## UNIT – III

Java utilities like java.lang, java.util, java.io, GUI in Java using AWT and Swing, Event Handling Mechanisms, AWT based effective GUI in java: Detailed overview of AWT classes, Graphics primitives and UI Components, Layout features, Standalone GUI applications, Layout Managers, Implementation of event driven mechanism, Delegation of even model, Listeners and Adapters, Inner classes.

# UNIT – III


CHAPTER » 10

# Java Utilities–java.lang and java.util

## 10.1 java.lang Package

The java.lang package contains classes that are fundamental to the design of the Java language. The most important classes are Object which is the root of the class hierarchy, and Class, instances of which represent classes at run time. The wrapper classes Boolean, Character, Integer, Long, Float, and Double are necessary to represent a value of primitive type to Object. An object of type Double, for example, contains a field whose type is double, representing that value in such a way that a reference to it can be stored in a variable of reference type. These classes also provide a number of methods for converting among primitive values, as well as supporting such standard methods as equals and hashCode, which are discuses in later.

The class Math provides commonly used mathematical functions such as sine, cosine, and square root. The classes String andStringBuffer similarly provide commonly used operations on character strings.

Classes ClassLoader, Process, Runtime, SecurityManager, and System provide "system operations" that manage the dynamic loading of classes, creation of external processes, host environment inquiries such as the time of day, and enforcement of security policies.

The package java.lang contains classes and interfaces that are essential to the Java language. These include:

- Object, the ultimate superclass of all classes in Java.
- Thread, the class that controls each thread in a multithreaded program.
- Throwable, the superclass of all error and exception classes in Java.
- Wrapper Classes that encapsulate the primitive data types in Java.
- Math, a class that provides standard mathematical methods.
- String, the class that represents strings.

Because the classes in the java.lang package are so essential, the java.lang package is implicitly imported by every Java source file. In other words, you can refer to all of the classes and interfaces in java.lang using their simple names.

### 10.1.1 java.lang.Object

Object class is in the default package i.e java.lang package. The *Object class* is the parent class of all the classes in java by default. It is the topmost class of java. The *java.lang.Object* class is the root of the class hierarchy. Every class has Object as a superclass. Object is the mother of all classes, in other words every other class in java is the subclass of Object class. The Object class defines the basic state and behavior that all objects must have, such as the ability to compare to another object, to convert to a string, to wait on a condition variable, to notify other objects that a condition variable has changed, and to return the object's class.

There are 11 methods in Object class which are explained below:

#### 10.1.1.1 public boolean equals(Object obj)

It is used to compare two objects for equality. This method returns true if the objects are equal, false otherwise. Note that equality does not mean that the objects are the same object. Consider this code that tests two Integers.

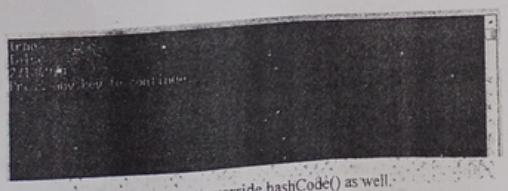
```
Integer a1 = new Integer(1);
Integer a2 = new Integer(1);
if (a1.equals(a2))
  System.out.println("objects are equal");
```

This code will display objects are equal even though a1 and a2 reference two different, distinct objects. They are considered equal because they contain the same integer value. It can be override this method to provide an appropriate equality test. The equals() method provided in the Object class uses the identity operator (==) to determine whether two objects are equal. For primitive data types, this gives the correct result. For objects, however, it does not. The equals() method provided by Object tests whether the object *references* are equal—that is, if the objects compared are the exact same object.

**Example 1: Program using public boolean equals (Object obj)**

```
class A
{
  int a;
  A(int b)
  {
    a=b;
  }

}
class Object_2
{
  public static void main(String arg[])
  {
    A a1=new A(5);
    A a2=a1;
    A a3=new A(10);
    System.out.println(a1.equals(a2));
    System.out.println(a1.equals(a3));
    System.out.println(a1.hashCode());
  }
```

**Output:**



...equals(), you must override hashCode() as well.



#### 10.1.1.2 The Protected finalize() Method

The Object class provides a callback method, finalize(), that *may be* invoked on an object when it becomes garbage. Object's implementation of finalize() does nothing—it can override finalize() to do cleanup, such as freeing resources.

The finalize() method *may be* called automatically by the system, but when it is called, or even if it is called, is uncertain. Therefore, it should not rely on this method to do cleanup code.

#### 10.1.1.3 public final Class getClass()

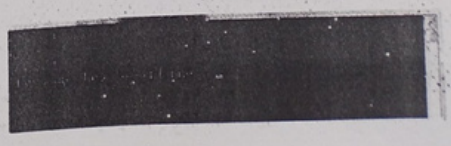
The getClass() method is a final method (cannot be overridden) that returns a runtime representation of the class of this object. This method returns a Class object. You can query the Class object for a variety of information about the class, such as its name, its superclass, and the names of the interfaces that it implements. The following method gets and displays the class name of an object:

```
void FrintClassName(Object obj)
{
  System.out.println("The Object's class is " + obj.getClass().getName());
}
```

**Example 2: Program using get class () method.**

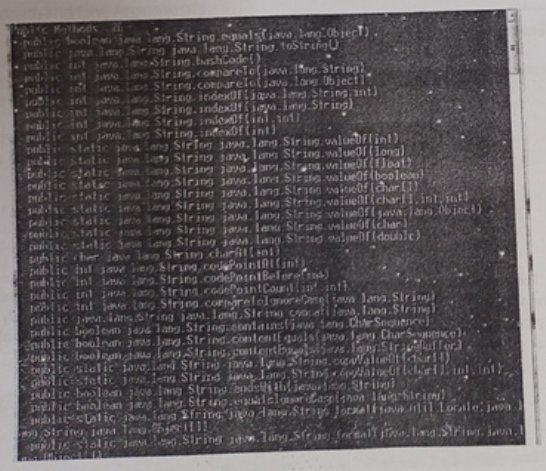
```
class A
{
  int a;
  A(int b)
  {
    a=b;
  }

}
class Object_3
{
  public static void main(String arg[])
  {
    A a1=new A(5);
    A a2=a1;
    A a3=new A(10);
    System.out.println(a1.getClass());
    System.out.println(a2.getClass());
    System.out.println(a3.getClass());
  }
}
```

**Output:**



**Example 3: Program which is works like a javap command.**

```
import java.lang.reflect.Method;
import java.lang.reflect.Field;
class Object_4

{
 public static void main(String arg[])
 {
  String s="Hello";
  Class c=s.getClass();
  Method m[]=c.getMethods();
  System.out.println("Public Methods  "+m.length);
  for(int i=0;i<m.length;i++)
  System.out.println("  "+m[i]);
  Field f[]=c.getFields();
  System.out.println("Public Fields  "+f.length);
  for(int i=0;i<f.length;i++)
  System.out.println("  "+f[i]);
  m=c.getDeclaredMethods();
  System.out.println("All Methods  "+m.length);
  for(int i=0;i<m.length;i++)
  System.out.println("  "+m[i]);
  f=c.getDeclaredFields();
  System.out.println("All Fields  "+f.length);
  for(int i=0;i<f.length;i++)
  System.out.println("  "+f[i]);
 }
}
```





### 10.1.1.4 The toString() Method

The toString() method returns a String representation of the object. The toString() is use to display an object. For example, it can display a String representation of the current Thread like this:

```
System.out.println(Thread.currentThread().toString());
```

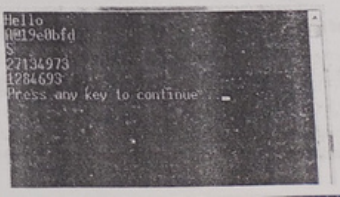
**Output:**

Thread[main,5,main]

The String representation for an object is entirely dependent on the object. The String representation of an Integer object is the integer value displayed as text. The String representation of a Thread object contains various attributes about the thread, such as its name and priority. The toString method is very useful for debugging and would be override in all classes.

**Example 4: Program using hashCode() method.**

```
class A

{
 int a;
 A(int b)
 {
  a=b;
 }

}
class ToString_6
{
 public static void main(String arg[])
 {
  A a1=new A(5);
  String s1="Hello";
  System.out.println(s1);
  System.out.println(a1);//class name@hashcode in hexadecimal
//format
  System.out.println(a1.a);
  System.out.println(a1.hashCode());
  A a2=new A(5);
  System.out.println(a2.hashCode());
 }
}
```



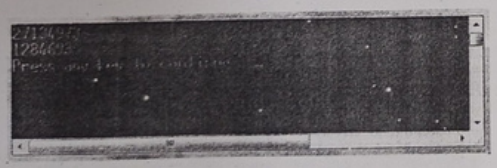
### 10.1.1.5 hashCode()

hashCode() is used for the HashTable. It returns the hash value of the object. The value returned by hashCode() is the object's hash code, which is the object's memory address in hexadecimal. By definition, if two objects are equal, their hash code *must also* be equal. If you override the equals() method, you change the way two objects are equated and Object's implementation of hashCode() is no longer valid. Therefore, if you override the equals() method, you must also override the hashCode() method as well.

**Example 5: Program using hashcode() method.**

```
class A
{
  int a;
  A(int b)
  {
    a=b;
  }

}
class Object1
{
  public static void main(String arg[])
  {
    A a1=new A(5);
  // System.out.println(a1);
  // System.out.println(a1.a);
    System.out.println(a1.hashCode());
      A a2=new A(5);
      System.out.println(a2.hashCode());
  }
}
```

**Ouput:**



### 10.1.1.6 protected Objectclone() Throws CloneNotSupportedException

It creates and returns the copy of the object . If a class, or one of its superclasses, implements the Cloneable interface, it can use the clone() method to create a copy from an existing object as given below:.

```
CloneableObject.clone();
```

Object's implementation of this method checks to see whether the object on which clone() was invoked implements the Cloneable interface. If the object does not, the method throws aCloneNotSupportedException exception.

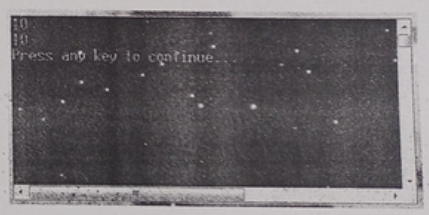


The simplest way to make a class cloneable is to add implements Cloneable to the class declaration. Then respective class objects can invoke the clone() method.

**Example 6: Program to creat clone of object.**

```
class A implements Cloneable
{
  int a;
  public A getClone() throws CloneNotSupportedException
  {
    return (A)clone();
  }

}
class Object_3
{
  public static void main(String arg[]) throws CloneNot Supported Exception
  {
    A a1=new A();
    A a2;
    a1.a=10;
    a2=a1.getClone();
    System.out.println(a1.a);
    System.out.println(a2.a);

  }
}
```

**Output:**




The Object class also provides five methods that are critical when writing multithreaded Java programs. Those methods of Object class, all play a part in synchronizing the activities of independently running threads in a program, which is discussed in a later lesson and won't be covered here. There are five of these methods:

- public final void notify()
- public final void notifyAll()
- public final void wait()
- public final void wait(long timeout)
- public final void wait(long timeout, int nanos)

These methods help to ensure that the threads are synchronized

#### 10.1.1.7 notify()

This method wakes up a single thread that is waiting on this object's monitor. It will,wake up the thread waiting for the objects monitor.

#### 10.1.1.8 notifyAll()

This method wakes up all threads that are waiting on this object's monitor. It will wakes up all the thread that are waiting for the objects monitor.

#### 10.1.1.9 wait()

This method causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.This method causes the current thread to place itself in the wait set for this object and then to relinquish any and all synchronization claims on this object.

#### 10.1.1.10 public final void wait(long timeout)

This method causes the current thread to wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.

#### 10.1.1.11 void wait(long timeout, int nanos)

This method causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.

## 10.2 Java Command Line Arguments

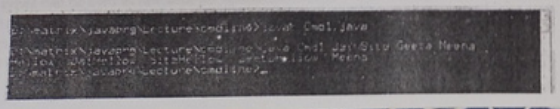
The java command-line argument is an argument that is passed at the time of running the java program. The arguments passed from the console can be received in the java program and it can be used as an input.

So, it provides a convenient way to check the behavior of the program for the different values. You can pass N (1,2,3 and so on) numbers of arguments from the command prompt.

**Example 7: Program using command line arguments**

```
class Cmd1
{
 public static void main(String arg[])
 {
  for(int i=0; i<arg.length;i++)
  {
   System.out.print("Hellow  "+arg[i]);
  }
 }
}
```

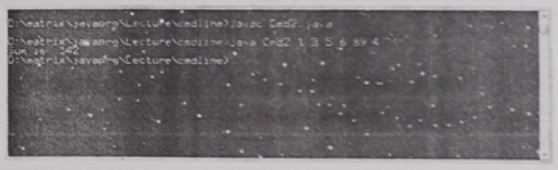
**Output:**





```
class Cmd2
{
 public static void main(String arg[])
 {
  int sum=0;

  for(int i=0; i<arg.length;i++)
  {
   sum+=Integer.parseInt(arg[i]);
  }
  System.out.print("Sum is  "+sum);
 }
}
```

**Output:**



## 10.3 Call by Value and Call by Reference in Java

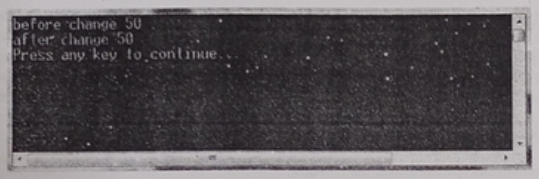
There is only call by value in java, not call by reference. If we call a method passing a value, it is known as call by value. The changes being done in the called method, is not affected in the calling method.

**Example 8: Program of call by value in java.**

In case of call by value original value is not changed.

```
class CallBy_1
{
 int data=50;

 void change(int data)
 {
 data=data+100;//changes will be in the local variable only
 }
  public static void main(String args[])
 {
  CallBy_1 op=new CallBy_1();
  System.out.println("before change "+op.data);
  op.change(500);
  System.out.println("after change "+op.data);

 }
}
```

**Output:**

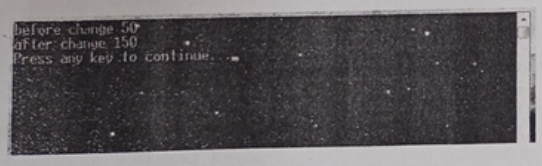



**Example 9: Program of call by value in java.**

In case of call by reference original value is changed if we made changes in the called method. If we pass object in place of any primitive value, original value will be changed. In this example we are passing object as a value.

```
class CallBy_2
{
  int data=50;

  void change(CallBy_2 op)
  {
  op.data=op.data+100;//changes will be in the local variable only
  }

  public static void main(String args[]){
    CallBy_2 op=new CallBy_2();

    System.out.println("before change "+op.data);
    op.change(op);
    System.out.println("after change "+op.data);

  }
}
```

**Output:**

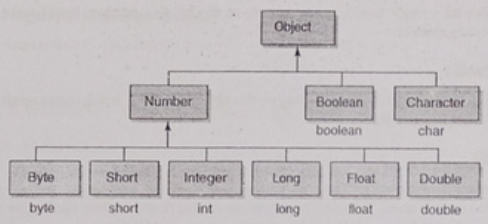



## 10.4 Wrapper Classes

*Wrapper class in java* provides the mechanism *to convert primitive into object and object into primitive.* In Java, a *wrapper class* is defined as a class in which a *primitive value* is wrapped up in Object format. These *primitive wrapper classes* are used to represent primitive data type values as objects. The Java platform



provides wrapper classes for each of the primitive data types. For example, *Integer wrapper class* holds primitive 'int' data type value. Similarly, *Float wrapper class* contain 'float' primitive values, *Character wrapper class* holds a 'char' type value, and *Boolean wrapper class* represents 'boolean' value.




All of the *numeric wrapper classes* are subclasses of the an abstract class java.lang.Number as shown in the above Wrapper Class Hierarchy diagram. There are four other subclasses of Number that are not shown here. BigDecimal and BigInteger are used for high-precision calculations. AtomicInteger and AtomicLong are used for multi-threaded applications. The remaining two wrapper classes Character & Boolean are direct subclasses of java.lang.Object.

The list of eight wrapper classes is given below:

| Primitive data type | Wrapper class |
|---|---|
| byte | Byte |
| short | Short |
| int | Integer |
| long | Long |
| float | Float |
| double | Double |
| char | Character |
| boolean | Boolean |

**Example 10: Program to convert Primitive to Wrapper class.**

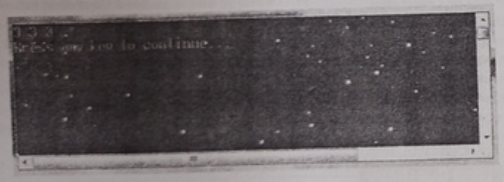
```
public class Wraper_1
{
  public static void main(String args[])
  {              //Converting int into Integer
    int a=20;
    Integer i=Integer.valueOf(a);//converting int into Integer
    Integer  j=a;//autoboxing,  now  compiler  will  write  Integer.valueOf(a)
// internally
    System.out.println(a+" "+i+" "+j);
  }
}
```

**Output:**

**Example 11: Program to convert Wrapper to Primitive data type.**

```
public class Wrapper_2
{
  public static void main(String args[])
  {
  //Converting Integer to int
    Integer a=new Integer(3);
    int i=a.intValue();//converting Integer to int
    int j=a;//unboxing, now compiler will write a.intValue() internally
    System.out.println(a+" "+i+" "+j);
  }
}
```

**Output:**



### 10.4.1 Need of Wrapper Classes in Java

Primitive types are used in many places but not all. In some cases it can't use primitive values, so wrapper classes are required. For example, if we need to store numbers in a collection, we can't use primitives because collections such as List, Set, and Map need objects as their elements. In such cases you must use wrapper classes.

### 10.4.2 Difference Between Primitive Data Types and Wrapper Classes

The main difference between primitive data types and wrapper classes (wrapper types) in java is that the primitive types can be used as raw data for operations such as arithmetic, logical, etc. and wrapper classes acts as data holders for these primitive data types.

The primitive data type values will be stored in Stack Memory whereas wrapper class objects (like any other java objects) are stored in Heap Memory.



## 10.5 java.util Package

Java.util package contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes. The collections framework and legacy collection classes will discussed in chapters. Here we discussed Date class, Random class and Scanner classes of java.util package.

### 10.5.1 java.util.Scanner Class

The java.util.Scanner class provides way to read input from the keyboard. The *Java Scanner* class breaks the input into tokens using a delimiter that is white space by default. 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.

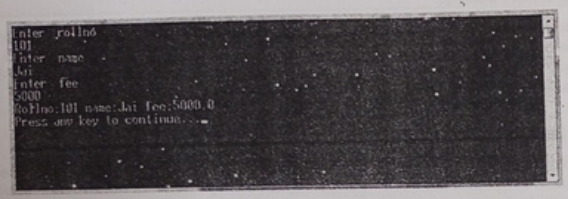
There is a list of commonly used Scanner class methods:

| 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. |

**Example 12: Program to get input from console using Scanner class.**

```
import java.util.Scanner;
class Scanner_1
{

  public static void main(String args[])
  {
    Scanner sc=new Scanner(System.in);
    System.out.println("Enter  rollno");
    int rollno=sc.nextInt();
    System.out.println("Enter  name");
    String name=sc.next();
    System.out.println("Enter  fee");
    double fee=sc.nextDouble();
    System.out.println("Rollno:"+rollno+" name:"+name+" fee:"+fee) ;
    sc.close();
  }
}
```

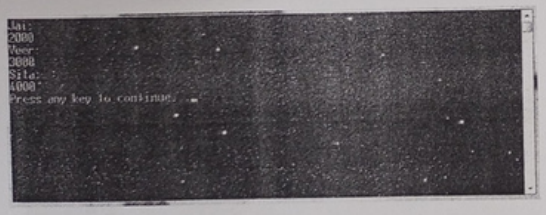
**Output:**




**Example 13: Program using Scanner with delimiter.**

```
import java.util.*;
public class Scanner_2
{
  public static void main(String args[])
  {
    String s1 = "Jai: 2000 Veer: 3000 Sita: 4000";
    Scanner sc = new Scanner(s1).useDelimiter("\\s");
    System.out.println(sc.next());
    System.out.println(sc.nextInt());
    System.out.println(sc.next());
    System.out.println(sc.nextInt());
    System.out.println(sc.next());
    System.out.println(sc.nextInt());
    sc.close();
  }
}
```

**Output:**




### 10.5.2 java.util.Date Class

The java.util.Date representing date and time. The java.util.Date class represents date and time in java. It provides constructors and methods to deal with date and time in java.

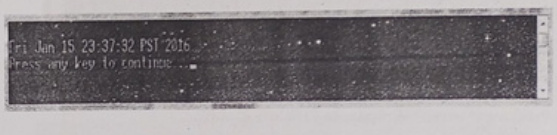
The java.util.Date class implements Serializable, Cloneable and Comparable<Date> interface. It is inherited by java.sql.Date, java.sql.Time and java.sql.Timestamp interfaces.



**Get Current Date:**

```
java.util.Date date=new java.util.Date();
System.out.println(date);
```

**Output:**




## Very Short Questions

1. Name the class which is parent of all classes by default.
2. List the Object class methods which are used in multithreading.
3. What is wrapper class?
4. What is use of Scanner class?
5. Write syntax to read from keyboard using Scanner class.

## Short Questions

1. Write short note on Object class.
2. Explain call by value and call by value in java with example.
3. What is wrapper class? Write down uses of wrapper classes.
4. Write short note on some important conversion from wrapper class objects to primitives.
5. Write short note on some important conversion from primitives to wrapper class objects.
6. Write difference between wrapper class and primitives.
7. Write syntax of Scanner class constructor to read from file.

## Long Questions

1. Write note on Object class. Explain methods of Object class in detail.
2. What are wrapper classes? Write note on wrapper to primitive and vice versa.
3. Explain Scanner class with common constructors and method with example.

# CHAPTER » 11

# File Handling in Java

## 11.1 Introduction to File Handling

The input and output operation that we have performed so far were done through screen and keyboard only. After the termination of program all the entered data is lost because primary memory is volatile. If the data has to be used later, then it becomes necessary to keep it in permanent storage device. So the Java language provides the concept of file through which data can be stored on the disk or secondary storage device. The stored data can be read whenever required.

**Streams:** A stream is an abstraction that either produce or consume information. Java programs performs I/O through streams. A stream is linked to a physical device by the Java Input/output . All streams behave in the same manner, even if the actual physical devices to which they are linked. Streams are clean way to deal with input/output without having every part of our code understand the difference between a keyword and a network, for example java implements stream within class hierarchies define in the java.io package.

*Java I/O* (Input and Output) is used to process the input and produce the output based on the input.

We can perform *file handling in java* by java IO API.

A stream is a sequence of data. In Java a stream is composed of bytes. It's called a stream because it's like a stream of water that continues to flow.

In java, 3 streams are created for us automatically. All these streams are attached with console.

1) **System.out:** standard output stream.
2) **System.in:** standard input stream.
3) **System.err:** standard error stream.

## 11.2 OutputStream Class and InputStream Class

Java application uses an *OutputStream* to write data to a destination, it may be a file, an array, peripheral device or socket. Java application uses an *InputStream* to read data from a source, it may be a file,an array,peripheral device or socket.

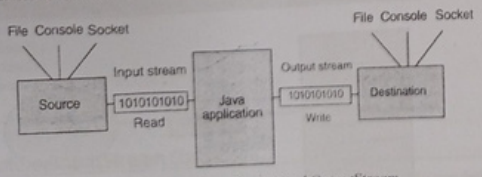



*Figure to show inputStream and OutputStream*

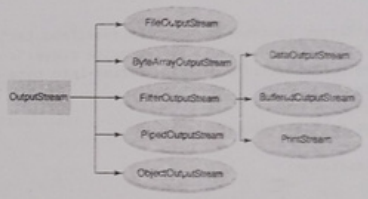


### 11.2.1 OutputStream

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

**Methods of OutputStream class**

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

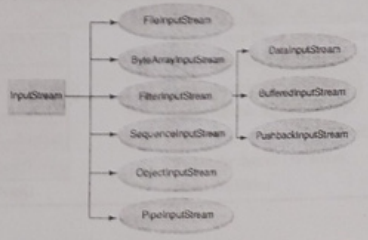



### 11.2.2 InputStream Class

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

**Methods of InputStream class**

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



## 11.3 Java FileOutputStream Class and Java FileInputStream Class

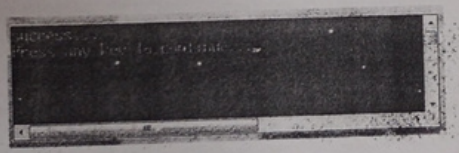
### 11.3.1 Java FileOutputStream

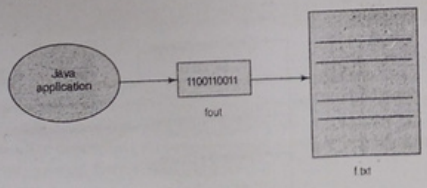
Java *FileOutputStream* is an output stream for writing data to a file. To write primitive values then use FileOutputStream. Instead, for character-oriented data, prefer FileWriter.

**Example 1: Program using Java FileOutputStream class.**

```
import java.io.*;
class File_1

{
 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:**








### 11.3.2 Java FileInputStream Class

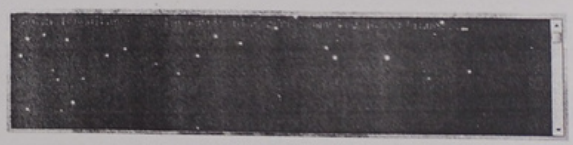
Java FileInputStream class obtains input bytes from a file. It is used for reading streams of raw bytes such as image data. For reading streams of characters, consider using FileReader.

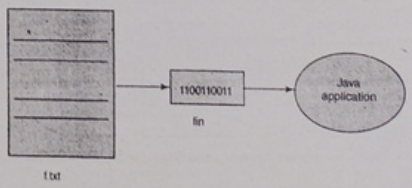
It should be used to read byte-oriented data for example to read image, audio, video etc.

**Example 2: Program using of FileInputStream class.**

```
import java.io.*;
class File_2

{
 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:**





**Example 3: Program to copy the file content into another file.**

```
import java.io.*;
class File_3
{
  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();
  }
}
```

## 11.4 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 4:**

By this example, the textual information is writing by 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 to 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");
  }
}
```

## 11.5 Java BufferedInputStream Class

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



**Example 5: Program using BufferedInputStream.**

```
import java.io.*;
class File_9
{
  public static void main(String args[])
  {
  try
  {
    FileInputStream fin=new FileInputStream("fi.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);
    }
  }
}
```

## 11.6 Java FileWriter and FileReader

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.

Good developer always suggested not to use the FileInputStream and FileOutputStream classes if you have to read and write the textual information.

### 11.6.1 FileWriter Class

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

Constructors and Methods 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. |
| public void write(String text) | writes the string into FileWriter. |
| public void write(char c) | writes the char into FileWriter. |
| 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. |

**Example 6: Program using Filewriter Class.**

```
import java.io.*;
class File_10
{
 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");
 }
}
```

### 11.6.2 FileReader Class

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

Constructors and Methods of FileReader 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. |
| public int read() | It returns a character in ASCII form. It returns -1 at the end of file. |
| public void close() | It closes FileReaderclass. |

**Example 7: Program using FileReader class.**

```
import java.io.*;
class file_11
{
 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();
```



```
 }
}
```

## 11.7 CharArrayWriter Class

The CharArrayWriter class can be used to write data to multiple files. This class implements the Appendable interface. Its buffer automatically grows when data is written in this stream.

**Example 8: Program using CharArrayWriter class.**

```
import java.io.*;
class File_12
{
 public static void main(String args[])throws Exception
 {

  CharArrayWriter out=new CharArrayWriter();
  out.write("File handling program");

  FileWriter f1=new FileWriter("a.txt");
  FileWriter f2=new FileWriter("b.txt");
  FileWriter f3=new FileWriter("c.txt");
  FileWriter f4=new FileWriter("d.txt");

  out.writeTo(f1);
  out.writeTo(f2);
  out.writeTo(f3);
  out.writeTo(f4);

  f1.close();
  f2.close();
  f3.close();
  f4.close();
 }
}
```

## 11.8 Reading Data from Keyboard

The following are ways to read data from the keyboard:

- InputStreamReader and BufferedReader class
- Console
- Scanner
- DataInputStream etc.

### 11.8.1 InputStreamReader Class and BufferedReader 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 9: Program using InputSteamReader class and BufferdReader class.**

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

```
import java.io.*;
class File_13

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

    InputStreamReader r=new InputStreamReader(System.in);
    BufferedReader br=new BufferedReader(r);

    System.out.println("Enter First your name");
    String name=br.readLine();
    System.out.println("Enter First your name");
    String lname=br.readLine();
    System.out.println("Welcome "+name+" "+lname);
  }
}
```

## 11.8.2 Console Class

The Java Console class is be used to get input from console. It provides methods to read text and password. The java.io.Console class is attached with system console internally. The Console class is introduced since 1.5.

**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. |

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



```
public static Console console(){}
```

Code to get the instance of Console class.

```
Console c=System.console();
```

**Example 10: Program using Console class.**

Program to reverse the given array taking input from user by using Console class.

```
import java.io.Console;
public class Reverse

{
  public static void main(String arg[])
  {
    Console con=System.console();
    System.out.println("Enter array length");
    int n=Integer.parseInt(con.readLine());
    int ml[]=new int[n];
    System.out.println("Enter Array Elements:");
    for(int  i=0;i<n;i++)
    {
      System.out.print("Array element is"+(i+1));
      ml[i]=Integer.parseInt(con.readLine());
    }
    //normal output
    for(int  i=0;i<n;i++)

      System.out.print(ml[i]+"\t");

    System.out.println();

    // reverse array elment

      for (int  i=0;i<n;i++)
      {
          int t=ml[i];
          ml[i]=ml[n];
          ml[n]=t;
          n--;

      }

   // sorted output
    for(int  i=0;i<n;i++)
    {

      System.out.print(ml[i]+"\t");

    System.out.println();
    }
```

```
    }
}
```

**Output:**

## 11.9 java.io.PrintStream Class

The PrintStream class provides methods to write data to another stream. The PrintStream class automatically flushes the data so there is no need to call flush() method. Moreover, its methods don't throw IOException.

**Example 11: Program using FileOutputStream class.**

```
import java.io.*;
class File_15
{
 public static void main(String args[])throws Exception
 {

   FileOutputStream fout=new FileOutputStream("myfile.txt");
   PrintStream pout=new PrintStream(fout);
   pout.println(1900);
   pout.println("Hello Java Program");
   pout.println("Welcome to Java");
   pout.close();
   fout.close();

 }
}
```

■ ■ ■

### Very Short Questions

1. Name the package which is used for file handling.
2. What is main use of Bufferedreader class?
3. What do you mean by byte stream classes?
4. What do you mean by character stream classes?
5. What is use of Console class?
6. Write down the package of Console class.
7. What is out? (System.out.println())



### Short Questions

1. **What are byte stream classes? Name any five byte stream classes.**
2. **What are character stream classes? Name any five character stream classes.**
3. **Write java code to read from console using BufferedReader class.**
4. **Write java code to read from console using Console class.**

### Long Questions

1. **What do you mean by file handling? Write java code to read and write from a file using byte stream classes.**
2. **Write java code to read, write and copy to another file using character stream classes.**
3. **Explain Console class with common constructors and method with example. Write java code for read from console using Console class.**

# CHAPTER » 12

# GUI in Java Using AWT and Swing

## 12.1 Java AWT

The AWT provides a well-designed object-oriented interface to these low-level services and resources.

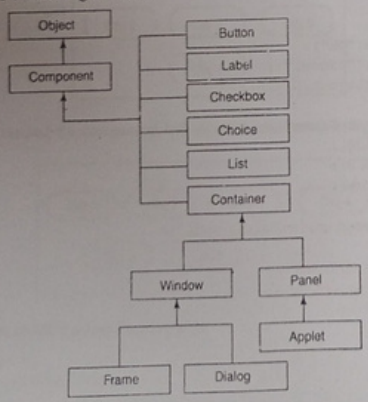
*Java AWT* (Abstract Windowing Toolkit) is an API to develop GUI or window-based application in java. Java AWT components are platform-dependent that is components are displayed according to the view of operating system. AWT components are heavy weight that is its components uses the resources of system. The Java programming language class library provides a user interface toolkit called the Abstract Window Toolkit, or the AWT. The AWT is both powerful and flexible.

AWT contains large number of classes and methods that allows you to create and manage windows GUI application. AWT is the foundation upon which *Swing* is made. It is used for GUI programming in Java. But now a days it is merely used because most GUI java programs are implemented using *Swing* because of its rich implementation of GUI controls and light-weighted nature.

The java.awt package provides classes for AWT API (Application Programming Interface) such as TextField, Label, TextArea, Radio Button, CheckBox, Choice, List etc.

### 12.1.1 Java AWT Hierarchy

The hierarchy of Java AWT classes is given here:






### 12.1.2 Container and Container Types

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

The AWT provides four container classes. They are class Window and its two subtypes class Frame and class Dialog -- as well as the Panel class. In addition to the containers provided by the AWT, the Applet class is a container -- it is a subtype of the Panel class and can therefore hold components. Brief descriptions of each container class provided by the AWT are provided below.

| | |
|---|---|
| Window | The window is the container that has no borders and menu bars. Frame, dialog or another window are used for creating a window. A top-level display surface (a window). An instance of the Window class is not attached to nor embedded within another container. An instance of the Window class has no border and no title. |
| Frame | The Frame is the container that has title bar. Title bar has an image icon, a title string and four button that are minimize, maximize, resize and close. Title bar is decoration of fame. The default layout of frame is border layout. A top-level display surface (a window) with a border and title. An instance of the Frame class may have a menu bar. It is otherwise very much like an instance of the Window class. |
| Dialog | A top-level display surface (a window) with a border and title. An instance of the Dialog class cannot exist without an associated instance of the Frame class. |
| Panel | The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc. A generic container for holding components. An instance of the Panel class provides a container to which to add components. |

**Useful Methods of Container 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. |

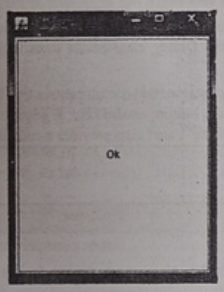
To create simple AWT example, it need a frame. There are two ways to create a frame in AWT.

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

**Example 1: Program using AWT by inheritance.**

```
import java.awt.*;
class MyFrame extends Frame

{
  public static void main(String args[])
  {
    MyFrame f=new MyFrame();
    //or
```

```
        //Frame f=new MyFrame();
        //or
        //Frame f=new Frame(); // no need to extends
            Button b1=new Button("Ok");
            f.add(b1);
        f.setTitle("First Frame");
        f.setSize(200,300);
        f.setLocation(0,0);
        //or
        //f.setBounds(0,0,200,300);
        //f.setResizable(false);
        //f.setUndecorated(true);
        f.setVisible(true);
        //or
        //f.show(); //deprecated
    }
}
```

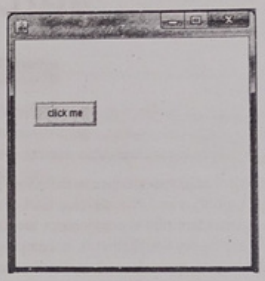



**Example 2: Program of AWT setting size and position of controls.**

```
import java.awt.*;
class Myframe_2 extends Frame

{
 Myframe_2()
 {
  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[])
 {
  Myframe_2 f=new Myframe_2();
 }
}
```



**Output:**

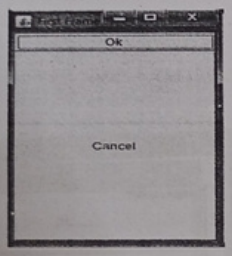



The *setBounds(int x axis, int y axis, int width, int height)* method is used in the above example that sets the position of the AWT button.

**Example 3: Program using direction of FrameLayout class.**

```
import java.awt.*;
class MyFrame extends Frame

{
  Button b1,b2;
  MyFrame()
  {
    b1=new Button("Ok");
    b2=new Button("Cancel");
    add(b1,BorderLayout.NORTH);
    add(b2);
    setTitle("First Frame");
    setSize(200,300);
    setLocation(0,0);
    setVisible(true);
  }
  public static void main(String args[])
  {
    MyFrame f=new MyFrame();
    //or
    //new MyFrame();
  }
}
```

**Output:**

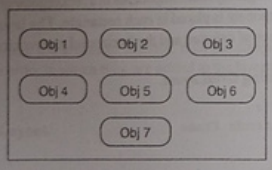



### 12.1.3 Component Layout

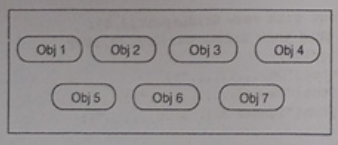
The AWT provides five layout managers. They range from very simple to very complex. The FlowLayout class and the BorderLayout class.

#### 12.1.3.1 Flow Layout

The FlowLayout class places components in a container from left to right. When the space in one row is completed another row is started. The single-argument version of a container's add() method is used to add components.




If the container is resized, the components will adjust themselves to new positions.



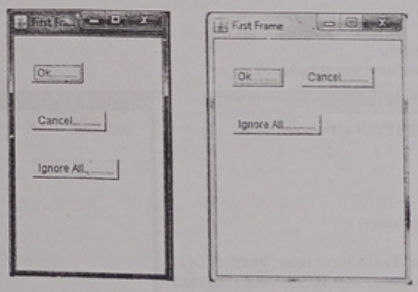

**Example 4: Program of AWT using FlowLayOut.**

```
import java.awt.*;
class MyFrame extends Frame
```



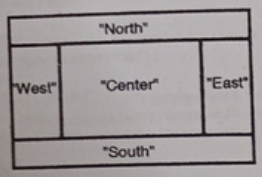
```
{
  Button b1,b2,b3;
  MyFrame()
  {
//FlowLayout flow =new FlowLayout();
//FlowLayout flow =new FlowLayout(FlowLayout.RIGHT);
FlowLayout flow =new FlowLayout(FlowLayout.LEFT,20,30);

    setLayout(flow);
    b1=new Button("Ok.........");
    b2=new Button("Cancel.....");
    b3=new Button("Ignore All.....");
    add(b1);
    add(b2);
    add(b3);
    setTitle("First Frame");
    setSize(200,300);
    setLocation(0,0);
    setVisible(true);
  }
  public static void main(String args[])
  {
    MyFrame f=new MyFrame();
  }
}
```

**Output:**




#### 12.1.3.2 Border Layout

The BorderLayout class has five zones. The zones are named "North", "South", "East", "West", and "Center". A single component can be placed in each of these five zones. When the enclosing container is resized, each border zone is resized just enough to hold the component placed within. Any excess space is given to the center zone. The two-argument version of a container's add() method is used to add components. The first argument is a String object that names the zone in which to place the component.




Each container class has a default layout manager. The default layout manager for the *Frame class* and *Dialog class* is the *BorderLayout* manager.

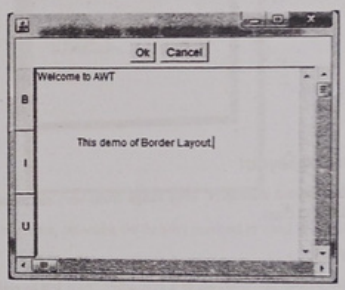
**Example 5: Program using differ cut layouts.**

```
import java.awt.*;
class MyFrame extends Frame

{
Button b1,b2,b3,b4,b5;
Panel p1,p2;
Scrollbar sb1,sb2;
TextArea ta1;
MyFrame()
{
 p1=new Panel();
 b1=new Button("Ok");
 b2=new Button("Cancel");
 p1.add(b1);
 p1.add(b2);

 p2=new Panel();
 p2.setLayout(new GridLayout(3,1));
 b3=new Button("B");
 b4=new Button("I");
 b5=new Button("U");
 p2.add(b3);
 p2.add(b4);
 p2.add(b5);

 ta1=new TextArea();

 sb1=new Scrollbar(Scrollbar.VERTICAL);
 sb2=new Scrollbar(Scrollbar.HORIZONTAL);

 add(p1,BorderLayout.NORTH);
 add(p2,BorderLayout.WEST);
 add(sb1,BorderLayout.EAST);
 add(sb2,BorderLayout.SOUTH);
 add(ta1);
 setSize(200,300);
 setVisible(true);
```



```
}
public static void main(String args[])
{
 new MyFrame();
}
}
```

**Output:**




### 12.1.3.3 GridLayout

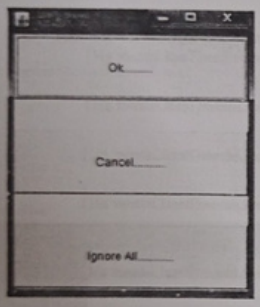
The *GridLayout* layout manager lays out components in a rectangular grid. The container is divided into equally sized rectangles. One component is placed in each rectangle. The layout manager takes four parameters. The number of rows, the number of columns and the horizontal and vertical gaps between components.

**Example 6: Program using GridLayout.**

```
import java.awt.*;
class MyFrame extends Frame

{
 Button b1,b2,b3;
 MyFrame()
 {
  //GridLayout grid =new GridLayout(3,1);
 GridLayout grid =new GridLayout(3,1,20,30);
  setLayout(grid);
  b1=new Button("Ok.....");
  b2=new Button("Cancel.....");
  b3=new Button("Ignore All.....");
  add(b1);
  add(b2);
  add(b3);
  setTitle("First Frame");
  setSize(200,300);
  setLocation(0,0);
```

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

**Output:**




#### 12.1.3.4 CardLayout

The CardLayout LayoutManager provides the means to manage multiple components, displaying one at a time. Components are displayed in the order in which they are added to the layout, or in an arbitrary order by using an assignable name.

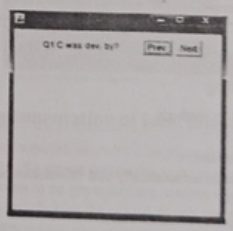
**Example 7: Program using Card Layout.**

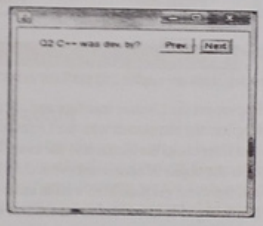
```
import java.awt.*;
import java.awt.event.*;
class MyFrame24 extends Frame implements ActionListener
{
  Panel p1,p2,mp;
  Button b1,b2;
  CardLayout c1;
  Label l1,l2;
  MyFrame24()
  {
    l1=new Label("Q1 C was dev. by?");
    l2=new Label("Q2 C++ was dev. by?");
    p1=new Panel();
    p1.add(l1);
```



```
    p2=new Panel();
    p2.add(l2);
    mp=new Panel();
    c1=new CardLayout();
    mp.setLayout(c1);
    mp.add(p1,"first");
    mp.add(p2,"second");
    b1=new Button("Prev.");
    b2=new Button("Next");
    b1.addActionListener(this);
    b2.addActionListener(this);
    setLayout(new FlowLayout());
    add(mp);add(b1);add(b2);
    setSize(300,300);setVisible(true);
  }
  public static void main(String args[])
  {
  new MyFrame24();
  }
  public void actionPerformed(ActionEvent ae)
  {
    Button b=(Button)ae.getSource();
    if(b==b1)
    c1.previous(mp);
    else
    c1.next(mp);
  }
}
```

**Output:**




## 12.2. Event and Listener (Java Event Handling)

Events are methods which are invoked when external event takes place. Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. 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.

Change in the state of an object is known as event i.e. event describes the change in state of source. Events are generated as result of user interaction with the graphical user interface components. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page are the activities that causes an event to happen.

**Types of Event**

The events can be broadly classified into two categories:

- **Foreground Events** - Those events which require the direct interaction of user. They are generated as consequences of a person interacting with the graphical components in Graphical User Interface. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page etc.
- **Background Events** - Those events that require the interaction of end user are known as background events. Operating system interrupts, hardware or software failure, timer expires, an operation completion are the example of background events.

**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.
3. The User clicks the button and the event is generated.
4. Now the object of concerned event class is created automatically and information about the source and the event get populated with in same object.
5. Event object is forwarded to the method of registered listener class.
6. The method is now get executed and returns.

**Points to remember about listener:**

- In order to design a listener class we have to develop some listener interfaces. These Listener interfaces forecast some public abstract callback methods which must be implemented by the listener class.



- If you do not implement the any if the predefined interfaces then your class can not act as a listener class for a source object.

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:

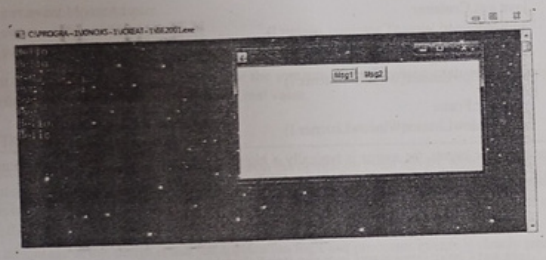
1. Same class
2. Other class
3. Annonymous class

**Example 8: Program of event handling in other class class.**

```
import java.awt.*;
import java.awt.event.*;
class MyFrame6 extends Frame
{
  Button b1,b2;
  MyFrame6()
  {
    setLayout(new FlowLayout());
    b1=new Button("Msg1");
    b2=new Button("Msg2");
    b1.addActionListener(new A());
    b2.addActionListener(new B());
    add(b1);add(b2);
    setSize(400,400);setVisible(true);
  }
```

```
    public static void main(String args[])
    {
     new MyFrame6();
    }
}
class A implements ActionListener
{
 public void actionPerformed(ActionEvent ae)
 {
  System.out.println("Hello");
 }
}
class B implements ActionListener
{
 public void actionPerformed(ActionEvent ae)
 {
  System.out.println("Bye");
 }
}
```

**Output:**



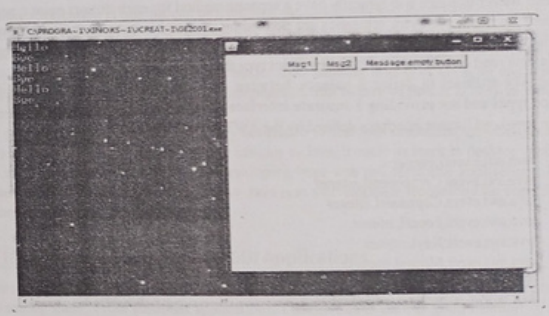
**Example 9: Programof event handling within class.**

```
import java.awt.*;
import java.awt.event.*;
class MyFrame8 extends Frame implements ActionListener

{
  Button b1,b2,b3;
  MyFrame8()
  { setLayout(new FlowLayout());
    b1=new Button("Msg1");
    b2=new Button("Msg2");
    b3=new Button("Message empty button");
    b1.addActionListener(this);
    b2.addActionListener(this);
    add(b1);
    add(b2);
```



```
    add(b3);
    setSize(400,400);
    setVisible(true);
  }
  public static void main(String args[])
  {
    new MyFrame8();
  }
  public void actionPerformed(ActionEvent ae)
  {
    Button b=(Button)ae.getSource();
    if(b==b1)
    System.out.println("Hello");
    else
    System.out.println("Bye");
  }
}
```

**Output:**



## 12.3 Implementation of Event Driven Mechanism

In the delegation model, an event is an object that describes a state change in a source. It can be generated as a consequence of a person interacting with the elements in a graphical user interface. Some of the activities that cause events to be generated are pressing a button, entering a character via the keyboard, selecting an item in a list, and clicking the mouse. Events may also occur that are not directly caused by interactions with a user interface.

For example, an event may be generated when a timer expires, a counter exceeds a value, software or hardware failure occurs, or an operation is completed.

Events are methods which are invoked when external event takes place. Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism has the code which is known as event handler that is executed when an event occurs. Java Uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events. The Delegation Event Model has the following key participants namely:

- **Source** - The source is an object on which event occurs. Source is responsible for providing information of the occurred event to it's handler. Java provide as with classes for source object.
- **Listener** - It is also known as event handler. Listener is responsible for generating response to an event. From java implementation point of view the listener is also an object. Listener waits until it receives an event. Once the event is received, the listener process the event an then returns.

**Callback Methods**

These are the methods that are provided by API provider and are defined by the application programmer and invoked by the application developer. Here the callback methods represents an event method. In response to an event java jre will fire callback method. All such callback methods are provided in listener interfaces.

If a component wants some listener will listen to it's events the the source must register itself to the listener.

## 12.4 Event Listeners

An EventListener interface will typically have a separate method for each distinct event type the event class represents. So, particular event semantics are defined by the combination of an Event class paired with a particular method in an EventListener. For example, the FocusEventListener interface defines two methods, focusGained() and focusLost(), one for each event type that FocusEvent class represents.

The API attempts to define a balance between providing a reasonable granularity of Listener interface types and not providing a separate interface for every single event type.

The low-level listener interfaces defined by the AWT are as follows:

java.util.EventListener
java.awt.event.ComponentListener
java.awt.event.ContainerListener
java.awt.event.FocusListener
java.awt.event.KeyListener
java.awt.event.MouseListener
java.awt.event.MouseMotionListener
java.awt.event.WindowListener

## 12.5 Event Sources

Because the events fired by an event source are defined by particular methods on that object, it is completely clear from the API exactly which events an object supports.

All AWT event sources support a multicast model for listeners. This means that multiple listeners can be added and removed from a single source. *The API makes no guarantees about the order in which the events are delivered to a set of registered listeners for a given event on a given source.* Additionally, any event which allows its properties to be modified will be explicitly copied such that each listener receives a replica of the original event.

Event delivery is synchronous, however programs should not make the assumption that the delivery of an event to a set of listeners will occur on the same thread.

Once again, a distinction is drawn between low-level and semantic events. For low-level events, the source will be one of the visual component classes (Button, Scrollbar, etc) since the event is tightly bound



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

```
public void addTypeListener(TypeListener el)
```

Here, Type is the name of the event and el is a reference to the event listener.

For example, the method that registers a keyboard event listener is called addKeyListener( ). The method that registers a mouse motion listener is called addMouseMotionListener( ). When an event occurs, all registered listeners are notified and receive a copy of the event object. This is known as multicasting the event. In all cases, notifications are sent only to listeners that register to receive them.

The low-level listeners are defined on the following components:

java.awt.Component
addComponentListener(ComponentListener l)
addFocusListener(FocusListener l)
addKeyListener(KeyListener l)
addMouseListener(MouseListener l)
addMouseMotionListener(MouseMotionListener l)

java.awt.Container
addContainerListener(ContainerListener l)

- java.awt.Dialog
  addWindowListener(WindowListener l)
- java.awt.Frame
  addWindowListener(WindowListener l)

For semantic events, the source is typically a higher-level interface representing the semantic model. Following are the semantic listeners defined for AWT components:

- java.awt.Button
  addActionListener(ActionListener l)
- java.awt.Choice (implements java.awt.ItemSelectable)
  addItemListener(ItemListener l)
- java.awt.Checkbox (implements java.awt.ItemSelectable)
  addItemListener(ItemListener l)
- java.awt.CheckboxMenuItem (implements java.awt.ItemSelectable)
  addItemListener(ItemListener l)
- java.awt.List (implements java.awt.ItemSelectable)
  addActionListener(ActionListener l)
  addItemListener(ItemListener l)
- java.awt.MenuItem
  addActionListener(ActionListener l)
- java.awt.Scrollbar (implements java.awt.Adjustable)
  addAdjustmentListener(AdjustmentListener l)

- java.awt.TextArea
  addTextListener(TextListener l)
- java.awt.TextField
  addActionListener(ActionListener l)
  addTextListener(TextListener l)

## 12.6 Adapters

Since many of the EventListener interfaces are designed to listen to multiple event subtypes, the AWT will provide a set of abstract "adapter" classes, one which implements each listener interface. This will allow programs to easily subclass the Adapters and override only the methods representing event types they are interested in.

The Adapter classes provided by AWT are as follows:

java.awt.event.ComponentAdapter
java.awt.event.ContainerAdapter
java.awt.event.FocusAdapter
java.awt.event.KeyAdapter
java.awt.event.MouseAdapter
java.awt.event.MouseMotionAdapter
java.awt.event.WindowAdapter

There are no default Adapters provided for the semantic listeners, since each of those only contain a single method and an adapter would provide no real value.

## 12.7 The Delegation Event Model

The modern approach to handling events is based on the delegation event model, which defines standard and consistent mechanisms to generate and process events. Its concept is quite simple, a source generates an event and sends it to one or more listeners. In this scheme, the listener simply waits until it receives an event. Once received, the listener processes the event and then returns.

The advantage of this design is that the application logic that processes events is cleanly separated from the user interface logic that generates those events. A user interface element is able to delegate the processing of an event to a separate piece of code. In the delegation event model, listeners must register with a source in order to receive an event notification. This provides an important benefit that is notifications are sent only to listeners that want to receive them.

This is a more efficient way to handle events than the design used by the old Java 1.0 approach. Previously, an event was propagated up the containment hierarchy until it was handled by a component. This required components to receive events that they did not process, and it wasted valuable time. The delegation event model eliminates this overhead.

Java also allows you to process events without using the delegation event model. This can be done by extending an AWT component.

**Purpose Delegation of Event Model modal**

There is great roll of new event model into the AWT. This new model has also been adopted by the JavaBeans architecture for general event processing and is described at a high level in the JavaBeans Specification document. The JDK1.1 will introduce a new delegation-based event model in AWT in order to:



- Resolve the problems mentioned previously.
- Provide a more robust framework to support more complex java programs.

**Design Goals**

The primary design goals of the new model in the AWT are the following:

- Simple and easy to learn.
- Support a clean separation between application and GUI code.
- Facilitate the creation of robust event handling code which is less error-prone (strong compile-time checking).
- Flexible enough to enable varied application models for event flow and propagation.
- For visual tool builders, enable run-time discovery of both events that a component generates as well as the events it may observe.
- Support backward binary compatibility with the old model.

## 12.8 The Standalone GUI Application

A standalone application is one that can be executed independently and would execute and produce some output either as a GUI or on the JVM console. Any java class with a main method can be considered a mini standalone java application. It is very important to remember that regardless of how difficult it is to build a standalone application in Java, it's always easier than building it in C or C++, or any other statically compiled language as they are not platform independent. But in case of java it is easy as java application are platform independent.

Java programming language was primarily developed to deal with embedded applications. But with the introduction of Swing and other user interface utilities to Java, it made its mark in desktop applications too. Now the trend is to use Java technology in developing large web and enterprise applications. Java is more focused on web then desktop applications. Now Java is as a development technology for stand alone desktop application development in present scenario.

## 12.9 Java is Suitable for Desktop GUI Applications

There are several applications built on Java technology. Development tools like Eclipse is one the popular example among them. From enterprise applications, Lotus Notes is another good example of desktop Java applications. Such examples show that Java can be used extensively to develop desktop applications apart from installation run times and splash screens. Let examine the benefits of using Java as development technology for stand alone applications.

### 1. Independence of Java

The most important feature of Java technology is that it can be used on any platform and on any operating system. Java Virtual Machines are developed for almost all types of hardware and operating system including high-end servers to small palmtop computers and mobile phones. A program written in Java programming language can be executed anywhere provided the Java runtime is available to understand the Byte Code of Java. This feature adds a great value to Java as a development technology for desktop applications. A developer can use the Eclipse application on different operating systems including Window, Linux and Mac Operating System. This means that desktop applications developed using Java can be used on multiple platforms and multiple operating systems without or with minor changes to it.

2. **Security**

   For stand alone applications, security features are more important. There are applications developed in programming languages that are native to operating system and are found shipping viruses. In this area Java is much secured and reliable technology. Java virtual machine itself intercepts all calls made to operating system from the application – weather it is to establish network connection or to print a file or to access a file from file system of operating system.

3. **Manageability**

   Java is almost pure object oriented programming language. So being an object oriented programming language it is easier to manage than process oriented native languages. Features of Java language like reusability, polymorphism, plug-ability, declarative-ness, configurability makes it most maintainable language.

4. **Pluggable User Interface**

   There was a time when Java language was much criticized for it's being poor in look and feel area. But, with the introduction of lightweight Swing components in Java and emergence of reusable JavaBeans, has made Java bounce back in market for developing competitive user interface. Most advantageous feature of Swing technology is that look and feel of the entire application can be changed at fly (at runtime programmatically).

5. **Open Source**

   Being an open source technology, Java is free to use even for commercial purpose. This way, in order to develop application over the top of Java technology one does not needs to spend money after licensing. This feature is important in developing desktop applications within a limited budget. Similar features which can be achieved with native languages can also be achieved using Java technology along with developing the application economically.

6. **Remote Access**

   Reusability of Java objects is one of the well-known features that Java offers. Additional to reusability, Java also offers remote access to object by Remote Method Invocation and CORBA. Using RMI, the developer can access Java objects that are residing on remote hosts. This architecture makes it possible to reuse objects implementing core business logic of the application. RMI also is extensively used to develop LAN applications.

7. **Networking Features**

   Sometimes stand-alone desktop applications need to access data from network host or internet. In such a case Java has network utilities which facilitates to establish connections to such hosts and access data in a standardized and secured manner. Here we have to remember that before establishing network connection Java checks for security first.

   But there are some points and areas where one has take extra care while developing stand-alone desktop applications using Java technology. As per the above discussion we have understood that Java can be suitably used for desktop application development.

## 12.10 AWT Components

The AWT provides nine basic non-container component classes from which a user interface may be constructed. These nine classes are class Button, Canvas, Checkbox, Choice, Label, List, Scrollbar, TextArea, and TextField.



1. **Button**

   Buttons are simple GUI components used to generate events when the user presses them. Buttons are components that can be placed on a container. The Button sends java.awt.event.ActionEvent objects to its listeners when it is pressed. An event is generated when the mouse clicks on a Button. The implementation of actionPerformed (ActionEvent e) describes the response to the button press.

**Example 10: Program to GUI form using button controll and textfield.**

```
import java.awt.*;
import java.awt.event.*;
class MyFrame9 extends Frame implements ActionListener

{
  Label l1,l2,l3;
  TextField tf1,tf2,tf3;
  Button b1,b2;
  MyFrame9()
  {
    setLayout(new GridLayout(4,2));
    l1=new Label("No.1");
    l2=new Label("No.2");
    l3=new Label("Result");
    tf1=new TextField();
    tf2=new TextField();
    tf3=new TextField();
    b1=new Button("Sum");
    b2=new Button("Avg");
    b1.addActionListener(this);
    b2.addActionListener(this);
    add(l1);add(tf1);
    add(l2);add(tf2);
    add(l3);add(tf3);
    add(b1);add(b2);
    setSize(400,400);
    setVisible(true);
  }
  public static void main(String args[])
  {
    new MyFrame9();
  }
  public void actionPerformed(ActionEvent ae)
  {
    int x=Integer.parseInt(tf1.getText());
    int y=Integer.parseInt(tf2.getText());
    Button b=(Button)ae.getSource();
    if(b==b1)
    tf3.setText(x+y+"");
    else
    tf3.setText((x+y)/2.0f+"");
  }

}
```

**Output:**


**Constructors and Methods**

1. **public Button():** Description Constructs a Button object with no label.
2. **public Button (String label):** Parameters label The text for the label on the button Description Constructs a Button object with text of label.
3. **public void addActionListener (ActionListener l):** Description Add a listener for the action event.
4. **public String getActionCommand():** It Returns Current action command string used for the action command.
5. **public String getLabel():** Returns Text of the Button's label.
6. **public void removeActionListener (ActionListener l):** Remove an action event listener.
7. **public void setActionCommand (String command):** Specify the string used for the action command.

2. Textfield

The TextField class provides a single line Component for user input. Text fields allow the user to enter text that can be processed by the program. One constructor takes an *int* representing the width of the field in characters (approximately). Others take an initial string as a parameter or both.

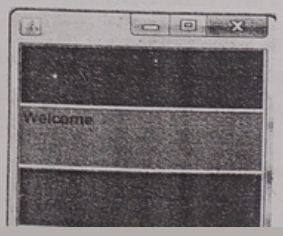
1. **public TextField():** It constructs a TextField object of the default size.
2. **public TextField (int columns):** It Constructs a TextField object of the given size.
3. **public TextField (String text):** It constructs a TextField object with the given content.
4. **public TextField (String text, int columns):** It constructs a TextField object with the given content and size.
5. **addActionListener:** It add a listener for the action event.
6. **echoCharIsSet:** It returns true if the TextField has an echo character used as a response to any input character, false other wise. An echo character can be used to create a TextField for hidden input.
7. **getEchoChar:** It returns the current echo character.
8. **removeActionListener:** It remove an action event listener.
9. **setColumns:** ItChanges the number of columns.



**Example 11: Program using textfield with textfield Listener.**

```
import java.awt.*;
import java.awt.event.*;
class MyFrame10 extends Frame implements FocusListener
{
 TextField tf1,tf2,tf3;
 MyFrame10()
 {
  setLayout(new GridLayout(3,1));
  tf1=new TextField();
  tf2=new TextField();
  tf3=new TextField();
  tf1.setBackground(Color.blue);
  tf2.setBackground(Color.blue);
  tf3.setBackground(Color.blue);
  tf1.addFocusListener(this);
  tf2.addFocusListener(this);
  tf3.addFocusListener(this);
  add(tf1);add(tf2);add(tf3);
  setSize(200,200);
  setVisible(true);
 }
 public static void main(String args[])
 {
  new MyFrame10();
 }
 public void focusLost(FocusEvent fe)
 {
  TextField tf=(TextField)fe.getSource();
  tf.setBackground(Color.blue);
 }
 public void focusGained(FocusEvent fe)
 {
  TextField tf=(TextField)fe.getSource();
  tf.setBackground(Color.red);
 }
}
```

**Output:**



### 3. CheckBox and RadioButton

The Checkbox is a Component that provides a true or false toggle switch for user input.

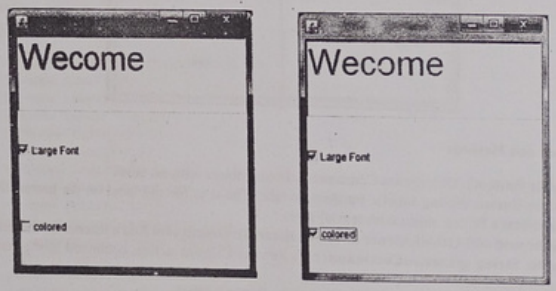
1. **public Checkbox():** It constructs a Checkbox object with no label that is initially false.
2. **public Checkbox (String label):** It constructs a Checkbox object with the given label that is initially false.
3. **public Checkbox (String label, boolean state):** It constructs a Checkbox with the given label, initialized to the given state.
4. **public Checkbox (String label, boolean state, CheckboxGroup group):** It constructs a Checkbox with the given label, initialized to the given state and belonging to group.
5. **addItemListener:** It adds a listener for the ItemEvent objects this Checkbox generates.
6. **getLabel:** It returns the text associated with the Checkbox.
7. **getState:** It returns The current state of the Checkbox.
8. **removeItemListener:** It removes the specified ItemListener so it will not receive ItemEvent objects from this Checkbox.
9. **setLabel:** It changes the text associated with the Checkbox.
10. **setState:** It changes the state of the Checkbox.

**Example 12: Program using chekbox and checkbok listener (Itermlistener).**

```
import java.awt.*;
import java.awt.event.*;
class MyFrame11 extends Frame implements ItemListener
{
  TextField tf1;
  Checkbox c1,c2;
  MyFrame11()
  {
    setLayout(new GridLayout(3,1));
    tf1=new TextField();
    c1=new Checkbox("Large Font");
    c2=new Checkbox("colored");
    c1.addItemListener(this);
    c2.addItemListener(this);
    add(tf1);add(c1);add(c2);
    setSize(300,300);
    setVisible(true);
  }
  public static void main(String args[])
  {
    new MyFrame11();
  }
  public void itemStateChanged(ItemEvent ie)
  {  Font f1;
    Checkbox c=(Checkbox)ie.getSource();
    if(c==c1)
    {
      if(c1.getState()==true)
      f1=new Font("Arial",Font.PLAIN,40);
      else
      f1=new Font("Arial",Font.PLAIN,10);
      tf1.setFont(f1);
    }
    else
    {
      if(c2.getState())
      tf1.setForeground(Color.RED);
      else
      tf1.setForeground(Color.BLACK);
    }
  }
}
```



**Output:**

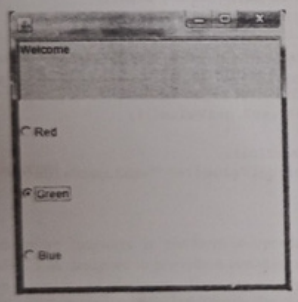



**Example 13: Program using RadioButton and Radiobutton listener (Itemlistner).**

```
import java.awt.*;
import java.awt.event.*;
class MyFrame12 extends Frame implements ItemListener
{
  TextField tf1;
  Checkbox c1,c2,c3;
  CheckboxGroup cbg;
  MyFrame12()
  {
    setLayout(new GridLayout(4,1));
    tf1=new TextField();
    //tf1.setBackground(Color.green);
    cbg=new CheckboxGroup();
    c1=new Checkbox("Red",cbg,false);
    c2=new Checkbox("Green",cbg,true);
    c3=new Checkbox("Blue",cbg,false);
    c1.addItemListener(this);
    c2.addItemListener(this);
```

```
    c3.addItemListener(this);
    add(tf1);add(c1);add(c2);add(c3);
    setSize(300,300);setVisible(true);
  }
  public static void main(String args[])
  {
    new MyFrame12();
  }
  public void itemStateChanged(ItemEvent ie)
  { if(c1.getState())
    tf1.setBackground(Color.red);
    else if(c2.getState())
    tf1.setBackground(Color.green);
    else if(c3.getState())
    tf1.setBackground(Color.blue);

  }
}
```

**Output:**




### 4. Dropdown List and ListBox

The Choice is a Component that provides a drop-down list of choices to choose from *Choice*

1. **public Choice():** It constructs a Choice object.
2. **public synchronized void add (String item):** It adds a new entry to the available choices.
3. **public synchronized void addItem (String item):** It replaced by add(String).
4. **public void addItemListener (ItemListener l):** It adds a listener for the ItemEvent objects this Choice generates.
5. **public int countItems():** It returns Number of items in the Choice.
6. **public String getItem (int index):** It returns a string for an entry at a given position. It throws ArrayIndexOutOfBoundsException If index is invalid, indices start at zero.



7. **public int getItemCount():** It returns Number of items in the Choice.
8. **public int getSelectedIndex():** It returns Position of currently selected entry.
9. **public synchronized String getSelectedItem():** It returns Currently selected entry as a String.
10. **public synchronized void insert (String item, int index):** It inserts item in the given position.
11. **public synchronized void remove (int position):** It removes the entry in the given position.
12. **public synchronized void remove (String string):** It makes the first entry that matches string the selected item.
13. **public synchronized void removeAll():** It removes all the entries from the Choice.
14. **public void removeItemListener (ItemListener l):** It Removes the specified ItemListener so it will not receive ItemEvent objects from this Choice.

**Example 14: Program using Dropdown list and its listener (i.e Itemlistener)**

```
import java.awt.*;
import java.awt.event.*;
class MyFrame13 extends Frame implements ItemListener
{
  TextField tf1;
  Choice c1;
  //List c1;
  MyFrame13()
  {
    setLayout(new GridLayout(2,1));
    tf1=new TextField();
    c1=new Choice();
    //c1=new List();
    c1.add("red");
    c1.add("green");
    c1.add("blue");
    c1.addItemListener(this);
    add(tf1);add(c1);
    setSize(300,300);setVisible(true);
  }
  public static void main(String args[])
  {
    new MyFrame13();
  }
  public void itemStateChanged(ItemEvent ie)
  {
    String s=c1.getSelectedItem();
    if(s.equals("red"))
    tf1.setBackground(Color.red);
    else if(s.equals("green"))
    tf1.setBackground(Color.green);
    else if(s.equals("blue"))
    tf1.setBackground(Color.blue);
  }
}
```

**Output:**


### 5. Scrollbar

The Scrollbar is a Component that provides the means to get and set values within a predetermined range. Scrollbars are most frequently used to help users manipulate areas too large to be displayed on the screen or to set a value within an integer range.

**Horizontal:** It is public final static and give a int value. It is constant used for a Scrollbar with a horizontal orientation.

**Vertical:** It is public final static and give a int value. It is a constant used for a Scrollbar with a vertical orientation.

**Constructors and Methods:**

1. **public Scrollbar():** It constructs a vertical Scrollbar object, slider size, minimum value, maximum value, and initial value are all zero.
2. **public Scrollbar (int orientation):** It constructs a Scrollbar object, in the designated direction, slider size, minimum value, maximum value, and initial value are all zero.
3. **public Scrollbar (int orientation, int value, int visible, int minimum, int maximum):** It constructs a Scrollbar object with the given values.
4. **public void addAdjustmentListener (AdjustmentListener l):** It add a listener for adjustment event.
5. **public int getValue():** It returns The current setting for the Scrollbar.
6. **removeAdjustmentListener:** It remove an adjustment event listener.
7. **public synchronized void setValue (int value):** It changes the current value of the Scrollbar.
8. **public synchronized void setValues (int value, int visible, int minimum, int maximum):** It changes the settings of the Scrollbar to the given amounts.

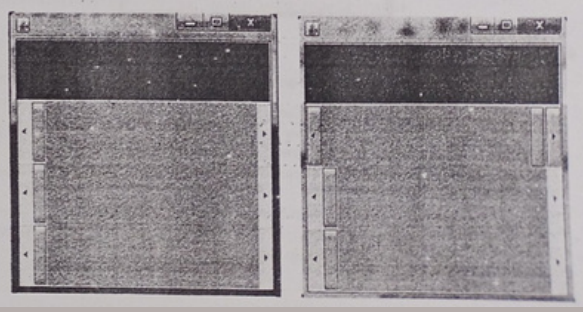
**Example 15: Program using scrollbar and its listener (i.e. Adjustment listner).**

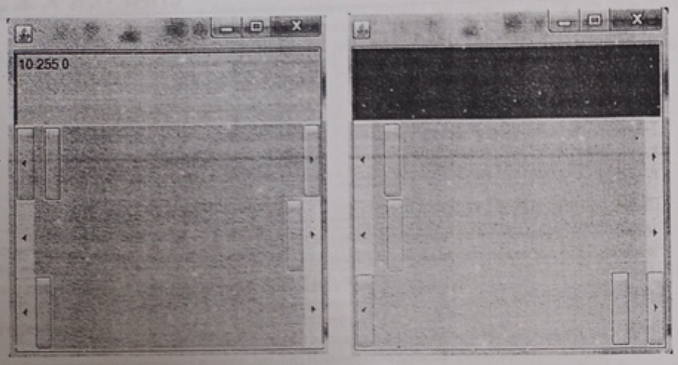
```
import java.awt.*;
import java.awt.event.*;
class MyFrame16 extends Frame implements AdjustmentListener,
```



```
{
  TextField tf1;
  Scrollbar sb1,sb2,sb3;
  MyFrame16()
  {
    setLayout(new GridLayout(4,1));
    tf1=new TextField("0");
    tf1.setBackground(Color.black);
sb1=new Scrollbar(Scrollbar.HORIZONTAL,0,1,0,256);
sb2=new Scrollbar(Scrollbar.HORIZONTAL,0,1,0,256);
sb3=new Scrollbar(Scrollbar.HORIZONTAL,0,1,0,256);
    sb1.addAdjustmentListener(this);
    sb2.addAdjustmentListener(this);
    sb3.addAdjustmentListener(this);
    add(tf1);add(sb1);
    add(sb2);add(sb3);
    setSize(300,300);
    setVisible(true);
  }
  public static void main(String args[])
  { new MyFrame16();
  }
  public void adjustmentValueChanged(AdjustmentEvent ae)
  {
    Color c1=new Color(sb1.getValue(),
      sb2.getValue(),sb3.getValue());

    tf1.setBackground(c1);
    tf1.setText(sb1.getValue()+" "+sb2.getValue()+" "+sb3.getValue()+" ");
  }
}
```

**Output:**






## 12.11 Swing

Based on the Abstract Windowing Toolkit, AWT, found in the package java.awt. AWT components called heavyweight components and implemented with native code (probably C++) written for the particular computer.

**Disadvantages of AWT:**

- Not uniform between platforms.
- Not very fast.
- Not very flexible.

Because the Java programming language is platform-independent, the AWT must also be platform-independent. The AWT was designed to provide a common set of tools for graphical user interface design that work on a variety of platforms. The user interface elements provided by the AWT are implemented using each platform's native GUI toolkit, thereby preserving the look and feel of each platform. This is one of the AWT's strongest points. The disadvantage of such an approach is the fact that a graphical user interface designed on one platform may look different when displayed on another platform.

Swing is actually part of a larger family of Java products known as the Java Foundation Classes (JFC), which incorporate many of the features of Netscape's Internet Foundation Classes (IFC) as well as design aspects from IBM's Taligent division and Lighthouse Design. Swing comes in late 1996. Swing has been in active development since the beta period of the Java Development Kit ( JDK) 1.1. The Swing APIs entered beta in the latter half of 1997 and were initially released in March 1998. When released, the Swing 1.0 libraries contained nearly 250 classes and 80 interfaces.

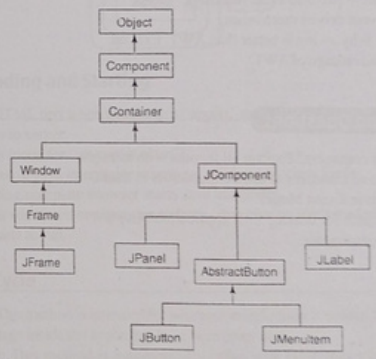
Version 1.2 of Java has extended the AWT with the Swing Set which consists of lightweight components that can be drawn directly onto containers using code written in Java.



**Comparison of AWT and Swing**

| java.awt | javax.swing |
|---|---|
| Frame | JFrame |
| Panel | JPanel |
| Label | JLabel |
| Button | JButton |
| TextField | JTextField |
| Checkbox | JCheckBox |
| List | JList |
| Choice | JComboBox |

**Swing Classes Hierarchy:**




Is Swing a Replacement for AWT?

No, Swing is actually built on top of the core AWT libraries. Because Swing does not contain any platform-specific (native) code, it can deploy the Swing distribution on any platform that implements the Java 1.1.5 or above virtual machine. In fact if you have JDK 1.2 or higher version, then the Swing classes are already available.

### Very Short Questions

1. What is default layout of Frame?
2. What is default layout of Panel?

UNIT • III