panel = new Panel();

panel.add(new Button("Close"));

add("South", panel);

setSize(200,200);

addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent windowEvent){

dispose();

}

});

}

public boolean action(Event evt, Object arg){

if(arg.equals("Close")){

dispose();

return true;

}

return false;

}

public void paint(Graphics g){

g.setColor(Color.white);

g.drawString("TutorialsPoint.Com", 25,70 );

g.drawString("Version 1.0", 60, 90);

} }}

# AWT FileDialog Class

## Introduction

FileDialog control represents a dialog window from which the user can select a file.

## Class declaration

Following is the declaration for **java.awt.FileDialog** class:

public class FileDialog

extends Dialog

## Field

Following are the fields for **java.awt.Image** class:

* **static int LOAD**-- This constant value indicates that the purpose of the file dialog window is to locate a file from which to read.
* **static int SAVE**-- This constant value indicates that the purpose of the file dialog window is to locate a file to which to write.

## Class constructors

|  |  |
| --- | --- |
| **S.N.** | **Constructor & Description** |
| 1 | **FileDialog(Dialog parent)**  Creates a file dialog for loading a file. |
| 2 | **FileDialog(Dialog parent, String title)**  Creates a file dialog window with the specified title for loading a file. |
| 3 | **FileDialog(Dialog parent, String title, int mode)**  Creates a file dialog window with the specified title for loading or saving a file. |
| 4 | **FileDialog(Frame parent)**  Creates a file dialog for loading a file. |
| 5 | **FileDialog(Frame parent, String title)**  Creates a file dialog window with the specified title for loading a file. |
| 6 | **FileDialog(Frame parent, String title, int mode)**  Creates a file dialog window with the specified title for loading or saving a file. |

## Class methods

|  |  |
| --- | --- |
| **S.N.** | **Method & Description** |
| 1 | **void addNotify()**  Creates the file dialog's peer. |
| 2 | **String getDirectory()**  Gets the directory of this file dialog. |
| 3 | **String getFile()**  Gets the selected file of this file dialog. |
| 4 | **FilenameFilter getFilenameFilter()**  Determines this file dialog's filename filter. |
| 5 | **int getMode()**  Indicates whether this file dialog box is for loading from a file or for saving to a file. |
| 6 | **protected String paramString()**  Returns a string representing the state of this FileDialog window. |
| 7 | **void setDirectory(String dir)**  Sets the directory of this file dialog window to be the specified directory. |
| 8 | **void setFile(String file)**  Sets the selected file for this file dialog window to be the specified file. |
| 9 | **void setFilenameFilter(FilenameFilter filter)**  Sets the filename filter for this file dialog window to the specified filter. |
| 10 | **void setMode(int mode)**  Sets the mode of the file dialog. |

## Methods inherited

This class inherits methods from the following classes:

* java.awt.Dialog
* java.awt.Window
* java.awt.Component
* java.lang.Object

## FileDialog Example

***AwtControlDemo.java***

import java.awt.\*;

import java.awt.event.\*;

public class AwtControlDemo {

private Frame mainFrame;

private Label headerLabel;

private Label statusLabel;

private Panel controlPanel;

public AwtControlDemo(){

prepareGUI();

}

public static void main(String[] args){

AwtControlDemo awtControlDemo = new AwtControlDemo();

awtControlDemo.showFileDialogDemo();

}

private void prepareGUI(){

mainFrame = new Frame("Java AWT Examples");

mainFrame.setSize(400,400);

mainFrame.setLayout(new GridLayout(3, 1));

mainFrame.addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent windowEvent){

System.exit(0);

}

});

headerLabel = new Label();

headerLabel.setAlignment(Label.CENTER);

statusLabel = new Label();

statusLabel.setAlignment(Label.CENTER);

statusLabel.setSize(350,100);

controlPanel = new Panel();

controlPanel.setLayout(new FlowLayout());

mainFrame.add(headerLabel);

mainFrame.add(controlPanel);

mainFrame.add(statusLabel);

mainFrame.setVisible(true);

}

private void showFileDialogDemo(){

headerLabel.setText("Control in action: FileDialog");

final FileDialog fileDialog = new FileDialog(mainFrame,"Select file");

Button showFileDialogButton = new Button("Open File");

showFileDialogButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

fileDialog.setVisible(true);

statusLabel.setText("File Selected :"

+ fileDialog.getDirectory() + fileDialog.getFile());

}

});

controlPanel.add(showFileDialogButton);

mainFrame.setVisible(true);

}

}

**Java Swing tutorial** is a part of 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.

Unlike AWT, Java Swing provides platform-independent and lightweight components.

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

Difference between AWT and Swing

There are many differences between java awt and swing that are given below.

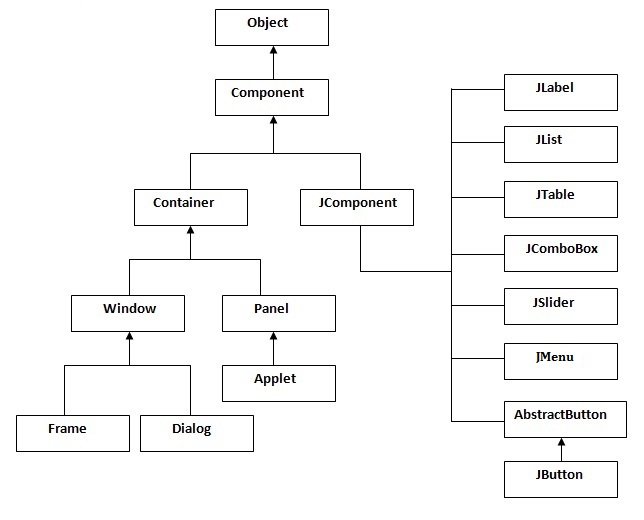
|  |  |  |
| --- | --- | --- |
| **No.** | **Java AWT** | **Java Swing** |
| 1) | AWT components are **platform-dependent**. | Java swing components are **platform-independent**. |
| 2) | AWT components are **heavyweight**. | Swing components are **lightweight**. |
| 3) | AWT **doesn't support pluggable look and feel**. | Swing **supports pluggable look and feel**. |
| 4) | AWT provides **less components** than Swing. | Swing provides **more powerful components**such as tables, lists, scrollpanes, colorchooser, tabbedpane etc. |
| 5) | AWT **doesn't follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view. | Swing **follows MVC**. |

What is JFC

The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.

Hierarchy of Java Swing classes

The hierarchy of java swing API is given below.



Commonly used Methods of Component class

The methods of Component class are widely used in java swing that are given below.

|  |  |
| --- | --- |
| **Method** | **Description** |
| public void add(Component c) | add a component on another component. |
| public void setSize(int width,int height) | sets size of the component. |
| public void setLayout(LayoutManager m) | sets the layout manager for the component. |
| public void setVisible(boolean b) | sets the visibility of the component. It is by default false. |

Java Swing Examples

There are two ways to create a frame:

* By creating the object of Frame class (association)
* By extending Frame class (inheritance)

We can write the code of swing inside the main(), constructor or any other method.

Simple Java Swing Example

Let's see a simple swing example where we are creating one button and adding it on the JFrame object inside the main() method.

*File: FirstSwingExample.java*

**import** javax.swing.\*;

**public** **class** FirstSwingExample {

**public** **static** **void** main(String[] args) {

JFrame f=**new** JFrame();//creating instance of JFrame

JButton b=**new** JButton("click");//creating instance of JButton

b.setBounds(130,100,100, 40);//x axis, y axis, width, height

f.add(b);//adding button in JFrame

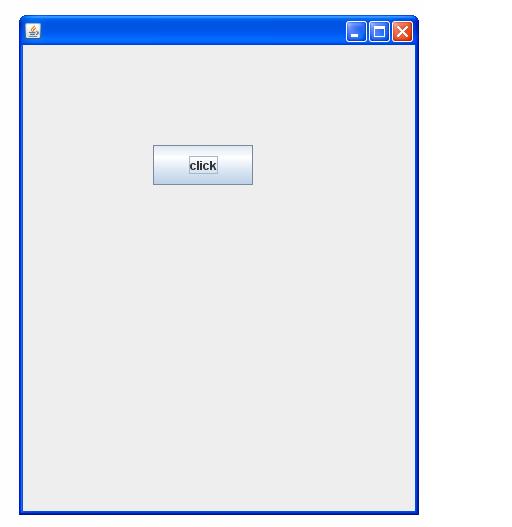
f.setSize(400,500);//400 width and 500 height

f.setLayout(**null**);//using no layout managers

f.setVisible(**true**);//making the frame visible

}

}



Example of Swing by Association inside constructor

We can also write all the codes of creating JFrame, JButton and method call inside the java constructor.

*File: Simple.java*

**import** javax.swing.\*;

**public** **class** Simple {

JFrame f;

Simple(){

f=**new** JFrame();//creating instance of JFrame

JButton b=**new** JButton("click");//creating instance of JButton

b.setBounds(130,100,100, 40);

f.add(b);//adding button in JFrame

f.setSize(400,500);//400 width and 500 height

f.setLayout(**null**);//using no layout managers

f.setVisible(**true**);//making the frame visible

}

**public** **static** **void** main(String[] args) {

**new** Simple();

}

}

The setBounds(int xaxis, int yaxis, int width, int height)is used in the above example that sets the position of the button.

Simple example of Swing by inheritance

We can also inherit the JFrame class, so there is no need to create the instance of JFrame class explicitly.

*File: Simple2.java*

**import** javax.swing.\*;

**public** **class** Simple2 **extends** JFrame{//inheriting JFrame

JFrame f;

Simple2(){

JButton b=**new** JButton("click");//create button

b.setBounds(130,100,100, 40);

add(b);//adding button on frame

setSize(400,500);

setLayout(**null**);

setVisible(**true**);

}

**public** **static** **void** main(String[] args) {

**new** Simple2();

|  |
| --- |
|  |

### Commonly used Constructors:

* **JButton():** creates a button with no text and icon.
* **JButton(String s):** creates a button with the specified text.
* **JButton(Icon i):** creates a button with the specified icon object.

### Commonly used Methods of AbstractButton class:

|  |
| --- |
| **1) public void setText(String s):** is used to set specified text on button. |
| **2) public String getText():** is used to return the text of the button. |
| **3) public void setEnabled(boolean b):** is used to enable or disable the button. |
| **4) public void setIcon(Icon b):** is used to set the specified Icon on the button. |
| **5) public Icon getIcon():** is used to get the Icon of the button. |
| **6) public void setMnemonic(int a):** is used to set the mnemonic on the button. |
| **7) public void addActionListener(ActionListener a):** is used to add the action listener to this object. |

#### Note: The JButton class extends AbstractButton class.

**import** java.awt.event.\*;

**import** javax.swing.\*;

**public** **class** ImageButton{

ImageButton(){

JFrame f=**new** JFrame();

JButton b=**new** JButton(**new** ImageIcon("b.jpg"));

b.setBounds(130,100,100, 40);

f.add(b);

f.setSize(300,400);

f.setLayout(**null**);

f.setVisible(**true**);

f.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

          .

    }

**public** **static** **void** main(String[] args) {

**new** ImageButton();

}}

# JRadioButton class

The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz.

It should be added in ButtonGroup to select one radio button only.

### Commonly used Constructors of JRadioButton class:

* **JRadioButton():** creates an unselected radio button with no text.
* **JRadioButton(String s):** creates an unselected radio button with specified text.
* **JRadioButton(String s, boolean selected):** creates a radio button with the specified text and selected status.

### Commonly used Methods of AbstractButton class:

|  |
| --- |
| **1) public void setText(String s):** is used to set specified text on button. |
| **2) public String getText():** is used to return the text of the button. |
| **3) public void setEnabled(boolean b):** is used to enable or disable the button. |
| **4) public void setIcon(Icon b):** is used to set the specified Icon on the button. |
| **5) public Icon getIcon():** is used to get the Icon of the button. |
| **6) public void setMnemonic(int a):** is used to set the mnemonic on the button. |
| **7) public void addActionListener(ActionListener a):** is used to add the action listener to this object. |

#### Note: The JRadioButton class extends the JToggleButton class that extends AbstractButton class.

### Example of JRadioButton class:

**import** javax.swing.\*;

**public** **class** Radio {

JFrame f;

Radio(){

f=**new** JFrame();

JRadioButton r1=**new** JRadioButton("A) Male");

JRadioButton r2=**new** JRadioButton("B) FeMale");

r1.setBounds(50,100,70,30);

r2.setBounds(50,150,70,30);

ButtonGroup bg=**new** ButtonGroup();

bg.add(r1);bg.add(r2);

f.add(r1);f.add(r2);

f.setSize(300,300);

f.setLayout(**null**);

f.setVisible(**true**);

}

**public** **static** **void** main(String[] args) {

**new** Radio();

}

}

### ButtonGroup class:

|  |
| --- |
| The ButtonGroup class can be used to group multiple buttons so that at a time only one button can be selected. |

### JRadioButton example with event handling

**import** javax.swing.\*;

**import** java.awt.event.\*;

**class** RadioExample **extends** JFrame **implements** ActionListener{

JRadioButton rb1,rb2;

JButton b;

RadioExample(){

rb1=**new** JRadioButton("Male");

rb1.setBounds(100,50,100,30);

rb2=**new** JRadioButton("Female");

rb2.setBounds(100,100,100,30);

ButtonGroup bg=**new** ButtonGroup();

bg.add(rb1);bg.add(rb2);

b=**new** JButton("click");

b.setBounds(100,150,80,30);

b.addActionListener(**this**);

add(rb1);add(rb2);add(b);

setSize(300,300);

setLayout(**null**);

setVisible(**true**);

}

**public** **void** actionPerformed(ActionEvent e){

**if**(rb1.isSelected()){

JOptionPane.showMessageDialog(**this**,"You are male");

}

**if**(rb2.isSelected()){

JOptionPane.showMessageDialog(**this**,"You are female");

}

}

**public** **static** **void** main(String args[]){

**new** RadioExample();

}}

**JComboBox class:**

The JComboBox class is used to create the combobox (drop-down list). At a time only one item can be selected from the item list.

Commonly used Constructors of JComboBox class:

|  |
| --- |
| JComboBox() |
| JComboBox(Object[] items) |
| JComboBox(Vector<?> items) |

Commonly used methods of JComboBox class:

|  |
| --- |
| **1) public void addItem(Object anObject):** is used to add an item to the item list. |
| **2) public void removeItem(Object anObject):** is used to delete an item to the item list. |
| **3) public void removeAllItems():** is used to remove all the items from the list. |
| **4) public void setEditable(boolean b):** is used to determine whether the JComboBox is editable. |
| **5) public void addActionListener(ActionListener a):** is used to add the ActionListener. |
| **6) public void addItemListener(ItemListener i):** is used to add the ItemListener. |

Example of JComboBox class:

**import** javax.swing.\*;

**public** **class** Combo {

JFrame f;

Combo(){

    f=**new** JFrame("Combo ex");

    String country[]={"India","Aus","U.S.A","England","Newzeland"};

    JComboBox cb=**new** JComboBox(country);

    cb.setBounds(50, 50,90,20);

    f.add(cb);

    f.setLayout(**null**);

    f.setSize(400,500);

    f.setVisible(**true**);

}

**public** **static** **void** main(String[] args) {

**new** Combo();

}

}

**JTable class (Swing Tutorial):**

|  |
| --- |
| The JTable class is used to display the data on two dimensional tables of cells. |

Commonly used Constructors of JTable class:

* **JTable():** creates a table with empty cells.
* **JTable(Object[][] rows, Object[] columns):** creates a table with the specified data.

Example of JTable class:

**import** javax.swing.\*;

**public** **class** MyTable {

    JFrame f;

MyTable(){

    f=**new** JFrame();

    String data[][]={ {"101","Amit","670000"},

              {"102","Jai","780000"},

                          {"101","Sachin","700000"}};

    String column[]={"ID","NAME","SALARY"};

    JTable jt=**new** JTable(data,column);

    jt.setBounds(30,40,200,300);

    JScrollPane sp=**new** JScrollPane(jt);

    f.add(sp);

    f.setSize(300,400);

//  f.setLayout(null);

    f.setVisible(**true**);

}

**public** **static** **void** main(String[] args) {

**new** MyTable();

}

}

**JColorChooser class:**

|  |
| --- |
| The JColorChooser class is used to create a color chooser dialog box so that user can select any color. |

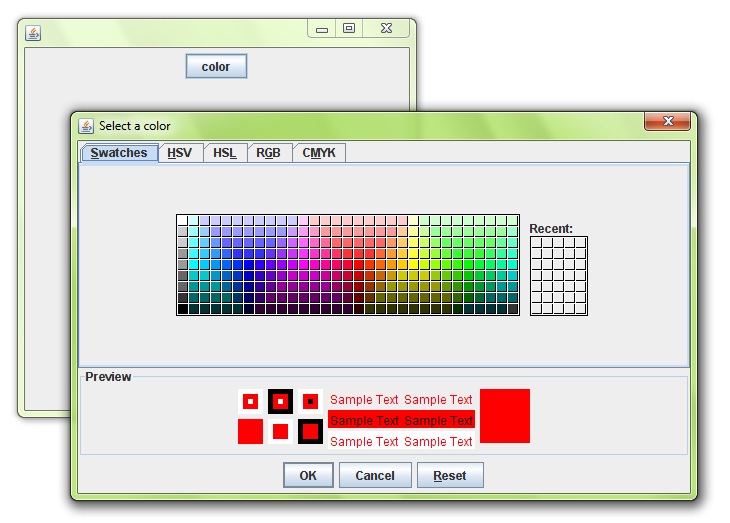
Commonly used Constructors of JColorChooser class:

|  |
| --- |
| * **JColorChooser():** is used to create a color chooser pane with white color initially. * **JColorChooser(Color initialColor):** is used to create a color chooser pane with the specified color initially. |

Commonly used methods of JColorChooser class:

|  |
| --- |
| **public static Color showDialog(Component c, String title, Color initialColor):** is used to show the color-chooser dialog box. |

Example of JColorChooser class:



**import** java.awt.event.\*;

**import** java.awt.\*;

**import** javax.swing.\*;

**public** **class** JColorChooserExample **extends** JFrame **implements** ActionListener{

JButton b;

Container c;

JColorChooserExample(){

    c=getContentPane();

    c.setLayout(**new** FlowLayout());

    b=**new** JButton("color");

    b.addActionListener(**this**);

    c.add(b);

}

**public** **void** actionPerformed(ActionEvent e) {

Color initialcolor=Color.RED;

Color color=JColorChooser.showDialog(**this**,"Select a color",initialcolor);

c.setBackground(color);

}

**public** **static** **void** main(String[] args) {

    JColorChooserExample ch=**new** JColorChooserExample();

    ch.setSize(400,400);

    ch.setVisible(**true**);

    ch.setDefaultCloseOperation(EXIT\_ON\_CLOSE);

}

}

**JProgressBar class:**

The JProgressBar class is used to display the progress of the task.

Commonly used Constructors of JProgressBar class:

* **JProgressBar():** is used to create a horizontal progress bar but no string text.
* **JProgressBar(int min, int max):** is used to create a horizontal progress bar with the specified minimum and maximum value.
* **JProgressBar(int orient):** is used to create a progress bar with the specified orientation, it can be either Vertical or Horizontal by using SwingConstants.VERTICAL and SwingConstants.HORIZONTAL constants.
* **JProgressBar(int orient, int min, int max):** is used to create a progress bar with the specified orientation, minimum and maximum value.

Commonly used methods of JProgressBar class:

|  |
| --- |
| **1) public void setStringPainted(boolean b):** is used to determine whether string should be displayed. |
| **2) public void setString(String s):** is used to set value to the progress string. |
| **3) public void setOrientation(int orientation):** is used to set the orientation, it may be either vertical or horizontal by using SwingConstants.VERTICAL and SwingConstants.HORIZONTAL constants.. |
| **4) public void setValue(int value):** is used to set the current value on the progress bar. |

Example of JProgressBar class:

**import** javax.swing.\*;

**public** **class** MyProgress **extends** JFrame{

JProgressBar jb;

**int** i=0,num=0;

MyProgress(){

jb=**new** JProgressBar(0,2000);

jb.setBounds(40,40,200,30);

jb.setValue(0);

jb.setStringPainted(**true**);

add(jb);

setSize(400,400);

setLayout(**null**);

}

**public** **void** iterate(){

**while**(i<=2000){

  jb.setValue(i);

  i=i+20;

**try**{Thread.sleep(150);}**catch**(Exception e){}

}

}

**public** **static** **void** main(String[] args) {

    MyProgress m=**new** MyProgress();

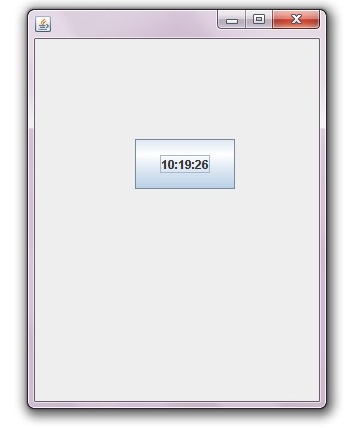
    m.setVisible(**true**);

    m.iterate();

}

}

Example of digital clock in swing:



**import** javax.swing.\*;

**import** java.awt.\*;

**import** java.text.\*;

**import** java.util.\*;

**public** **class** DigitalWatch **implements** Runnable{

JFrame f;

Thread t=**null**;

**int** hours=0, minutes=0, seconds=0;

String timeString = "";

JButton b;

DigitalWatch(){

    f=**new** JFrame();

    t = **new** Thread(**this**);

        t.start();

    b=**new** JButton();

        b.setBounds(100,100,100,50);

    f.add(b);

    f.setSize(300,400);

    f.setLayout(**null**);

    f.setVisible(**true**);

}

**public** **void** run() {

**try** {

**while** (**true**) {

            Calendar cal = Calendar.getInstance();

            hours = cal.get( Calendar.HOUR\_OF\_DAY );

**if** ( hours > 12 ) hours -= 12;

            minutes = cal.get( Calendar.MINUTE );

            seconds = cal.get( Calendar.SECOND );

            SimpleDateFormat formatter = **new** SimpleDateFormat("hh:mm:ss");

            Date date = cal.getTime();

            timeString = formatter.format( date );

            printTime();

            t.sleep( 1000 );  // interval given in milliseconds

         }

      }

**catch** (Exception e) { }

 }

**public** **void** printTime(){

b.setText(timeString);

}

**public** **static** **void** main(String[] args) {

**new** DigitalWatch();

}

}

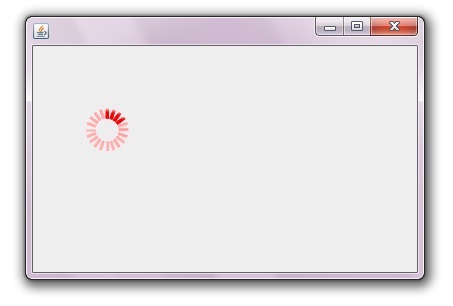
**Displaying image in swing:**

|  |
| --- |
| For displaying image, we can use the method drawImage() of Graphics class. |

Syntax of drawImage() method:

|  |
| --- |
| 1. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image. |

Example of displaying image in swing:



**import** java.awt.\*;

**import** javax.swing.JFrame;

**public** **class** MyCanvas **extends** Canvas{

**public** **void** paint(Graphics g) {

        Toolkit t=Toolkit.getDefaultToolkit();

        Image i=t.getImage("p3.gif");

        g.drawImage(i, 120,100,**this**);

    }

**public** **static** **void** main(String[] args) {

        MyCanvas m=**new** MyCanvas();

        JFrame f=**new** JFrame();

        f.add(m);

        f.setSize(400,400);

        f.setVisible(**true**);

    }

}

**Example of creating Edit menu for Notepad:**

**import** javax.swing.\*;

**import** java.awt.event.\*;

**public** **class** Notepad **implements** ActionListener{

JFrame f;

JMenuBar mb;

JMenu file,edit,help;

JMenuItem cut,copy,paste,selectAll;

JTextArea ta;

Notepad(){

f=**new** JFrame();

cut=**new** JMenuItem("cut");

copy=**new** JMenuItem("copy");

paste=**new** JMenuItem("paste");

selectAll=**new** JMenuItem("selectAll");

cut.addActionListener(**this**);

copy.addActionListener(**this**);

paste.addActionListener(**this**);

selectAll.addActionListener(**this**);

mb=**new** JMenuBar();

mb.setBounds(5,5,400,40);

file=**new** JMenu("File");

edit=**new** JMenu("Edit");

help=**new** JMenu("Help");

edit.add(cut);edit.add(copy);edit.add(paste);edit.add(selectAll);

mb.add(file);mb.add(edit);mb.add(help);

ta=**new** JTextArea();

ta.setBounds(5,30,460,460);

f.add(mb);f.add(ta);

f.setLayout(**null**);

f.setSize(500,500);

f.setVisible(**true**);

}

**public** **void** actionPerformed(ActionEvent e) {

**if**(e.getSource()==cut)

ta.cut();

**if**(e.getSource()==paste)

ta.paste();

**if**(e.getSource()==copy)

ta.copy();

**if**(e.getSource()==selectAll)

ta.selectAll();

}

**public** **static** **void** main(String[] args) {

**new** Notepad();

}

}

**Example of open dialog box:**

**import** java.awt.\*;

**import** javax.swing.\*;

**import** java.awt.event.\*;

**import** java.io.\*;

**public** **class** OpenMenu **extends** JFrame **implements** ActionListener{

JMenuBar mb;

JMenu file;

JMenuItem open;

JTextArea ta;

OpenMenu(){

open=**new** JMenuItem("Open File");

open.addActionListener(**this**);

file=**new** JMenu("File");

file.add(open);

mb=**new** JMenuBar();

mb.setBounds(0,0,800,20);

mb.add(file);

ta=**new** JTextArea(800,800);

ta.setBounds(0,20,800,800);

add(mb);

add(ta);

}

**public** **void** actionPerformed(ActionEvent e) {

**if**(e.getSource()==open){

openFile();

}

}

**void** openFile(){

JFileChooser fc=**new** JFileChooser();

**int** i=fc.showOpenDialog(**this**);

**if**(i==JFileChooser.APPROVE\_OPTION){

File f=fc.getSelectedFile();

String filepath=f.getPath();

displayContent(filepath);

}

}

**void** displayContent(String fpath){

**try**{

BufferedReader br=**new** BufferedReader(**new** FileReader(fpath));

String s1="",s2="";

**while**((s1=br.readLine())!=**null**){

s2+=s1+"\n";

}

ta.setText(s2);

br.close();

}**catch** (Exception e) {e.printStackTrace();  }

}

**public** **static** **void** main(String[] args) {

    OpenMenu om=**new** OpenMenu();

    om.setSize(800,800);

    om.setLayout(**null**);

    om.setVisible(**true**);

    om.setDefaultCloseOperation(EXIT\_ON\_CLOSE);

}

}

**BorderLayout (LayoutManagers):**

**LayoutManagers:**

The LayoutManagers are used to arrange components in a particular manner. LayoutManager is an interface that is implemented by all the classes of layout managers. There are following classes that represents the layout managers:

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout
6. javax.swing.BoxLayout
7. javax.swing.GroupLayout
8. javax.swing.ScrollPaneLayout
9. javax.swing.SpringLayout etc.

**BorderLayout:**

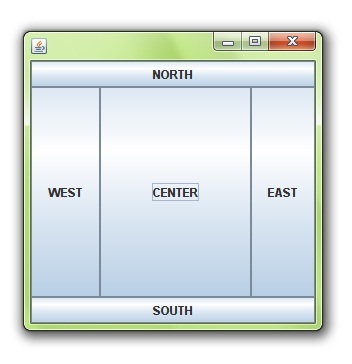
The BorderLayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. The BorderLayout provides five constants for each region:

1. **public static final int NORTH**
2. **public static final int SOUTH**
3. **public static final int EAST**
4. **public static final int WEST**
5. **public static final int CENTER**

Constructors of BorderLayout class:

* **BorderLayout():** creates a border layout but with no gaps between the components.
* **JBorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.

Example of BorderLayout class:



**import** java.awt.\*;

**import** javax.swing.\*;

**public** **class** Border {

JFrame f;

Border(){

    f=**new** JFrame();

    JButton b1=**new** JButton("NORTH");;

    JButton b2=**new** JButton("SOUTH");;

    JButton b3=**new** JButton("EAST");;

    JButton b4=**new** JButton("WEST");;

    JButton b5=**new** JButton("CENTER");;

    f.add(b1,BorderLayout.NORTH);

    f.add(b2,BorderLayout.SOUTH);

    f.add(b3,BorderLayout.EAST);

    f.add(b4,BorderLayout.WEST);

    f.add(b5,BorderLayout.CENTER);

    f.setSize(300,300);

    f.setVisible(**true**);

}

**public** **static** **void** main(String[] args) {

**new** Border();

}

}

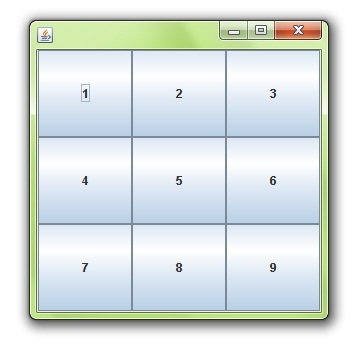
**GridLayout**

|  |
| --- |
| The GridLayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle. |

Constructors of GridLayout class:

|  |
| --- |
| 1. **GridLayout():** creates a grid layout with one column per component in a row. 2. **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components. 3. **GridLayout(int rows, int columns, int hgap, int vgap):** creates a grid layout with the given rows and columns alongwith given horizontal and vertical gaps. |

Example of GridLayout class:



**import** java.awt.\*;

**import** javax.swing.\*;

**public** **class** MyGridLayout{

JFrame f;

MyGridLayout(){

    f=**new** JFrame();

    JButton b1=**new** JButton("1");

    JButton b2=**new** JButton("2");

    JButton b3=**new** JButton("3");

    JButton b4=**new** JButton("4");

    JButton b5=**new** JButton("5");

        JButton b6=**new** JButton("6");

        JButton b7=**new** JButton("7");

    JButton b8=**new** JButton("8");

        JButton b9=**new** JButton("9");

    f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);

    f.add(b6);f.add(b7);f.add(b8);f.add(b9);

    f.setLayout(**new** GridLayout(3,3));

    //setting grid layout of 3 rows and 3 columns

    f.setSize(300,300);

    f.setVisible(**true**);

}

**public** **static** **void** main(String[] args) {

**new** MyGridLayout();

}

}

**FlowLayout**

|  |
| --- |
| The FlowLayout is used to arrange the components in a line, one after another (in a flow). It is the default layout of applet or panel. |

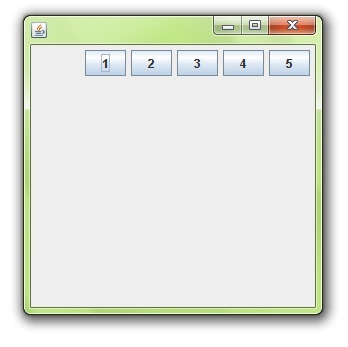
Fields of FlowLayout class:

|  |
| --- |
| 1. **public static final int LEFT** 2. **public static final int RIGHT** 3. **public static final int CENTER** 4. **public static final int LEADING** 5. **public static final int TRAILING** |

Constructors of FlowLayout class:

|  |
| --- |
| 1. **FlowLayout():** creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap. 2. **FlowLayout(int align):** creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap. 3. **FlowLayout(int align, int hgap, int vgap):** creates a flow layout with the given alignment and the given horizontal and vertical gap. |

Example of FlowLayout class:



**import** java.awt.\*;

**import** javax.swing.\*;

**public** **class** MyFlowLayout{

JFrame f;

MyFlowLayout(){

    f=**new** JFrame();

    JButton b1=**new** JButton("1");

    JButton b2=**new** JButton("2");

    JButton b3=**new** JButton("3");

    JButton b4=**new** JButton("4");

    JButton b5=**new** JButton("5");

    f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);

    f.setLayout(**new** FlowLayout(FlowLayout.RIGHT));

    //setting flow layout of right alignment

    f.setSize(300,300);

    f.setVisible(**true**);

}

**public** **static** **void** main(String[] args) {

**new** MyFlowLayout();

}

}

# BoxLayout class:

|  |
| --- |
| The BoxLayout is used to arrange the components either vertically or horizontally. For this purpose, BoxLayout provides four constants. They are as follows: |

#### Note: BoxLayout class is found in javax.swing package.

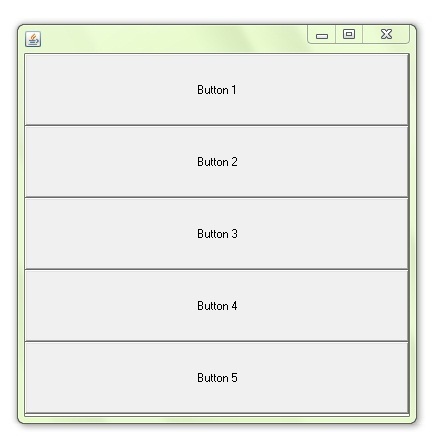
### Fields of BoxLayout class:

|  |
| --- |
| 1. **public static final int X\_AXIS** 2. **public static final int Y\_AXIS** 3. **public static final int LINE\_AXIS** 4. **public static final int PAGE\_AXIS** |

### Constructor of BoxLayout class:

|  |
| --- |
| 1. **BoxLayout(Container c, int axis):** creates a box layout that arranges the components with the given axis. |

### Example of BoxLayout class with Y-AXIS:



**import** java.awt.\*;

**import** javax.swing.\*;

**public** **class** BoxLayoutExample1 **extends** Frame {

 Button buttons[];

**public** BoxLayoutExample1 () {

   buttons = **new** Button [5];

**for** (**int** i = 0;i<5;i++) {

      buttons[i] = **new** Button ("Button " + (i + 1));

      add (buttons[i]);

    }

setLayout (**new** BoxLayout (**this**, BoxLayout.Y\_AXIS));

setSize(400,400);

setVisible(**true**);

}

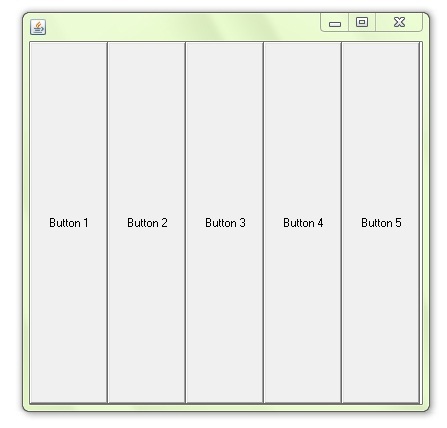
**public** **static** **void** main(String args[]){

BoxLayoutExample1 b=**new** BoxLayoutExample1();

}

}

### Example of BoxLayout class with X-AXIS:



**import** java.awt.\*;

**import** javax.swing.\*;

**public** **class** BoxLayoutExample2 **extends** Frame {

 Button buttons[];

**public** BoxLayoutExample2() {

   buttons = **new** Button [5];

**for** (**int** i = 0;i<5;i++) {

      buttons[i] = **new** Button ("Button " + (i + 1));

      add (buttons[i]);

    }

setLayout (**new** BoxLayout(**this**, BoxLayout.X\_AXIS));

setSize(400,400);

setVisible(**true**);

}

**public** **static** **void** main(String args[]){

BoxLayoutExample2 b=**new** BoxLayoutExample2();

}

}

**CardLayout class**

|  |
| --- |
| The CardLayout class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout. |

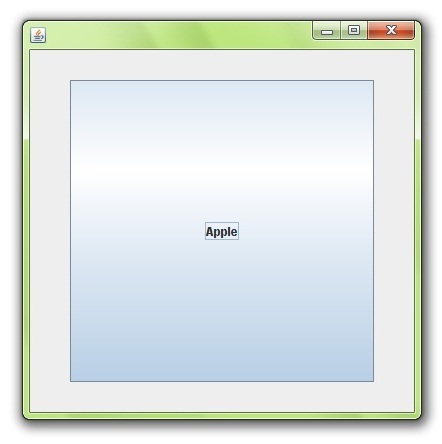
Constructors of CardLayout class:

|  |
| --- |
| 1. **CardLayout():** creates a card layout with zero horizontal and vertical gap. 2. **CardLayout(int hgap, int vgap):** creates a card layout with the given horizontal and vertical gap. |

Commonly used methods of CardLayout class:

|  |
| --- |
| * **public void next(Container parent):** is used to flip to the next card of the given container. * **public void previous(Container parent):** is used to flip to the previous card of the given container. * **public void first(Container parent):** is used to flip to the first card of the given container. * **public void last(Container parent):** is used to flip to the last card of the given container. * **public void show(Container parent, String name):** is used to flip to the specified card with the given name. |

Example of CardLayout class:



**import** java.awt.\*;

**import** java.awt.event.\*;

**import** javax.swing.\*;

**public** **class** CardLayoutExample **extends** JFrame **implements** ActionListener{

CardLayout card;

JButton b1,b2,b3;

Container c;

    CardLayoutExample(){

        c=getContentPane();

        card=**new** CardLayout(40,30);

//create CardLayout object with 40 hor space and 30 ver space

        c.setLayout(card);

        b1=**new** JButton("Apple");

        b2=**new** JButton("Boy");

        b3=**new** JButton("Cat");

        b1.addActionListener(**this**);

        b2.addActionListener(**this**);

        b3.addActionListener(**this**);

        c.add("a",b1);c.add("b",b2);c.add("c",b3);

    }

**public** **void** actionPerformed(ActionEvent e) {

    card.next(c);

    }

**public** **static** **void** main(String[] args) {

        CardLayoutExample cl=**new** CardLayoutExample();

        cl.setSize(400,400);

        cl.setVisible(**true**);

        cl.setDefaultCloseOperation(EXIT\_ON\_CLOSE);

    }

}

# Java Applet

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

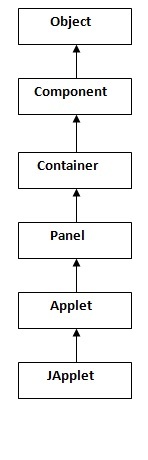
### Advantage of Applet

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

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

### Drawback of Applet

* Plugin is required at client browser to execute applet.



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

### Lifecycle of Java Applet

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

### Lifecycle methods for Applet:

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

### java.applet.Applet class

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

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

### java.awt.Component class

The Component class provides 1 life cycle method of applet.

1. **public void paint(Graphics g):** is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

### How to run an Applet?

There are two ways to run an applet

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

### Simple example of Applet by html file:

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

//First.java

**import** java.applet.Applet;

**import** java.awt.Graphics;

**public** **class** First **extends** Applet{

**public** **void** paint(Graphics g){

g.drawString("welcome",150,150);

}

}

#### Note: class must be public because its object is created by Java Plugin software that resides on the browser.

### myapplet.html

<html>

<body>

<applet code="First.class" width="300" height="300">

</applet>

</body>

</html>

### Simple example of Applet by appletviewer tool:

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

//First.java

**import** java.applet.Applet;

**import** java.awt.Graphics;

**public** **class** First **extends** Applet{

**public** **void** paint(Graphics g){

g.drawString("welcome to applet",150,150);

}

}

/\*

<applet code="First.class" width="300" height="300">

</applet>

\*/

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

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

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

**Displaying Graphics in Applet**

java.awt.Graphics class provides many methods for graphics programming.

Commonly used methods of Graphics class:

1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
5. **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
6. **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
9. **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.
10. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
11. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

**Example of Graphics in applet:**

**import** java.applet.Applet;

**import** java.awt.\*;

**public** **class** GraphicsDemo **extends** Applet{

**public** **void** paint(Graphics g){

g.setColor(Color.red);

g.drawString("Welcome",50, 50);

g.drawLine(20,30,20,300);

g.drawRect(70,100,30,30);

g.fillRect(170,100,30,30);

g.drawOval(70,200,30,30);

g.setColor(Color.pink);

g.fillOval(170,200,30,30);

g.drawArc(90,150,30,30,30,270);

g.fillArc(270,150,30,30,0,180);

}

}

myapplet.html

<html>

<body>

<applet code="GraphicsDemo.class" width="300" height="300">

</applet>

</body>

</html>

**Animation in Applet**

|  |
| --- |
| Applet is mostly used in games and animation. For this purpose image is required to be moved. |

Example of animation in applet:

**import** java.awt.\*;

**import** java.applet.\*;

**public** **class** AnimationExample **extends** Applet {

  Image picture;

**public** **void** init() {

    picture =getImage(getDocumentBase(),"bike\_1.gif");

  }

**public** **void** paint(Graphics g) {

**for**(**int** i=0;i<500;i++){

      g.drawImage(picture, i,30, **this**);

**try**{Thread.sleep(100);}**catch**(Exception e){}

    }

  }

}

|  |
| --- |
| In the above example, drawImage() method of Graphics class is used to display the image. The 4th argument of drawImage() method of is ImageObserver object. The Component class implements ImageObserver interface. So current class object would also be treated as ImageObserver because Applet class indirectly extends the Component class. |

**myapplet.html**

<html>

<body>

<applet code="DisplayImage.class" width="300" height="300">

</applet>

</body>

</html>

**END**