MODULE IV

1. STRING MANIPULATIONS

1.1 Introduction:

- **Creating empty string:** To create an empty String, we need to call the default constructor.

```
String s = new String();
```

- **Creating string initialized by an array of characters:** To create a String initialized by an array of characters, use the constructor **String(char chars[ ])**

```
char chars[] = { 'a', 'b', 'c' };
String s = new String(chars); // s is initialized with abc
```

- **Creating string initialized with a subrange of a character array:** To create a String, initialized with a subrange of a character array use the constructor **String(char chars[ ], int startIndex, int numChars)**

```
char chars[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
String s = new String(chars, 2, 3); // s is initialized with cde
```

- **Creating string object that contains the same character sequence as another String object:** To create a String object that contains the same character sequence as another String object use the constructor **String(String strObj)**

```
char c[] = {'J', 'a', 'v', 'a'};
String s1 = new String(c); // s1 is initialized with Java
String s2 = new String(s1); // s2 is initialized with Java
```

- **Creating string from byte array:** To create a String, from a byte array use the following two constructors:

  String(byte asciiChars[ ])

  String(byte asciiChars[ ], int startIndex, int numChars)

  Example:

```
byte ascii[] = {65, 66, 67, 68, 69, 70 };
String s1 = new String(ascii); //s1 contains ABCDEF
String s2 = new String(ascii, 2, 3); //s2 contains CDE
```

- **Creating string using string literal:** We can use a string literal to initialize a String object.

```
String s2 = "abc";
```

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

```
String s1="Welcome";
String s2="Welcome";//will not create new instance
```

  We can use a string literal any place we can use a String object. We can call methods directly on a quoted string as if it were an object reference.

```
System.out.println("abc".length());
```

- **Finding string length:** To find the length of a String, we call the **length()** method of **String** class.

```
char chars[] = { 'a', 'b', 'c' };
String s = new String(chars);
System.out.println(s.length()); // Output: 3
```

- **String concatenation:**

  In java, string concatenation forms a new string *that is* the combination of multiple strings. There are two ways to concat string in java:

  1. By + operator
  2. By concat() method

1. String Concatenation by + operator

- We can use '+' operator for concatenation of multiple strings.

```
String age = "9";
```

```
String s = "He is " + age + " years old.";
System.out.println(s); // Output: “He is 9 years old.
```

- We can concatenate strings with other types of data.

```
int age = 9;
String s = "He is " + age + " years old.";
System.out.println(s); // Output: “He is 9 years old.
```

Here, the int value in *age* is automatically converted into its string representation within a String object. This string is then concatenated as before.

NOTE: The compiler will convert an operand to its string equivalent whenever the other operand of the + is an instance of String.

- We should be careful when we mix other types of operations with string concatenation expression.

```
String s1 = "four: " + 2 + 2;
System.out.println(s1); // Output: four: 22

String s2 = "four: " + (2 + 2);
System.out.println(s2); // Output: four: 4

String s3 = 2 + 2 + ": four";
System.out.println(s3); // Output: 4: four
```

2. **String Concatenation by concat( ) method**

- The String concat() method concatenates the specified string to the end of current string.

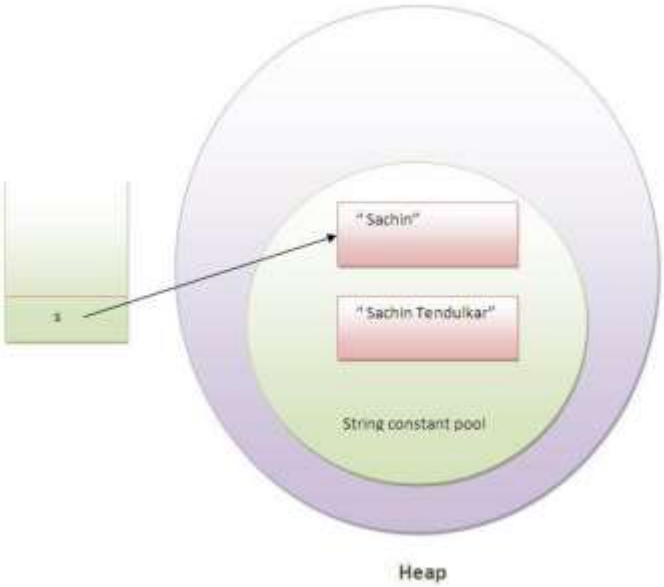
Syntax:public String concat(String another)

```
class TestStringConcat{
 public static void main(String args[]){
   String s1="Sachin ";
   String s2="Tendulkar";
   String s3=s1.concat(s2);
   System.out.println(s3);//Sachin Tendulkar
  }
}
```

NOTE:

- In Java, string objects are immutable (unmodifiable or unchangeable). Once string object is created its data or state can't be changed but a new string object is created.

```
String s="Sachin";
s.concat(" Tendulkar");
System.out.println(s);
            //print Sachin because strings are immutable objects
```



- But if we explicitly assign it to the reference variable, it will refer to "Sachin Tendulkar" object.

```
String s="Sachin";
s=s.concat(" Tendulkar");
System.out.println(s);//print Sachin Tendulkar
```

- **String Conversion and toString( ):**
  Every class implements toString( ) method because it is defined by Object class.Formostimportantclassesthatyou create, youwillwanttooverridetoString()andprovideyourownstringrepresentations.The toString( ) method has this general form:**String toString( )**.

```
class Box
{
      double width;
      double height;
      double depth;
      Box(double w, double h, double d)
      {
            width = w;
            height = h;
            depth = d;
      }
     public String toString()
     {
            return "Dimensions are " + width + " by " + depth + " by "
                    + height + ".";
      }
}
class toStringDemo
{
     public static void main(String args[])
     {
           Box b = new Box(10, 12, 14);
           String s = "Box b: " + b; // concatenate Box object
           System.out.println(b);    //convert    Box    to    string
           System.out.println(s);
      }
}
```

Output:

Dimensions are 10.0 by 14.0 by 12.0

Box b: Dimensions are 10.0 by 14.0 by 12.0

NOTE: Box's toString( ) method is automatically invoked when a Box object is used in a concatenation expression or in a call to println( ).

- **Character Extraction:**
  1. **char charAt(int i):** Returns the character at i^{th} index.

```
String s=new String("INDIA");
System.out.println(s.charAt(2));     // prints  'D'
```

  2. **void getChars(int sourceStart, int sourceEnd, char target[ ], int targetStart):** Stores the character from index **sourceStart** to index **sourceEnd**from the source string at the target string starting at **targetStart** index.

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

```
        {
                String s = "This is a demo of the getChars method.";
                int start = 10,end = 14;
                char buf[] = new char[end - start];
                s.getChars(start, end, buf, 0);
                System.out.println(buf);// prints  demo
        }
}
```

3. **String substring (int i):** Return the substring from the i^{th} index character to end.
   "GeeksforGeeks".substring(3); // returns "ksforGeeks"
4. **String substring (int i, int j):** Returns the substring from i to j-1 index.
   "GeeksforGeeks".substring(2, 5); // returns "eks"

- **String Comparison:**
  1. **boolean equals(Object str):** Here**str** is the String object being compared with the invoking String object.
     Boolean out = "Geeks".equals("Geeks"); // returns true
     Boolean out = "Geeks".equals("geeks"); // returns false
  2. **boolean equalsIgnoreCase (String anotherString):** Compares invoking String objectwith another string, ignoring case considerations.
     Boolean out= "Geeks".equalsIgnoreCase("Geeks"); // returns true
     Boolean out = "Geeks".equalsIgnoreCase("geeks"); // returns true
  3. **boolean regionMatches(int startIndex, String str2, int str2StartIndex, int numChars):** Comparesaspecificregioninsideastringwithanotherspecific region in another string.
  4. **boolean regionMatches(boolean ignoreCase, int startIndex, String str2, int str2StartIndex, int numChars):** Overloaded version of the previous function. If **ignoreCase** is true, the case of the characters is ignored. Otherwise, case is significant.
- **boolean startsWith(String str):** The startsWith( ) method determines whether a given String begins with a specified string.

```
"Foobar".startsWith("Foo");        // returns true
```

- **boolean endsWith(String str):** The, endsWith( ) determines whether the String in question ends with a specified string.

```
"Foobar".endsWith("bar");          // returns true
```

- **equals( ) Versus ==:** The equals( ) method compares the characters inside a String object. The == operator compares two object references to see whether they refer to the same instance.

```
String s1 = "Hello";
String s2 = new String(s1);
System.out.println(s1.equals(s2));      // prints true
System.out.println((s1 == s2));         // prints false
```

- **int compareTo(String str):** This method compares String strwiththeinvoking String.Theresultofthecomparison is returned and is interpreted, as shown here:

Value	Meaning
Less than zero	The invoking string is less than *str*.
Greater than zero	The invoking string is greater than *str*.
Zero	The two strings are equal.

Assignment: Write a Java Program to sort names of a group of students.

- **Searching Strings:**The String class provides the following methods that allow us to search a string for a specified character or substring:
  int indexOf(int ch): Searches the first occurrence of a character ch.

int lastIndexOf(int ch):Searches the last occurrence of a character ch.
int indexOf(String str): Searches the first occurrence of the string str.
int lastIndexOf(String str):Searches the last occurrence of the string str.
int indexOf(int ch, int startIndex): Searches the first occurrence of a character ch starting from the index startIndex.
int lastIndexOf(int ch, int startIndex):Searches the last occurrence of a character ch starting from the index startIndex.
int indexOf(String str, int startIndex): Searches the first occurrence of the string str starting from the index startIndex.
int lastIndexOf(String str, int startIndex):Searches the last occurrence of the string str starting from the index startIndex.

- **Modifying a String:**The replace( ) method has the following two forms:
  - It replaces all occurrences of one character in the invoking string with another character. It has the following general form:

    String replace(char original, char replacement)

    Example:

```
String s1="javatpoint is a very good website";
String s2=s1.replace('a','e');//replaces all occurrences of 'a' to 'e'
```

  - It replaces one character sequence with another. It has the following general form:

    String replace(CharSequence original, CharSequence replacement)

    Example:

```
String s1="my name is khan my name is java";
String s2=s1.replace("is","was");//replaces all occurrences of "is" to
                                        "was"
```

- **trim():**Thetrim()methodreturnsacopyoftheinvokingstringfromwhichanyleadingandtrailing whitespace has been removed.

```
String s1="  hello string   ";
System.out.println(s1+"java");//Prints hello string   java
System.out.println(s1.trim()+"java");//Prints hello stringjava
```

- **valueOf():**This method converts different types of values into string. By the help of string valueOf() method, we can convert int to string, long to string, boolean to string, character to string, float to string, double to string, object to string and char array to string.The signature or syntax of String valueOf() method is given below:

  public static String valueOf(boolean b)
  public static String valueOf(char c)
  public static String valueOf(char[] c)
  public static String valueOf(int i)
  public static String valueOf(long l)
  public static String valueOf(float f)
  public static String valueOf(double d)
  public static String valueOf(Object o)

  Example:

```
int value=30;
String s1=String.valueOf(value);
System.out.println(s1+10);//Prints 3010
```

  - **Changing the case of characters within a string:** The method toLowerCase() converts all the characters in a string from uppercase to lowercase.ThetoUpperCase()methodconvertsallthecharactersinastringfromlowercase

touppercase.Non-alphabeticalcharacters,suchasdigits,areunaffected. The following arethegeneral forms of these methods:

String toLowerCase( )

String toUpperCase( )

Example:

```
String s = "This is a test.";
System.out.println(s);  //Prints This is a test.
String upper = s.toUpperCase();
String lower = s.toLowerCase();
System.out.println(upper); //Prints THIS IS A TEST.
System.out.println(lower);  //Prints this is a test.
```

1.2 StringBuffer:

StringBuffer is a peer class of String that provides much of the functionality of strings. String represents fixed-length, immutable character sequences. In contrast, StringBufferrepresentsgrowableandwriteablecharactersequences. StringBuffermayhave characters and substrings inserted in the middle or appended to the end. StringBuffer will automaticallygrowtomakeroomforsuchadditionsandoftenhasmorecharacterspre-allocated thanareactuallyneeded,toallowroomforgrowth.

- **StringBuffer Constructors:**StringBuffer defines the following four constructors:
  StringBuffer( ): This constructs a string buffer with no characters in it and an initial capacity of 16 characters.
  StringBuffer(int size): This constructs a string buffer with no characters in it and the specified initial capacity.
  StringBuffer(String str): Accepts a String argument that sets the initial contents of the StringBuffer object and reserves room for 16 more characters.
  StringBuffer(CharSequence chars): Creates an object that contains the character sequence contained in chars.
- **length( ) and capacity( ):**The length of a StringBuffer can be found by the length( ) method, while the total allocated capacity can be found by the capacity( ) method.

```
StringBuffer sb = new StringBuffer("Hello");
System.out.println("length = " + sb.length()); // Prints 5
System.out.println("capacity = " + sb.capacity()); // Prints 21
```

- **ensureCapacity(int minimumCapacity):** Thismethod ensures that the capacity is at least equal to the specified minimum. If the current capacity is less than the argument, then a new internal array is allocated with greater capacity. The new capacity is the larger of:
  - ✓ The **minimumCapacity** argument.
  - ✓ Twice the old capacity, plus 2.

Example:

```
import java.lang.*;
public class StringBufferDemo
{

      public static void main(String[] args)
      {
            StringBuffer buff1 = new StringBuffer("tuts point");
            System.out.println("buffer1 = " + buff1);

            // returns the current capacity of the string buffer 1
            System.out.println("Old Capacity = " + buff1.capacity());

            // returns twice the capacity plus 2
            buff1.ensureCapacity(28);
```

```
                System.out.println("New Capacity = " + buff1.capacity());

                StringBuffer buff2 = new StringBuffer("compile online");
                System.out.println("buffer2 = " + buff2);

                // returns the current capacity of string buffer 2
                System.out.println("Old Capacity = " + buff2.capacity());

                buff2.ensureCapacity(29);
                System.out.println("New Capacity = " + buff2.capacity());
        }
    }
```

Output:

```
buffer1 = tuts point
Old Capacity = 26
New Capacity = 54
buffer2 = compile online
Old Capacity = 30
New Capacity = 30
```

- **void setLength(int len):** method sets the length of the character sequence. The sequence is changed to a new character sequence whose length is specified by the argument. If the **newLength** argument is greater than or equal to the current length, sufficient null characters ('\u0000') are appended so that length becomes the **newLength** argument.

Example:

```
import java.lang.*;
public class StringBufferDemo
{
   public static void main(String[] args)
   {
            StringBuffer buff = new StringBuffer("tutorials");
            System.out.println("buffer1 = " + buff);

            System.out.println("length = " + buff.length());
            buff.setLength(5);
            System.out.println("buffer2 = " + buff);
            System.out.println("length = " + buff.length());
   }
}
```

Output:

```
buffer1 = tutorials
length = 9
buffer2 = tutor
length = 5
```

- **char charAt(int index):** Thismethod returns the char value in this sequence at the specified **index**.

Example:

```
        StringBuffer buff = new StringBuffer("Tutorials Point");
        System.out.println(buff.charAt(4)); //Prints r
```

- **void setCharAt(int index, char ch):** Thismethod sets the character at the specified **index** to **ch**.

  Example:

```
StringBuffer buff = new StringBuffer("AMIT");
System.out.println(buff); //Prints AMIT
buff.setCharAt(3, 'L');   //Prints AMIL
System.out.println(buff);
```

- **void getChars(int srcBegin, int srcEnd, char target[ ], int targetBegin):** Thismethod copy the characters from this sequence into the destination character array **target**. The first character to be copied is at index **srcBegin**. The last character to be copied is at index **srcEnd - 1**. The total number of characters to be copied is **srcEnd - srcBegin**. The characters are copied into the subarray of **target** starting at index **targetBegin** and ending at index: **targetbegin + (srcEnd-srcBegin) - 1**

  Example:

```
StringBuffer buff = new StringBuffer("java programming");
System.out.println(buff); //Prints java programming
char[] chArr = new char[]{'t','u','t','o','r','i','a','l','s'};
buff.getChars(5, 10, chArr, 3);
System.out.println(chArr);  // Prints tutprogrs
```

- **append():**

  Theappend()methodconcatenatesthestringrepresentationofanyothertypeofdatatothe end of the invoking StringBuffer object. It has several overloaded versions. Some of these forms are:

  StringBuffer append(String str)
  StringBuffer append(int num)
  StringBuffer append(Object obj)

  String.valueOf( ) is called for each parameter to obtain its string representation. The result is appended to the current StringBuffer object.

  Example:

```
StringBuffer s;
int a = 42;
StringBuffer sb = new StringBuffer(40);
s = sb.append("a = ").append(a).append("!");
System.out.println(s);  // Prints a=42!
```

- **insert( ):** This method inserts the data into a substring of this StringBuffer. We should specify the offset value (integer type) of the buffer, at which we need to insert the data. Using this method, data of various types like integer, character, string etc. can be inserted. Java provides separate method for each primitive data type:

  public StringBuffer insert(int offset, boolean b)
  public StringBuffer insert(int offset, char c)
  public insert(int offset, char[] str)
  public StringBuffer insert(int index, char[] str, int offset, int len)
  public StringBuffer insert(int offset, float f)
  public StringBuffer insert(int offset, int i)
  public StringBuffer insert(int offset, long l)
  public StringBuffer insert(int offset, Object obj)
  public StringBuffer insert(int offset, String str)

  ItcallsString.valueOf() to obtain the string representation of the value it is called with. This string is then inserted into the invoking StringBuffer object.

  Example:

```
StringBuffer sb = new StringBuffer("I Java!");
```

```
sb.insert(2, "like ");
System.out.println(sb);  // Prints I like Java!
```

- **StringBuffer reverse ( ):** This method returns the reversed StringBuffer object on which it was called.

  Example:

```
StringBuffer s = new StringBuffer("abcdef");
System.out.println(s); // Prints abcdef
s.reverse();
System.out.println(s); // Prints fedcba
```

- **delete( ) and deleteCharAt( ):** Wecan delete characters within a StringBuffer by using the methods delete( ) and deleteCharAt( ). These methods are shown here:

  StringBuffer delete(int startIndex, int endIndex): Deletes the substring from startIndex to endIndex–1. The resulting StringBuffer object is returned.

  StringBuffer deleteCharAt(int loc): Deletes the character at the index specified by loc. It returns the resulting StringBuffer object.

  Example:

```
StringBuffer sb = new StringBuffer("This is a test.");
sb.delete(4, 7);
System.out.println(sb);//PrintsThis a test.
sb.deleteCharAt(0);
System.out.println(sb);//Printshis a test.
```

- **replace( ):** Wecan replace one set of characters with another set inside a StringBuffer object by calling replace( ). Its signature is shown here:

  StringBuffer replace(int startIndex, int endIndex, String str): The substring at**startIndex**through**endIndex–1**isreplaced by string**str**. The **resulting StringBuffer object is returned**.

  Example:

```
StringBuffer sb = new StringBuffer("This is a test.");
sb.replace(5, 7, "was");
System.out.println(sb);//Prints This was a test.
```

- **substring( ):** Wecan obtain a portion of a StringBuffer by calling substring( ). It has the following two forms:

  String substring(int startIndex): Returns the substring that starts at startIndex and runs to the end of the invoking StringBuffer object

  String substring(int startIndex, int endIndex): Returns the substring that starts at startIndex and runs through endIndex–1.

  Example:

```
StringBuffer buff = new StringBuffer("admin");
System.out.println(buff.substring(3));
System.out.println(buff.substring(1, 4));
```

1.3 StringBuilder:

It is identical to StringBuffer except for one important difference: it is not synchronized, which means that it is not thread-safe. The advantage of StringBuilderisfasterperformance.However, the casesinwhich we areusingmultithreading, you must use StringBuffer rather than StringBuilder.

- **Important StringBuilder Constructors:**

Constructor	Description
StringBuilder()	Creates an empty string Builder with the initial capacity of 16.

StringBuilder(String str)	Creates a string Builder with the specified string.
StringBuilder(int length)	Creates an empty string Builder with the specified capacity as length.

- **Important methods of StringBuilder Class:**

Method	Description
public StringBuilder append(String s)	Used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
public StringBuilder insert(int offset, String s)	Used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.
public StringBuilder replace(int startIndex, int endIndex, String str)	Used to replace the string from specified startIndex and endIndex.
public StringBuilder delete(int startIndex, int endIndex)	Used to delete the string from specified startIndex and endIndex.
public StringBuilder reverse()	Used to reverse the string.
public int capacity()	Used to return the current capacity.
public void ensureCapacity(int minimumCapacity)	Used to ensure the capacity at least equal to the given minimum.
public char charAt(int index)	Used to return the character at the specified position.
public int length()	Used to return the length of the string i.e. total number of characters.
public String substring(int beginIndex)	Used to return the substring from the specified beginIndex.
public String substring(int beginIndex, int endIndex)	Used to return the substring from the specified beginIndex and endIndex.

Difference between String and StringBuffer:

String	StringBuffer
String class is immutable.	StringBuffer class is mutable.
String is slow and consumes more memory when you concat too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when you concat strings.

String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.

Difference between StringBufferand StringBuilder:

StringBuffer	StringBuilder
StringBuffer is synchronized i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.	StringBuilder is non-synchronized i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.
StringBuffer is less efficient than StringBuilder.	StringBuilder is more efficient than StringBuffer.

1.4 StringTokenizer:

- During the processing of text sometime there is a need of parsing a formatted input string. Parsing is the division of text into a set of discrete parts, or tokens, which in a certain sequence can convey a semantic meaning.
- The StringTokenizer class provides the first step in this parsing process, often called the lexer (lexical analyzer) or scanner.
- To use StringTokenizer, we specify an input string and a string that contains delimiters. Delimiters are characters that separate tokens.
- Each character in the delimiters string is considered a valid delimiter—for example, ",;:" sets the delimiters to a comma, semicolon, and colon.
- The default set of delimiters consists of the whitespace characters: space, tab, newline,andcarriagereturn.
- The StringTokenizer constructors are shown here:
  StringTokenizer(String str): Default delimiters are used for parsing the string str.
  StringTokenizer(String str, String delim): String str is parsed using the delimiters provided in string delim.
  StringTokenizer(String str, String delim, boolean delimAsToken): Similar to the previous one. The only difference is if delimAsToken is true, then the delimiters are also returned as tokens when the string is parsed. Otherwise, the delimiters are not returned.
- Methods of StringTokenizer class:

Public method	Description
boolean hasMoreTokens()	Checks if there is more tokens available.
String nextToken()	Returns the next token from the StringTokenizer object.
String nextToken(String delim)	Returns the next token based on the delimeter.
boolean hasMoreElements()	Same as hasMoreTokens() method.
Object nextElement()	Same as nextToken() but its return type is Object.
int countTokens()	Returns the total number of tokens.

NOTE:

- SinceStringTokenizerimplementsEnumeration,thehasMoreElements()and nextElement()methodsarealsoimplemented,andtheyactthesameashasMoreTokens()and nextToken()respectively.

Example:

```
import java.util.StringTokenizer;
class STDemo
{
        static String in = "title=Java: The Complete Reference;" +
        "author=Schildt;"    +    "publisher=Osborne/McGraw-Hill;"    +
        "copyright=2007";
        public static void main(String args[])
        {
              StringTokenizer st = new StringTokenizer(in, "=;");
              while(st.hasMoreTokens())
              {
                    String key = st.nextToken();
                    String val = st.nextToken();
                    System.out.println(key + "\t" + val);
              }
        }
}
```

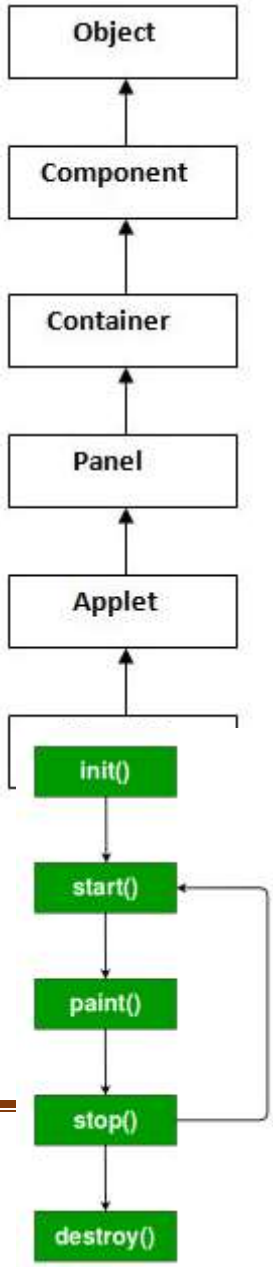
Output:

```
title  Java: The Complete Reference
author  Schildt
publisher  Osborne/McGraw-Hill
copyright  2007
```

2. APPLET

2.1 Introduction:

- Applet is a Java program that can be embedded into a web page.
- It runs inside the web browser and works at client side. Applet is embedded in a HTML page using the APPLET or OBJECT tag and hosted on a web server.
- Applets are used to make the web site more dynamic and entertaining.
- All applets are sub-classes (either directly or indirectly) of java.applet.Applet class.
- Applets are not stand-alone programs. Instead, they run within either a web browser or an applet viewer. JDK provides a standard applet viewer tool called **appletviewer**.
- There are some important differences between an applet and a standalone Java application, including the following:
  - A main() method is not invoked on an applet, and an applet class will not define main().
  - When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.
  - A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.
  - The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.
  - Applets have strict security rules that are enforced by the Web browser. The security of an applet is often referred to as sandbox security, comparing the applet to a child playing in a sandbox with various rules that must be followed.
  - Output of an applet window is not performed by System.out.println(). Rather it is handled with various AWT methods, such as drawString().
- **Advantages:**
  - It works at client side so less response time.
  - Secured
  - It can be executed by browsers running under many platforms, including Linux, Windows, Mac Os etc.
- **Drawback:**
  - Plugin is required at client browser to execute applet.
- **Hierarchy of Applet**: ----------→



2.2 Life cycle of an applet:

- The life cycle of applet pass through the following phases:
  - Applet is initialized.
  - Applet is started.
  - Applet is painted.
  - Applet is stopped.
  - Applet is destroyed.
- When an applet begins, the following methods are called, in this sequence:

1. **init( ):** The init( ) method is the first method to be called. This is where we should initialize variables. This method is called only once during the run time of the applet.
2. **start( ):** The start( ) method is called after init( ) method or browser is maximized. It is also called to restart an applet after it has been stopped. Note that init( ) is called once i.e. when the first time an applet is loaded whereas start( ) is called each time an applet's HTML document is displayed onscreen. So, if a user leaves a web page and comes back, the applet resumes execution at start( ).
3. **paint( ):** The paint( ) method is called each time an AWT-based applet's output must be redrawn. This situation can occur for several reasons. For example, the window in which the applet is running

may be overwritten by another window and then uncovered. Or the applet window may be minimized and then restored. paint( ) is also called when the applet begins execution. Whatever may be the cause, whenever the applet must redraw its output, the paint( ) method is called.

The paint( ) method has one parameter of type Graphics. This parameter will contain the graphics context, which describes the graphics environment in which the applet is running.

- When an applet is terminated, the following sequence of method calls takes place:
  1. **stop( ):** The stop( ) method to stop the Applet. It is invoked when Applet is to be stopped or browser is minimized. It is called when a web browser leaves the HTML document containing the applet—when it goes to another page, for example. When stop( ) is called, the applet is probably running. We should use stop( ) to suspend threads that don't need to run when the applet is not visible. We can restart them when start( ) is called if the user returns to the page.
  2. **destroy( ):**The destroy( ) method is called when the environment determines that our applet needs to be removed completely from memory. At this point, we should free up any resources the applet may be using. The stop( ) method is always called before destroy( ). It is invoked only once.
- **Note:** The **java.applet.Applet** class provides four life cycle methods init(), start(), stop(), destroy() and **java.awt.Component** class provides one life cycle method paint() for an applet.

2.3 Creating First applet:

- There are two ways to run an applet
  - By html file.
  - By appletviewer tool (for testing purpose).
- **Applet by html file:** To execute the applet by html file, create an applet and compileit. After that create an html file and place the applet code in html file. Now open the html file in a browser.

First.java

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

myapplet.html

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

Explanation:

- This applet class (**First**) must be declared as public, because it will be accessed by code that is outside the program. In fact the applet class object is created by Java Plugin software that resides on the browser.
- Inside First, paint( ) is declared. This method is defined by the AWT and must be overridden by the applet.
- Inside paint( ), there is a call to drawString( ), which is a member of the Graphics class. This method outputs a string beginning at the specified x, y location. It has the following general form:

```
void drawString(String message, int x, int y)
```

- Here, message is the string to be output beginning at x, y. In a Java window, the upper-left corner is location 0,0. The call to drawString( ) in the applet causes the message "Hello World" to be displayed beginning at location 150, 150.
- Note that the applet does not have a main( ) method. Unlike Java programs, applets do not begin execution at main( ). In fact, most applets don't even have a main( ) method. Instead, an applet begins execution when the name of its class is passed to an applet viewer or to a network browser.

- **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.

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
/*
<applet code="First.class" width="300" height="300">
</applet>
*/
public class First extends Applet
{

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

Output:

2.4 The Applet Class:

- **Applet** provides all necessary support for applet execution, such as starting and stopping. It also provides methods that load and display images, and methods that load and play audio clips.
- Applet extends the AWT class Panel. In turn, Panel extends Container, which extends Component. These classes provide support for Java's window-based, graphical interface. Thus, Applet provides all of the necessary support for window-based activities.
- Some of the methods of Applet class are given below:

Methods	Description
void init()	The first method to be called when an applet begins its execution.
void start()	Called automatically after **init()** method to start the execution of an applet.
void stop()	Called when an applet has to pause/stop its execution.
void destoy()	Called when an applet is finally terminated.
String getParameter(String ParameterName)	Returns the value of a parameter defined in an applet
Image getImage(URL *url*)	Returns an Image object that contains an image specified at location, *url*

void play(URL *url)	Plays an audio clip found at the specified at location, *url*
showStatus(String str)	Shows a message in the status window of the browser or appletviewer.

2.5 The Applet Architecture:

- An applet is a window-based program. Its architecture is different from the console-based programs.
- Applets are event driven.
- An applet resembles a set of interrupt service routines.
- An applet waits until an event occurs. The run-time system notifies the applet about an event by calling an event handler that has been provided by the applet. Once this happens, the applet must take appropriate action and then quickly return.
- The user initiates interaction with an applet. In a non-windowed program, when the program needs input, it will prompt the user and then call some input method, such as readLine( ). Instead in an applet, the user interacts with the applet as he or she wants. These interactions are sent to the applet as events to which the applet must respond.
    - **Example**: when the user clicks the mouse inside the applet's window, a mouse-clicked event is generated. If the user presses a key while the applet's window has input focus, a key press event is generated. Applets can contain various controls, such as push buttons and check boxes. When the user interacts with one of these controls, an event is generated.

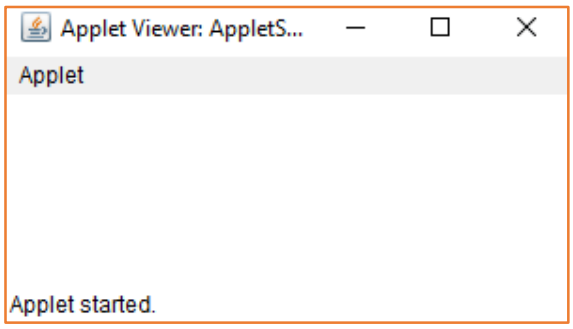
2.6 An Applet Skeleton:

- All but the most trivial applets override a set of methods that provides the basic mechanism by which the browser or applet viewer interfaces to the applet and controls its execution.
- Four of these methods, init( ), start( ), stop( ), and destroy( ), apply to all applets and are defined by Applet. Default implementations for all of these methods are provided.
- AWT-based applets will also override the paint( ) method, which is defined by the AWT Component class. This method is called when the applet's output must be redisplayed.
- These five methods can be assembled into the skeleton shown here:

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

      /*    Called second, after init().
            Also called whenever the applet is restarted. */
      public void start()
      {
            // start or resume execution
      }

      // Called when the applet is stopped.
      public void stop()
      {
            // suspends execution
      }

      /*    Called when applet is terminated.
            This is the last method executed. */
      public void destroy()
      {
            // perform shutdown activities
```

```
        }

        // Called when an applet's window must be restored.
        public void paint(Graphics g)
        {
              // redisplay contents of window
        }
}
```

Output:



2.7 Applet Display Methods:

- To output a string to an applet, use **drawString( )**, which is a member of the **Graphics** class. It is called from within either **update( )** or **paint( )**. It has the following general form:

```
void drawString(String message, int x, int y)
```

- To set the background and foreground color of an applet's window, use **setBackground( )** and **setForeground( )** methods respectively. These methods are defined by **Component**, and they have the following general forms:

```
void setBackground(Color newColor)
void setForeground(Color newColor)
```

Example:

```
setBackground(Color.green);
setForeground(Color.red);
```

Normally we set the foreground and background colors in the init( ) method.

- We can obtain the current settings for the background and foreground colors by calling getBackground( ) and getForeground( ), respectively. Theese methods are also defined by Component and are shown here:

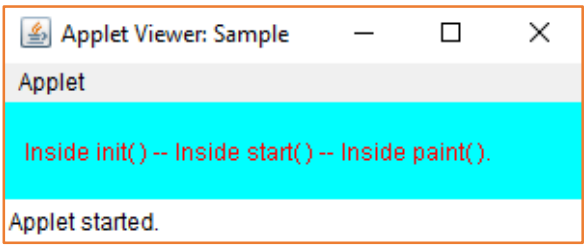
```
Color getBackground( )
Color getForeground( )
```

Example:

```
/* A simple applet that sets the foreground and background
colors and outputs a string. */
import java.awt.*;
import java.applet.*;

/* <applet code="Sample" width=300 height=50></applet> */
public class Sample extends Applet
{
      String msg;
      // set the foreground and background colors.
      public void init()
      {
            setBackground(Color.cyan);
            setForeground(Color.red);
            msg = "Inside init( ) --";
      }

      // Initialize the string to be displayed.
      public void start()
      {
            msg += " Inside start( ) --";
      }
```

```
        // Display msg in applet window.
        public void paint(Graphics g)
        {
            msg += " Inside paint( ).";
            g.drawString(msg, 10, 30);
        }
    }
```

Output:



2.8 Requesting Repainting:

- An applet writes to its window only when its update( ) or paint( ) method is called by the AWT.
- Whenever our applet needs to update the information displayed in its window, it simply calls repaint( ).
- The repaint( ) method is defined by the AWT. It causes the AWT run-time system to execute a call to your applet's update( ) method, which, in its default implementation, calls paint().
- The repaint() method has four forms:

  `void repaint( ):` This version causes the entire window to be repainted.

  `void repaint(int left, int top, int width, int height):` This version repaints a specified region.

Note: Calling repaint( ) is essentially a request that our applet be repainted sometime soon. However, if our system is slow or busy, update() might not be called immediately. Multiple requests for repainting that occur within a short time can be collapsed by the AWT in a manner such that update() is only called sporadically. To avoid this situations, we can use the following forms of repaint( ):

```
void repaint(long maxDelay)
void repaint(long maxDelay, int x, int y, int width, int height)
```

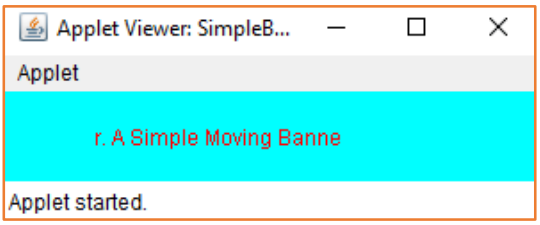
Here, maxDelay specifies the maximum number of milliseconds that can elapse before update() is called.

Example:

```
/* A simple banner applet.This applet creates a thread
that scrolls the message contained in msg right to left
across the applet's window.
*/
import java.awt.*;
import java.applet.*;
/*      <applet      code="SimpleBanner"      width=300
height=50></applet> */
public class SimpleBanner extends Applet implements
Runnable
{
    String msg = " A Simple Moving Banner.";
    Thread t = null;
    int state;
    boolean stopFlag;

    // Set colors and initialize thread.
    public void init()
    {
        setBackground(Color.cyan);
        setForeground(Color.red);
```

```
        }

        // Start thread
        public void start()
        {
            t = new Thread(this);
            stopFlag = false;
            t.start();
        }

        // Entry point for the thread that runs the banner.
        public void run()
        {
            char ch;
            // Display banner
            for( ; ; )
            {
                try
                {
                    repaint();
                    Thread.sleep(250);
                    ch = msg.charAt(0);
                    msg = msg.substring(1,
msg.length());
                    msg += ch;
                    if(stopFlag)
                        break;
                }
                catch(InterruptedException e) {}
            }
        }

        // Pause the banner.
        public void stop()
        {
            stopFlag = true;
            t = null;
        }

        // Display the banner.
        public void paint(Graphics g)
        {
            g.drawString(msg, 50, 30);
        }
    }
```

Output:



Explanation:

- The init( ) method initializes the foreground and background colors of the applet.
- The run-time system calls start( ) to start the applet running.
- Inside start( ) method, a new thread of execution is created and assigned to the Thread variable t. Then, the boolean variable stopFlag, which controls the execution of the applet, is set to false. Next, the thread is started by a call to t.start( ) which causes run( ) to begin executing. It does not cause a call to the version of start( ) defined by Applet. Note that these are two separate methods.

- Inside run( ), the characters in the string contained in msg are repeatedly rotated left. Between each rotation, a call to repaint( ) is made. This eventually causes the paint( ) method to be called, and the current contents of msg are displayed.
- Between each iteration, run( ) sleeps for a quarter of a second. The net effect of run( ) is that the contents of msg are scrolled right to left in a constantly moving display.
- The stopFlag variable is checked on each iteration. When it is true, the run( ) method terminates. If a browser is displaying the applet when a new page is viewed, the stop( ) method is called, which sets stopFlag to true, causing run( ) to terminate. This is the mechanism used to stop the thread when its page is no longer in view. When the applet is brought back into view, start( ) is once again called, which starts a new thread to execute the banner.

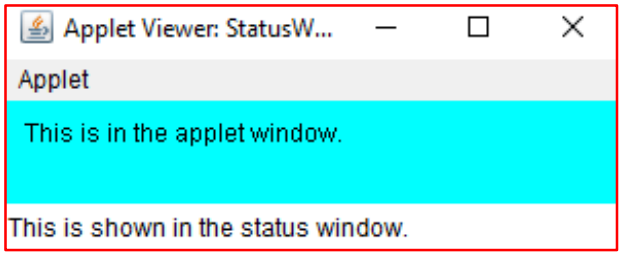
2.9 Using the Status Window:

- In addition to displaying information in its window, an applet can also output a message to the status window of the browser or applet viewer on which it is running. To do so, we need to call showStatus( ) with the string that we want to display.
- The following applet demonstrates showStatus( ):

Example:

```
// Using the Status Window.
import java.awt.*;
import java.applet.*;
/* <applet code="StatusWindow" width=300 height=50></applet> */
public class StatusWindow extends Applet
{
      public void init()
      {
            setBackground(Color.cyan);
      }
      // Display msg in applet window.
      public void paint(Graphics g)
      {
            g.drawString("This is in the applet window.", 10,
20);
            showStatus("This is shown in the status window.");
      }
}
```

Output:



2.10 The HTML APPLET Tag:

- Sun currently recommends that the APPLET tag be used to start an applet from both an HTMLdocument and from an applet viewer.
- The syntax for a fuller form of the APPLET tag is shown here. Bracketed items are optional.

```
< APPLET
[CODEBASE = codebaseURL]
CODE = appletFile
[ALT = alternateText]
[NAME = appletInstanceName]
WIDTH = pixels HEIGHT = pixels
[ALIGN = alignment]
[VSPACE = pixels]
```

[HSPACE = pixels]
>
[< PARAM NAME = AttributeName VALUE = AttributeValue>]
[< PARAM NAME = AttributeName2 VALUE = AttributeValue>]
. . .
[HTML Displayed in the absence of Java]
</APPLET>

- **CODEBASE:** (optional) specifies the base path of the applet's class. The default base path for the applet is the same path as the HTML file.
- **CODE:** Specifies the applet class name, with the ".class" extension..
- **ALT:** (optional) alternate text if the applet fails to be displayed
- **NAME:** (optional) specify a name for the applet instance. Applets must be named in order for other applets on the same page to find them by name and communicate with them.To obtain an applet by name, use **getApplet()**,which is defined by the **AppletContext** interface.
- **WIDTH and HEIGHT:** Specify the width and height of the applet's display area inside the browser..
- **ALIGN:** (optional) specifies the alignment of the applet. This attribute is treated the same as the HTMLIMG tag with these possible values: LEFT, RIGHT, TOP, BOTTOM, MIDDLE, BASELINE, TEXTTOP, ABSMIDDLE, and ABSBOTTOM.
- **VSPACE and HSPACE:** (optional) VSPACE specifies the space, in pixels, above and below the applet. HSPACE specifies the space, in pixels, on each side of the applet.
- **PARAM and VALUE:** (optional) The PARAM tag allows you to specify applet-specific arguments in an HTMLpage. Applets access their attributes with the getParameter( ) method.
- **ARCHIEVE:** (optional) specifies the JAR file that contains the applet classes. If your applet involves more than one classes, it is common and more efficient to jar (i.e., zip) them together into a single JAR file for distribution.

2.11 Passing Parameters to Applets:

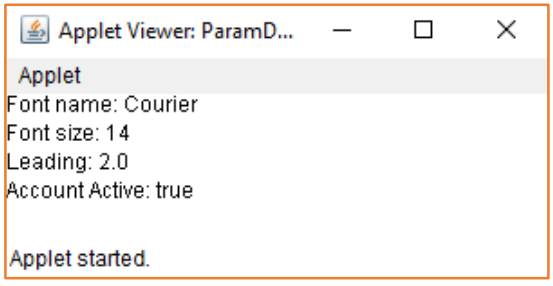
- The APPLET tag in HTML allows us to pass parameters to our applet.
- To retrieve a parameter, we can use the getParameter( ) method. It returns the value of the specified parameter in the form of a String object. Thus, for numeric and boolean values, you will need to convert their string representations into their internal formats.

Example:

```
// Use Parameters
import java.awt.*;
import java.applet.*;
/* <applet code="ParamDemo" width=300 height=80>
<param name=fontName value=Courier>
<param name=fontSize value=14>
<param name=leading value=2>
<param name=accountEnabled value=true>
</applet> */

public class ParamDemo extends Applet
{
      String fontName;
      int fontSize;
      float leading;
      boolean active;

      // Initialize the string to be displayed.
      public void start()
      {
            String param;
            fontName = getParameter("fontName");
```

```
                if(fontName == null)
                        fontName = "Not Found";
                param = getParameter("fontSize");
                try
                {
                        if(param != null) // if not found
                                fontSize = Integer.parseInt(param);
                        else
                                fontSize = 0;
                }
                catch(NumberFormatException e)
                {
                        fontSize = -1;
                }
                param = getParameter("leading");
                try
                {
                        if(param != null) // if not found
                        leading = Float.valueOf(param).floatValue();
                        else leading = 0;
                }
                catch(NumberFormatException e)
                {
                        leading = -1;
                }
                param = getParameter("accountEnabled");
                if(param != null)
                        active = Boolean.valueOf(param).booleanValue();
        }

        // Display parameters.
        public void paint(Graphics g)
        {
                g.drawString("Font name: " + fontName, 0, 10);
                g.drawString("Font size: " + fontSize, 0, 26);
                g.drawString("Leading: " + leading, 0, 42);
                g.drawString("Account Active: " + active, 0, 58);
        }
}
```

Output:



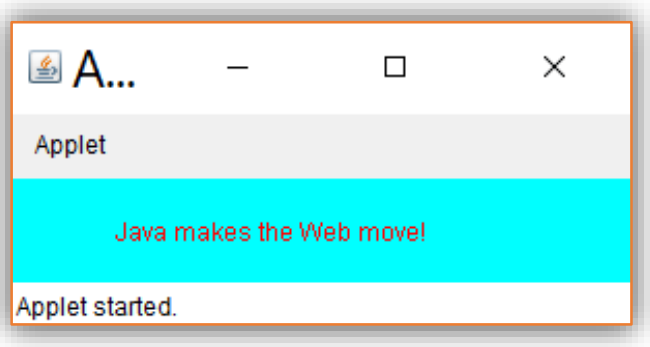
- In the following program we are passing the message as a parameter which allows the banner applet to display a different message each time it is executed.

Example:

```
// A parameterized banner
import java.awt.*;
import java.applet.*;
/*
<applet code="ParamBanner" width=300 height=50>
<param name=message value="Java makes the Web move!">
</applet>
*/
public class ParamBanner extends Applet implements Runnable
{
        String msg;
```

```
        Thread t = null;
        int state;
        boolean stopFlag;
        // Set colors and initialize thread.
        public void init()
        {
              setBackground(Color.cyan);
              setForeground(Color.red);
        }

        // Start thread
        public void start()
        {
              msg = getParameter("message");
              if(msg == null)
                    msg = "Message not found.";
              msg = " " + msg;
              t = new Thread(this);
              stopFlag = false;
              t.start();
        }

        // Entry point for the thread that runs the banner.
        public void run()
        {
              char ch;
              // Display banner
              for( ; ; )
              {
                    try
                    {
                          repaint();
                          Thread.sleep(250);
                          ch = msg.charAt(0);
                          msg = msg.substring(1, msg.length());
                          msg += ch;
                          if(stopFlag)
                                break;
                    }
                    catch(InterruptedException e) {}
              }
        }

        // Pause the banner.
        public void stop()
        {
              stopFlag = true;
              t = null;
        }

        // Display the banner.
        public void paint(Graphics g)
        {
              g.drawString(msg, 50, 30);
        }
  }
```

Output:



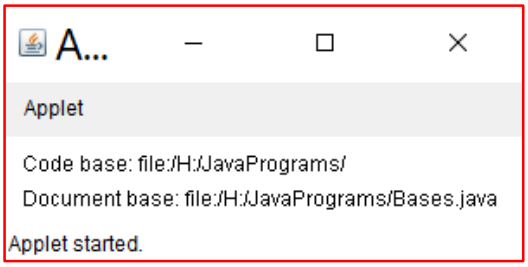
2.12 getDocumentBase( ) and getCodeBase( ):

- Java allows the applet to load data from the directory holding the HTML file that started the applet (the document base) and the directory from which the applet's class file was loaded (the code base).
- These directories are returned as URL objects by getDocumentBase( ) and getCodeBase( ). They can be concatenated with a string that names the file you want to load.

Example:

```
import java.awt.*;
import java.applet.*;
import java.net.*;
/*
<applet code="Bases" width=300 height=50>
</applet>
*/
public class Bases extends Applet
{
      // Display code and document bases.
      public void paint(Graphics g)
      {
            String msg;
            URL url = getCodeBase(); // get code base
            msg = "Code base: " + url.toString();
            g.drawString(msg, 10, 20);
            url = getDocumentBase(); // get document base
            msg = "Document base: " + url.toString();
            g.drawString(msg, 10, 40);
      }
}
```

Output:



2.13 Animation in Applet:

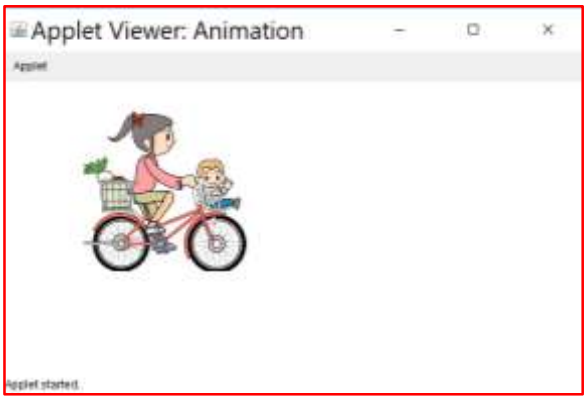
- One of the uses of applets is in performing animation, developing animation and developing games. Animation represents moving the objects from one place to another so that the objects look like alive.
- The following program shows a moving bi-cycle.

Example:

```
import java.awt.*;
import java.applet.*;
/* <applet code="Animation" width=600 height=300></applet> */
public class Animation extends Applet
```

```
{
  Image picture;

  public void init()
  {
          picture = getImage(getDocumentBase(),"Cycle.gif");
  }
  public void paint(Graphics g)
  {
    for(int i=0;i<500;i++)
     {
          Dimension d = getSize();
          g.setColor(Color.WHITE);
          g.fillRect(0, 0, d.width, d.height);
          g.drawImage(picture, i,30, null);
          try
          {
               Thread.sleep(50);
          }
          catch(Exception e){}
    }
  }
}
```

Output:



Explanation:

- **Image** class belongs to **java.awt** package and **getImage( )** method belongs to **Applet** class
- Method **setColor( )** belongs to **Graphics** class which sets the color to a specified color passed as argument.
- Method **fillRect( )** belongs to **Graphics** class which fills the rectangle of specified size with a already specified color by **setColor( )** method.
- To display an image in the applet, we ca use **drawImage()** method of **Graphics** class as:

  `g.drawImage(img, X, Y, obj);`

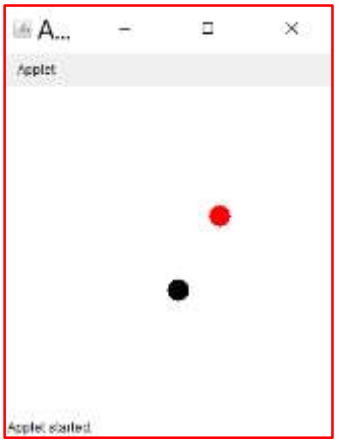
  Here, **img** represents the Image class object where the image is found. **X** and **Y** represents the coordinate at which the image would be displayed and **obj** represents the **ImageObserver** object which stores the history of how the image is loaded into memory. We can pass ***null*** in that place.
- The following program shows a bouncing ball applet.

Example:

```
import java.applet.Applet;
import java.awt.Color;
import java.awt.Graphics;
/*  <applet  code="BouncingBALL"  width=300  height=300></applet>
*/
class Ball
{
     int x,y,radius,dx,dy;
```

```
        Color BallColor;
      public  Ball(int  x,int  y,int  radius,int  dx,int  dy,Color
bColor)
        {
              this.x=x;
              this.y=y;
              this.radius=radius;
              this.dx=dx;
              this.dy=dy;
              BallColor=bColor;
        }
}
public class BouncingBALL extends Applet implements Runnable
{
        Ball redBall,blackBall;
        public void init()
        {
              redBall=new Ball(80,80,20,2,4,Color.red);
              blackBall=new Ball(40,70,20,4,2,Color.black);
              Thread t=new Thread(this);
              t.start();
        }
        public void paint(Graphics g)
        {
              g.setColor(redBall.BallColor);
              g.fillOval(redBall.x, redBall.y, redBall.radius,
                          redBall.radius);
              g.setColor(blackBall.BallColor);
              g.fillOval(blackBall.x, blackBall.y,
                            blackBall.radius, blackBall.radius);
        }
        public void run()
        {
              while(true)
              {
                    try
                    {
                          displacementOperation(redBall);
                          displacementOperation(blackBall);
                          Thread.sleep(20);
                          repaint();
                    }
                    catch(Exception e){}
              }
        }
        //This  method  checks  the  boundary  condition  of  ball
movement
        public void displacementOperation(Ball ball)
        {
              if(ball.y >= 200 || ball.y <= 0)
              {
                    ball.dy=-ball.dy;
              }
              if(ball.x >= 200 || ball.x <= 0)
              {
                    ball.dx=-ball.dx;
              }
              ball.y=ball.y-ball.dy;
              ball.x=ball.x-ball.dx;
        }
}
```

Output:



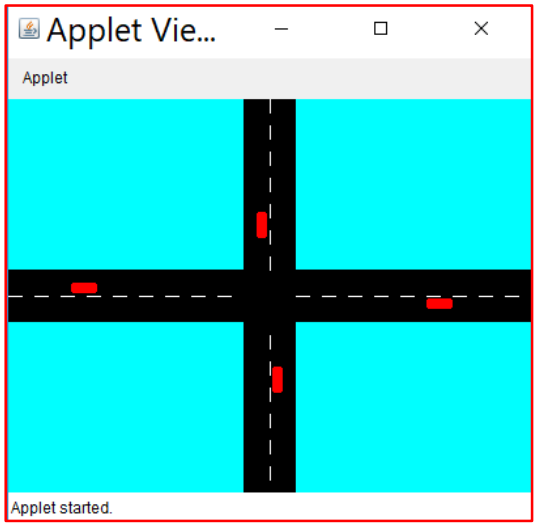
- The following program shows a car animation in which 4 cars are crossing each other at a square.

Example:

```
import java.awt.*;
import java.util.*;
import java.applet.*;
/*
<APPLET CODE="CarAnimation" WIDTH=400 HEIGHT=300>
</APPLET>
*/
public class CarAnimation extends Applet implements Runnable
{
      Thread t;
      //4 variables used to vary the car's positions.
      int x1=0,x2=380,y1=50,y2=250;
      public void start()
      {
            if(t==null)
            {
                  t=new Thread(this,"New Thread);
                  t.start();
            }
      }
      public void stop()
      {
            if(t!=null)
            {
            //On stop of applet the created thread is destroyed.
                  t=null;
            }
      }

      //Implementation of method run() of Runnable interface.
      public void run()
      {
            Thread t1=Thread.currentThread();
            while(t==t1)
            {
                  repaint();
                  try
                  {
                        Thread.sleep(100);
                  }
                  catch(Exception e)
                  {   }
            }
      }
      public void paint(Graphics g)
      {
            setBackground(Color.cyan);
```

```
                    g.setColor(Color.BLACK);
                    x1=(x1+16)%400;
                    x2=x2-16;
                    y1=(y1+12)%300;
                    y2=y2-12;
                    if(y2<0)
                          y2=288;
                    if(x2<0)
                          x2=384;

          //Draw the roads using 2 filled rectangles using black color.
                    g.fillRect(0,130,400,40);
                    g.fillRect(180,0,40,305);
                    //Draw the white colored lines.
                    g.setColor(Color.white);
                    for(int i=0;i<20;i++)
                    {
                          if(i!=9 && i!=10)
                                g.drawLine(i*20,150,i*20+10,150);
                    }
                    for(int j=0;j<15;j++)
                    {
                          if(j!=7 && j!=8)
                          g.drawLine(200,j*20,200,j*20+10);
                    }

                    //Draw 4 colored cars using filled round rectangles.
                    g.setColor(Color.red);
                    g.fillRoundRect(x2,152,20,8,2,2);
                    g.fillRoundRect(x1,140,20,8,2,2);
                    g.fillRoundRect(190,y1,8,20,2,2);
                    g.fillRoundRect(202,y2,8,20,2,2);
              }
          }
```

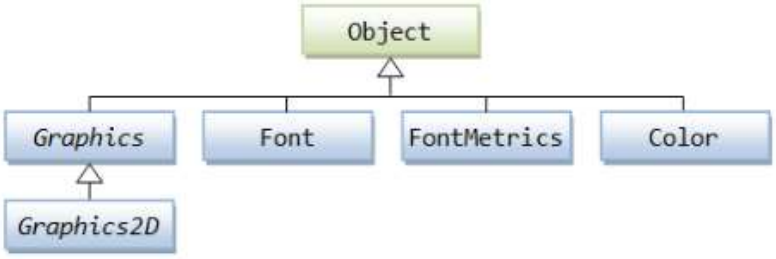
Output:



3. Graphics Programming in Java

3.1 The java.awt.Graphics Class:

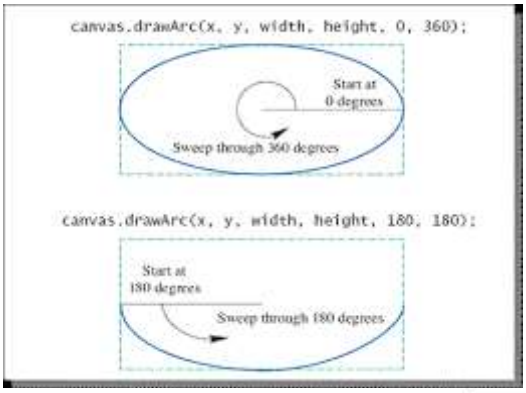
- In Java, custom painting is done via the java.awt.Graphics class, which manages a graphics context, and provides a set of device-independent methods for drawing texts, figures and images on the screen on different platforms.
- The java.awt.Graphics is an abstract class, as the actual act of drawing is system-dependent and device-dependent. Each operating platform will provide a subclass of Graphics to perform the actual drawing under the platform, but conform to the specification defined in Graphics.

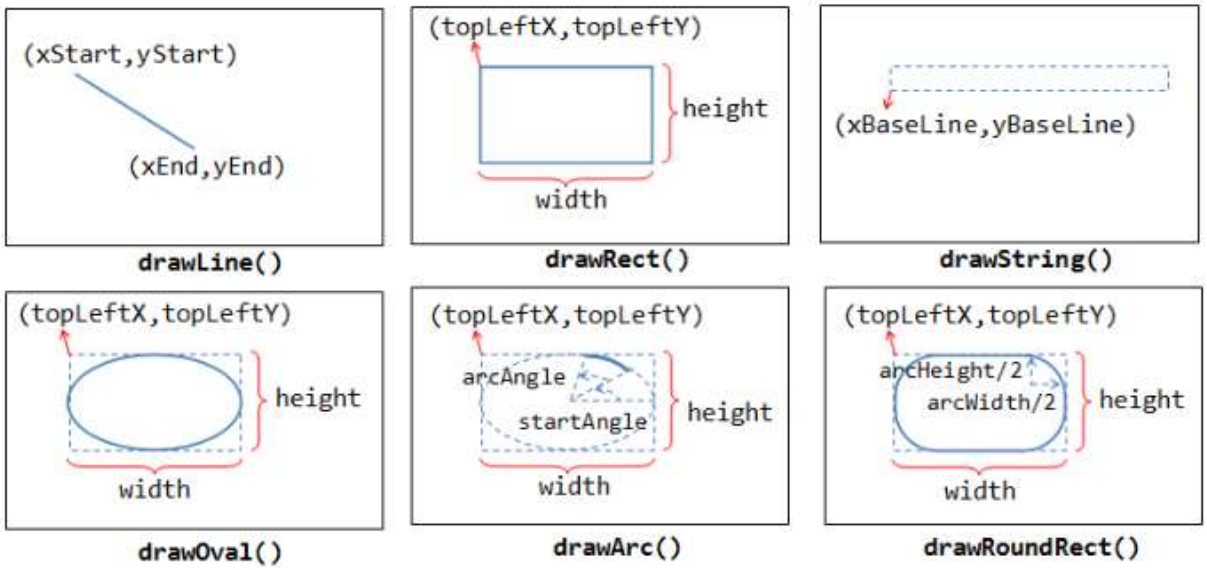


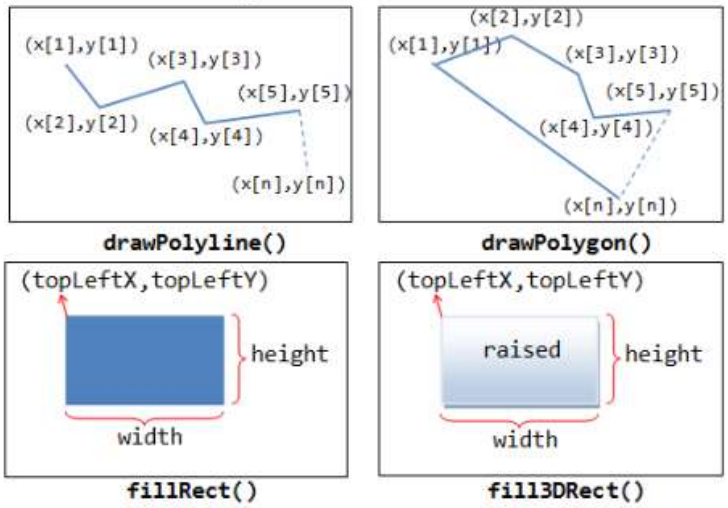
3.2 Graphics Class' Drawing Methods:

- The Graphics class provides methods for drawing three types of graphical objects:
  1. **Text strings:** via the drawString() method. Take note that System.out.println() prints to the system console, not to the graphics screen.
  2. **Vector-graphic primitives and shapes**: via methods drawXxx() and fillXxx(), where Xxx could be Line, Rect, Oval, Arc, PolyLine, RoundRect, or 3DRect.
  3. **Bitmap images:** via the drawImage() method.
- The methods of Graphics class we use frequently are described below:
  1. **public abstract Color getColor()** : Gets the current color.
  2. **public abstract void setColor(Color c):** Sets the current color to the specified color **c**. All subsequent graphics operations will use this specified color.
  3. **public abstract Font getFont():** Gets the current font.
  4. **public abstract void setFont(Font font):** Sets the font for all subsequent text-drawing operations.
  5. **public abstract void drawLine(int x1, int y1, int x2, int y2):** Draws a line between the coordinates (x1, y1) and (x2, y2).
  6. **public abstract void fillRect(int x, int y, int width, int height):** Fills the specified rectangle with the current color.
  7. **public void drawRect(int x, int y, int width, int height):** Draws the outline of the specified rectangle using the current color. Use drawRect(x, y, width-1, height-1) to draw the outline inside the specified rectangle.
  8. **public abstract void clearRect(int x, int y, int width, int height):** Clears the specified rectangle by filling it with the current background color of the current drawing surface.
  9. **public abstract void drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)**: Draws an outlined rounded corner rectangle using the current color. Here arcWidth is the diameter of the arc and arcHeight is the radius of the arc.
  10. **public abstract void fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight):** Draws a rounded rectangle filled in with the current color.
  11. **public abstract void drawOval(int x, int y, int width, int height):** Draws an oval inside the specified rectangle using the current color.
  12. **public abstract void fillOval(int x, int y, int width, int height):** Fills an oval inside the specified rectangle using the current color.
  13. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** Draws an arc bounded by the specified rectangle from startAngle to endAngle. 0 degrees is at the 3-o'clock position.Positive arc angles indicate counter-clockwise rotations, negative arc angles are drawn clockwise.
  14. **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** Fills an arc using the current color. This generates a pie shape.
  15. **public abstract void drawPolygon(int xPoints[], int yPoints[], int nPoints):** Draws a polygon defined by an array of x points and y points. Here xPoints is an array of x points, yPoints is an array of y points and nPoints is the total number of points
  16. **public abstract void fillPolygon(int xPoints[], int yPoints[], int nPoints):** Fills a polygon with the current color.
  17. **public abstract void drawString(String str, int x, int y):** Draws the specified String using the current font and color. The x ,y position is the starting point of the baseline of the String.

18. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** Draws the specified image at the specified coordinate (x, y). If the image is incomplete the image observer will be notified later. Here **ImageObserver** observer - notifies if the image is complete or not.







3.3 Colors:

- The class java.awt.Color provides 13 standard colors as named-constants. They are: Color.RED, GREEN, BLUE, MAGENTA, CYAN, YELLOW, BLACK, WHITE, GRAY, DARK_GRAY, LIGHT_GRAY, ORANGE, and PINK.
- We can use the toString() to print the RGB values of these color (e.g., System.out.println(Color.RED))

- We can also use the RGB values to construct our own color via constructors:
```
Color(int r, int g, int b);              // between 0 and 255
```
Example:
```
Color myColor1 = new Color(123, 111, 222);
```

3.4 Fonts:

- The class java.awt.Font represents a specific font face, which can be used for rendering texts. We can use the following constructor to construct a Font instance:
```
public Font(String name, int style, int size);
```
  - Here **name** is the font family name like "Dialog", "DialogInput", "Monospaced", "Serif", or "SansSerif" etc. We can also use String constants Font.DIALOG, Font.DIALOG_INPUT, Font.MONOSPACED, Font.SERIF, Font.SANS_SERIF etc.
  - The parameter **style** indicates font style like Font.PLAIN, Font.BOLD, Font.ITALIC or Font.BOLD|Font.ITALIC (Bit-OR)
  - The parameter size is the point size of the font (in pt) (1 inch has 72 pt).
- We can use the setFont() method to set the current font for the Graphics context g for rendering texts.
- Example:
```
Font myFont1 = new Font(Font.MONOSPACED, Font.PLAIN, 12);
Font myFont2 = new Font(Font.SERIF, Font.BOLD | Font.ITALIC, 16);
// bold and italics
JButton btn = new JButton("RESET");
btn.setFont(myFont1);
JLabel lbl = new JLabel("Hello");
lbl.setFont(myFont2);
......
g.drawString("In default Font", 10, 20);     // in default font
Font myFont3 = new Font(Font.SANS_SERIF, Font.ITALIC, 12);
g.setFont(myFont3);
g.drawString("Using the font set", 10, 50);  // in myFont3
```

4. Abstract Window Toolkit (AWT)

4.1 Introduction to Graphics User Interface (GUI):

- The environment where the user can interact with an application through graphics or images is called Graphics User Interface (GUI).
- GUI has the following advantages:
  - It is user-friendly. The user need not worry about any commands.
  - It adds attraction and beauty to any application by adding pictures, colors, menus, animation etc.
  - It is possible to simulate the real life objects using GUI. For example, we can develop a calculator program which can behave like a real calculator.
  - GUI helps to create graphical coponents like push buttons, radio buttons, check boxes, et and use them effectively in our programs.

4.2 AWT:

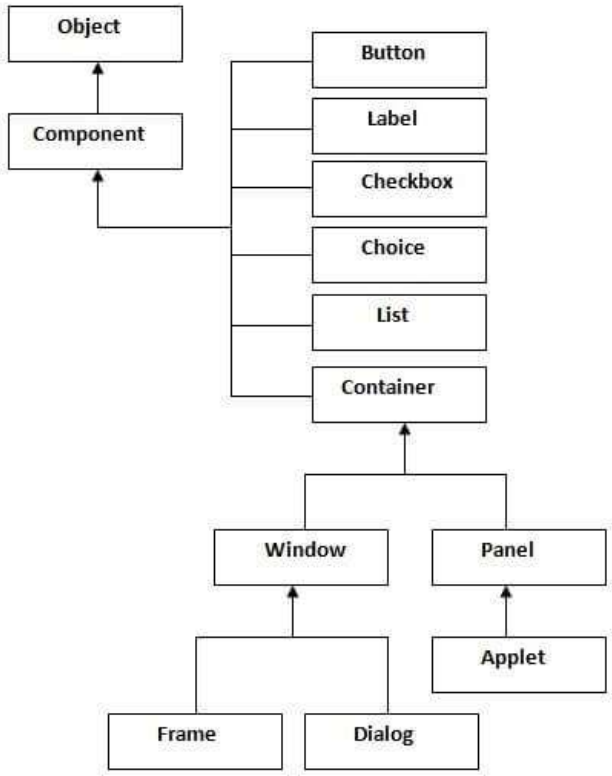
- Java Abstract Window Toolkit (AWT) is an API to develop GUI or window-based applications in java.
- Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.
- The **java.awt** package provides classes for AWT API such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.
- AWT is huge! It consists of 12 packages of 370 classes.
- The **java.awt** package contains the core AWT graphics classes:
  - GUI Component classes, such as Button, TextField, and Label.
  - GUI Container classes, such as Frame and Panel.
  - Layout managers, such as FlowLayout, BorderLayout and GridLayout.
  - Custom graphics classes, such as Graphics, Color and Font.
- The **java.awt.event** package supports event handling:

- Event classes, such as ActionEvent, MouseEvent, KeyEvent and WindowEvent,
- Event Listener Interfaces, such as ActionListener, MouseListener, MouseMotionListener, KeyListener and WindowListener,
- Event Listener Adapter classes, such as MouseAdapter, KeyAdapter, and WindowAdapter.

- AWT provides a platform-independent and device-independent interface to develop graphic programs that runs on all platforms, including Windows, Mac OS X, and Unixes.

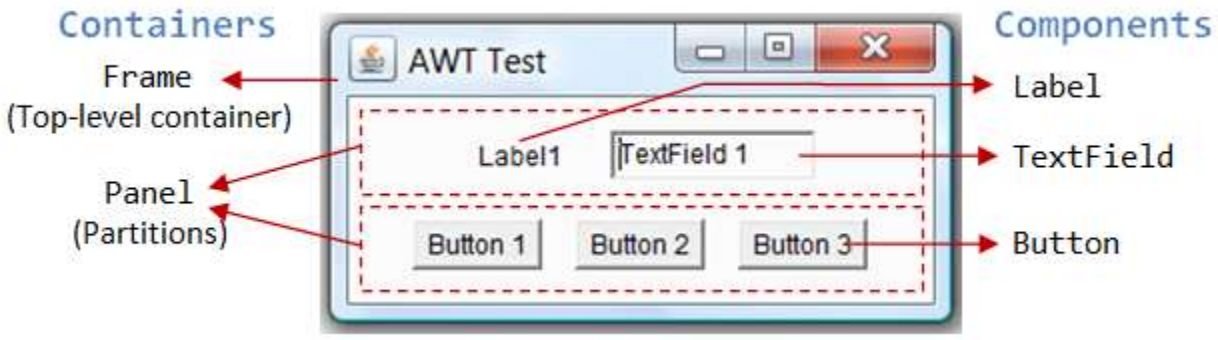
4.3 Java AWT Hierarchy:

- The hierarchy of Java AWT classes are given below.



4.4 Containers and Components:

- There are two types of GUI elements:
  - **Component**: Components are elementary GUI entities, such as Button, Label, and TextField. At the top of the AWT hierarchy is the **Component** class. **Component** is an abstract class that encapsulates all of the attributes of a visual component. All user interface elements that are displayed on the screen and that interact with the user are subclasses of **Component**.
  - **Container**: The **Container** class is a subclass of **Component**. It has additional methods that allow other **Component** objects to be nested within it. Other **Container** objects can be stored inside of a **Container.** Containers, such as Frame and Panel, are used to hold components in a specific layout (such as FlowLayout or GridLayout). A container can also hold sub-containers.

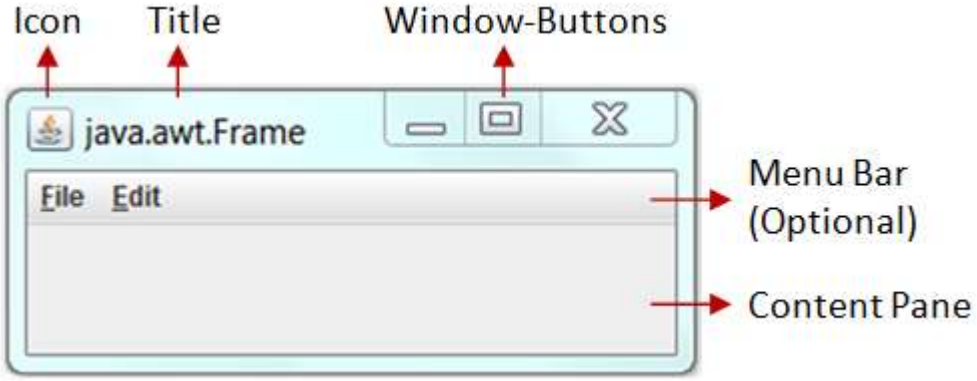


- In the above figure, there are three containers: a Frame and two Panels.
- A Frame is the top-level container of an AWT program. A Frame has a title bar (containing an icon, a title, and the minimize/maximize/close buttons), an optional menu bar and the content display area.
- The Panel class is a concrete subclass of Container. It doesn't add any new methods; it simply implements Container. A Panel is a rectangular area used to group related GUI components in a certain layout. In the above figure, the top-level Frame contains two Panels. There are five components: a Label (providing description), a TextField (for users to enter text), and three Buttons (for user to trigger certain programmed actions).
- In a GUI program, a component must be kept in a container. We need to identify a container to hold the components. Every container has a method called add(Component c). A container (say c) can invoke c.add(aComponent) to add aComponent into itself. For example,

```
Panel pnl = new Panel();          // Panel is a container
Button btn = new Button("Press"); // Button is a component
pnl.add(btn);                     // The Panel container adds a Button component
```

4.5 AWT Containers Classes:

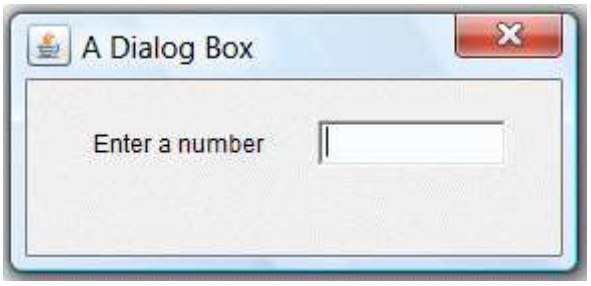
- Each GUI program has a top-level container. The commonly-used top-level containers in AWT are:
  - **Window and Frame**: The window is the container that have no borders and menu bars. We must use frame, dialog or another window for creating a window.



A Frame provides the "main window" for your GUI application. It has a title bar (containing an icon, a title, the minimize, maximize/restore-down and close buttons), an optional menu bar, and the content display area. To write a GUI program, we typically start with a subclass extending from **java.awt.Frame** to inherit the main window as follows:

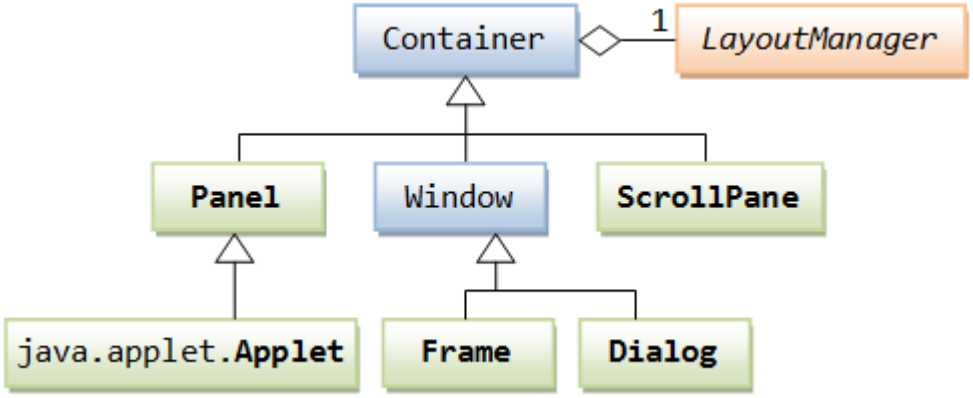
```
import java.awt.Frame;

// A GUI program as a subclass of Frame
public class MyGUIProgram extends Frame
{

      // private variables
        ......

      // Constructor to setup the GUI components
      public MyGUIProgram() { ...... }

      // methods
      ......
      ......

      // The entry main() method
      public static void main(String[] args)
      {
      // Invoke the constructor (to setup the GUI)
```

```
        new MyGUIProgram();
        }
}
```



- **Dialog**: An AWT Dialog is a "pop-up window" used for interacting with the users. A Dialog has a title-bar (containing an icon, a title and a close button) and a content display area.
- **Applet**: An AWT Applet (in package java.applet) is the top-level container for an applet, which is a Java program running inside a browser.

- Secondary containers are placed inside a top-level container or another secondary container. AWT provides these secondary containers:
  - **Panel:** a rectangular box used to layout a set of related GUI components in pattern such as grid or flow.
  - **ScrollPane:** provides automatic horizontal and/or vertical scrolling for a single child component others.
- The hierarchy of the AWT Container classes is as follows:



- A Container has a LayoutManager to layout the components in a certain pattern.

4.6 AWT Component Classes:

- AWT provides many ready-made and reusable GUI components in package java.awt. The frequently-used are: Button, TextField, Label, Checkbox, CheckboxGroup (radio buttons), List, and Choice, as illustrated below:

4.7 Creating a Frame:

- As frame is the basic component in AWT, it has to be created before creation of any other component.
- All other components can be displayed in a frame.
- There are three ways to create a frame, which are as follows:
  - Create a frame class object by calling the default constructor of frame class:

```
Frame f = new Frame();
```

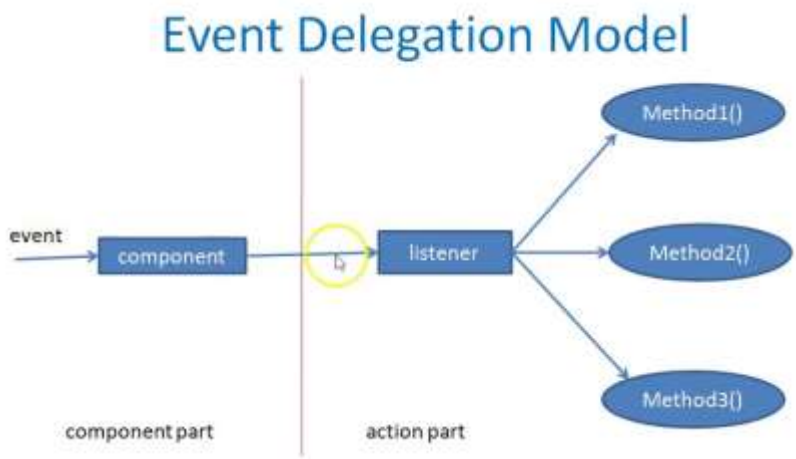
- Create a frame class object by calling the parameterized constructor of frame class which receives the title of the frame:
```
Frame f = new Frame("My Frame");
```
- Create a subclass MyFrame to the Frame class and create an object to the subclass as:
```
Class MyFrame extends Frame
MyFrame f = new MyFrame( );
```

- Initially a frame with initial size of 0 pixels width and 0 pixel height will be created.
- Frame size can be increased by using the **setSize( )** method as:
```
f.setSize(400,300);
```
- We can display the frame by using the **setVisible( )** method as:
```
f.setVisible(true);
```

4.8 AWT Event in Java:

4.8.1 Event Delegation Model:

- When we create a component, the component is displayed but is not capable of performing any action.
- An event represents a specific action done on a component. Example: clicking, typing, mouse over etc.
- To let the component understand that an event is generated on it, we should add some listener to the component. A listenener is an interface which listens to an event coming from a component. A listener will have some abstract methods which need to be implemented by the programmer.
- When an event is generated by the user on the component, the component sends (delegates) that event to the listener attached to it. The listener hands over (delegates) the event to an appropriate method. Finally, the method is executed and the event is handled. This is called event delegation model as shown in the following diagram.



- **Advantage of Event Delegation Model:**
  - The component and the action parts can be developed in two separate environments for example, we can create the component in Java and the action logic can be developed in VisualBasic.
  - We can modify one part (component part or action part) without effecting any modification to the other. This makes debugging and maintenance of code very easy.

4.8.2 Event Handling:

- Each AWT component, except Panel and Label generates events when interacted by the user like clicking over a button or pressing enter key in a text field etc. Listeners handle the events. The style (or design pattern) Java follows to handle the events are as follows:
- The event handling Java involves four types of classes.
  1. Event Sources
  2. Event classes
  3. Event Listeners
  4. Event Adapters

1. **Event Sources:** Event sources are components, subclasses of java.awt.Component, capable to generate events. The event source can be a button, TextField or a Frame etc.
2. **Event classes:** Almost every event source generates an event and is named by some Java class. For example, the event generated by button is known as ActionEvent and that of Checkbox is known as ItemEvent. All the events are listed in java.awt.event package. Following list gives some of the components and their corresponding listeners.
3. **Event Listener:** The events generated by the GUI components are handled by a special group of interfaces known as "listeners". Note, Listener is an interface. Every component has its own listener, say, AdjustmentListener handles the events of scrollbar. Some listeners handle the events of multiple components. For example, ActionListener handles the events of Button, TextField, List and Menus. Listeners are from java.awt.event package. The following table lists components, their corresponding listeners, and the listener methods.

Component	Listener (Interface)	Listener Methods	Event Class
Button	ActionListener	public void actionPerformed (ActionEvent ae)	ActionEvent
CheckBox	ItemListener	public void itemStateChanged(ItemEvent e)	ItemEvent
CheckBoxGroup	ItemListener	public void itemStateChanged(ItemEvent e)	ItemEvent
TextField	ActionListener	public void actionPerforemed(ActionEvent e)	ActionEvent
	FocusListener	public void focusGained(FocusEvent e) public void focusLost(FocusEvent e)	FocusEvent
TextArea	ActionListener	public void actionPerformed(ActionEvent e)	ActionEvent
	FocusListener	public void focusGained(FocusEvent e) public void focusLost(FocusEvent e)	FocusEvent
Choice	ActionListener	public void actionPerformed(ActionEvent e)	ActionEvent
	ItemListener	public void itemStateChanged(ItemEvent e)	ItemEvent
List	ActionListener	public void actionPerformed(ActionEvent e)	ActionEvent
	ItemListener	public void itemStateChanged(ItemEvent e)	ItemEvent
ScrollBar	AdjustmentListener	public void adjustmentValueChanged (AdjustmentEvent e)	AdjustmentEvent
	MouseMotionListener	public void mouseDragged(MouseEvent e) public void mouseMoved(MouseEvent e)	MouseEvent
Frame	WindowListener	public void windowActivated(WindowEvent e) public void windowClosed(WindowEvent e) public void windowClosing(WindowEvent e) public void windowDeactivated(WindowEvent e) public void windowDeiconified(WindowEvent e) public void windowIconified(WindowEvent e) public void windowOpened(WindowEvent e)	WindowEvent
KeyBoard	KeyListener	public void KeyPressed(KeyEvent e) public void KeyReleased(KeyEvent e) public void KeyTyped(KeyEvent e)	KeyEvent
Mouse	MouseListener	public void mouseReleased(MouseEvent e) public void mouseClicked(MouseEvent e) public void mouseExited(MouseEvent e) public void mouseEntered(MouseEvent e) public void mousepressed(MouseEvent e)	MouseEvent
	MouseMotionListener	void mouseDragged(MouseEvent e) void mouseMoved(MouseEvent e)	

4. **Event Adapters:** When a listener includes many abstract methods to override, the coding becomes heavy to the programmer. For example, to close the frame, you override seven abstract methods of WindowListener, in which, infact you are using only one method. To avoid this heavy coding, the designers come with another group of classes known as

"adapters". Adapters are abstract classes defined in java.awt.event package. Every listener that has more than one abstract method has got a corresponding adapter class.

4.8.3 Event Handling Example:

- Observe the following skeleton code:

```
public class ButtonDemo extends Frame implements ActionListener
{
public ButtonDemo()
{
      Button btn = new Button("OK");
      btn.addActionListener(this);
      add(btn);
}
public void actionPerformed(ActionEvent e)
{
      String str = e.getActionCommand();
}
}
```

- A button object btn is created for which events are yet to be linked. For this, the first step is implementing ActionListener to the class ButtonDemo. In the second step, we link or register the button btn with the ActionListener. For this addActionListener() method of Button class is used. The parameter "this" refers the ActionListener.

```
btn.addActionListener(this);
```

- With the above statement, btn is linked with the ActionListener.
  1. When the button btn is clicked, the button generates an event called ActionEvent. It is the nature of the button taken care by JVM.
  2. This ActionEvent reaches the ActionListener because we registered the button with ActionListener earlier.
  3. Now, the question is, what ActionListener does with the ActionEvent object it received?

     The ActionListener simply calls actionPerformed() method and passes the ActionEvent object to the parameter as follows.

     ```
     public void actionPerformed(ActionEvent e)
     ```

     The parameter for the above method comes from ActionListener.
  4. Finally the ActionEvent object generated by the button btn reaches the ***e*** object of ActionEvent. All this is done by JVM implicitly.

4.9 Closing the Frame:

- To close the frame follow the following steps:
  - **Attach a listener available in java.awt.event package to the frame component.** The most suitable listener to the frame is 'WindowListener'. It can be attached using addWindowListener ( ) method as:

    ```
    f.addWindowListener(WindowListener obj);
    ```

    Note: The addWindowListener( ) method has a parameter that is expecting object of WindowListener interface. Since it is not possible to create an object te an interface, we should create an object to the implementation class ofthe interface and pass it to the method.
  - **Implement all the methods of the WindowListener interface.** The following methods are found in WindowListener interface:

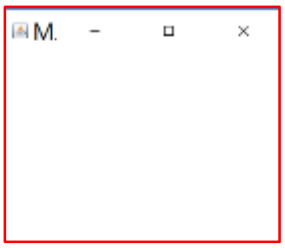
    ```
    public void windowActivated(WindowEvent e)
    public void windowClosed(WindowEvent e)
    public void windowClosing(WindowEvent e)
    public void windowDeactivated(WindowEvent e)
    public void windowDeiconified(WindowEvent e)
    public void windowIconified(WindowEvent e)
    public void windowOpened(WindowEvent e)
    ```

Note: To close the frame only WindowClosing method is enough. For the remaining methods we can provide empty body.

Example:

```
import java.awt.*;
import java.awt.event.*;
class MyFrame extends Frame
{
      public static void main(String args[])
      {
            Frame f = new Frame("My AWT Frame");
            f.setSize(300,250);
            f.setVisible(true);
            f.addWindowListener(new MyClass());
      }
}
class MyClass implements WindowListener
{
      public void windowActivated(WindowEvent e){}
      public void windowClosed(WindowEvent e){}
      public void windowClosing(WindowEvent e)
      {
            System.exit(0);
      }
      public void windowDeactivated(WindowEvent e){}
      public void windowDeiconified(WindowEvent e){}
      public void windowIconified(WindowEvent e){}
      public void windowOpened(WindowEvent e){}
}
```

Output:



- In the previous program we had to mention all the methods of WindowListener interface, just for the sake of one method.
- There is another alternative to do this in an efficient way. There is a class **WindowAdapter** in java. awt. event package, that contains all the methods of the WindowListener interface with an empty implementation (body).
- By extending our class Myclass from this WindowAdapter class, then we need not write all the methods with empty implementation. We can write only that method which interests us. This is shown in the following modified program:

  Example:

```
import java.awt.*;
import java.awt.event.*;
class MyFrame extends Frame
{
   public static void main(String args[])
   {
         MyFrame f = new MyFrame();
         f.setTitle("My AWT Frame");
         f.setSize(300,250);
         f.setVisible(true);
         f.addWindowListener(new MyClass());
   }
}
class MyClass extends WindowAdapter
{
   public void windowClosing(WindowEvent e)
```

```
            {
                  System.exit(0);
            }
      }
```

NOTE: An adapter Class is an implementation class of a listener interface which contains all methods implemented with empty body. For example, WindowAdapter is an adapter class of WindowListener interface. Adapter classes reduce overhead on programming while working with listener interfaces.

- We can also rewrite the above program by using anonymous inner class concept as follows:

Example:

```
import java.awt.*;
import java.awt.event.*;
class MyFrame extends Frame
{
      public static void main(String args[])
      {
            MyFrame f = new MyFrame();
            f.setTitle("My AWT Frame");
            f.setSize(300,250);
            f.setVisible(true);
            f.addWindowListener(new WindowAdapter()
            {
                  public void windowClosing(WindowEvent e)
                  {
                        System.exit(0);
                  }
            });
      }
}
```

4.10 Uses of Frame:

- After the creation of frame, we can use it for any of the following purposes:
  - ✓ To draw some graphical shapes like dots, lines, rectangles, etc. in the frame.
  - ✓ To display some text in the frame.
  - ✓ To display pictures or images in the frame.
  - ✓ To display components like push buttons, radio buttons, etc. in the frame.

4.11 Drawing in the Frame

- Graphics class of java. awt package has many methods which help to draw various shapes.
- These methods are already discussed in section 4.2.
- The following program draws a smiling face using the methods of Graphics class.

Example:

```
import java.awt.*;
import java.awt.event.*;
class DrawSmile extends Frame
{

      public void paint(Graphics g)
      {
            g.setColor(Color.blue);
            g.drawRect(50,50,160,125);
            g.drawOval(90,70,80,80);
            g.drawOval(110,95,5,5);
            g.drawOval(145,95,5,5);
            g.drawLine(130,95,130,115);
            g.drawArc(113,115,35,20,180,180);
      }
      public static void main(String args[])
      {
            DrawSmile d = new DrawSmile();
            d.setTitle("My Drawing");
            d.setSize(400,400);
            d.setVisible(true);
```

```
                d.addWindowListener(new WindowAdapter()
                {
                      public void windowClosing(WindowEvent e)
                      {
                            System.exit(0);
                      }
                });
          }
    }
```

Output:

- **Note:**The paint() method is automatically called when a frame is created and displayed.
- We can rewrite the above program in the following way.

```
import java.awt.*;
import java.awt.event.*;
class DrawSmile extends Frame
{
      DrawSmile()
      {
            this.addWindowListener(new WindowAdapter()
            {
                  public void windowClosing(WindowEvent e)
                  {
                        System.exit(0);
                  }
            });

      }
      public void paint(Graphics g)
      {
            g.setColor(Color.blue);
            g.drawRect(50,50,160,125);
            g.drawOval(90,70,80,80);
            g.drawOval(110,95,5,5);
            g.drawOval(145,95,5,5);
            g.drawLine(130,95,130,115);
            g.drawArc(113,115,35,20,180,180);
      }
      public static void main(String args[])
      {
            DrawSmile d = new DrawSmile();
            d.setTitle("My Drawing");
            d.setSize(400,400);
            d.setVisible(true);
      }
}
```

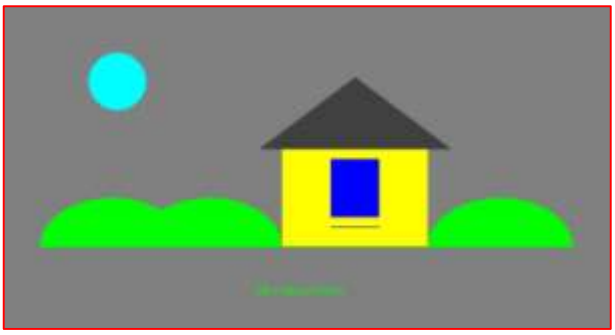
- The following program draws a home with a moon at background using the methods of Graphics class.

Example:

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

```
class Home extends Frame
{
	Home()
	{
		this.addWindowListener(new WindowAdapter()
		{
			public void windowClosing(WindowEvent e)
			{
				System.exit(0);
			}
		});

	}
	public void paint(Graphics g)
	{
		int x[]={375,275,475};
		int y[]={125,200,200};
		int n=3;

		this.setBackground(Color.gray);

		g.setColor(Color.yellow);
		g.fillRect(300,200,150,100);

		g.setColor(Color.blue);
		g.fillRect(350,210,50,60);

		g.drawLine(350,280,400,280);

		g.setColor(Color.darkGray);
		g.fillPolygon(x,y,n);

		g.setColor(Color.cyan);
		g.fillOval(100,100,60,60);

		g.setColor(Color.green);
		g.fillArc(50,250,150,100,0,180);
		g.fillArc(150,250,150,100,0,180);
		g.fillArc(450,250,150,100,0,180);

		g.drawLine(50,300,600,300);
		g.drawString("My Happy Home",275,350);
	}
	public static void main(String args[])
	{
		Home h = new Home();
		h.setTitle("My Home");
		h.setSize(650,400);
		h.setVisible(true);
	}
}
```

Output:

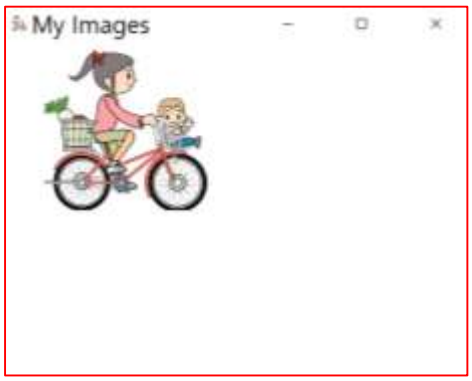


- The following program demonstrates how to display an image in the frame and its title bar.

Example:

```
import java.awt.*;
import java.awt.event.*;
class Images extends Frame
{
      static Image img;
      Images()
      {
            //load the image into Image object
            img=Toolkit.getDefaultToolkit().getImage("Cycle.gif");

            //wait till the image is loaded into img object
            //for this prpose, create MediaTracker
            MediaTracker track=new MediaTracker(this);

            //add image to MediaTracker
            track.addImage(img,0);
            try
            {
                  track.waitForID(0);
            }
            catch(InterruptedException e){}

            addWindowListener(new WindowAdapter()
            {
                  public void windowClosing(WindowEvent e)
                  {
                        System.exit(0);
                  }
            });

      }
      public void paint(Graphics g)
      {

            g.drawImage(img,50,50,null);
      }
      public static void main(String args[])
      {
            Images i = new Images();
            i.setTitle("My Images");
            i.setSize(500,400);
            i.setIconImage(img);//display the same image in the title bar
            i.setVisible(true);
      }
}
```

Output:



- **Explanation:** To display images like .gif and .jpg files in the frame, we should follow the following steps:
  - Load the image into Image class object using getImage () method of Toolkit class.

```
img=Toolkit.getDefaultToolkit().getImage("Cycle.gif");
```

- Loading the image into img takes some time. JVM uses a separate thread to load image into img and continues with the rest of the program. Therefore, there is a possibility of completing the program execution before the image is completely loaded into img. To avoid this, we should make JVM the image is completely loaded into img object. i.e. we need to wait for that.
- To do that we need to create an ID of that image. For this purpose, we need MediaTracker class. So add the image to MediaTracker class and allot an identification number to it starting 0,1, ...
```
MediaTracker track=new MediaTracker(this);
track.addImage(img,0);
```
- Now, MediaTracker keeps JVM waiting till the image is loaded completely. This is done by the following code:
```
track.waitForID(0);
```
- Once the image is loaded and available in img, then we can display the image using drawImage() method of Graphics class, as:
```
g.drawImage(img,50,50,null);
```
- Another alternative for drawImage() is:
```
g.drawImage(img,50,50,200,250,null);
```
Here 200, 250 indicates the height and width of the image.
- To display an image in the title bar of the frame, we can use setIconImage () method of Frame class, as:
```
i.setIconImage(img);
```

4.12 Component Class Methods:

- A component is a graphical representation of an object on the screen. For example, frame, push button, radio buttons, menus, etc. are components.
- There is Component class available in java.awt package which contains the following methods which are applicable any component.
  - **Font getFont ():** This method returns the font of the component.
  - **void setFont(Font f) :** This method sets a particular font f for the text of the component.
  - **Color getForeground ()** : This method gives the foreground color of the component.
  - **void setForeground (Color c)** : This method sets a foreground color c to the component.
  - **Color getBackground ()** : Gets the background color of the component.
  - **void setBackground (Color c)** : Sets the background color c for the component.
  - **String getName ()** : Returns the name of the component.
  - **void setName (String name):** Sets a new name for the component.
  - **int getHeight ()** : Returns the height of the component in pixels as an integer.
  - **int getWidth ()**: Returns the width of the component in pixels as an integer.
  - **Dimension getSize ()**: Returns the size of the component as an object of Dimension class. Dimenison.width and Dimenision.height will provide the width and height of the component.
  - **int getX ()** : Returns the cunenl x coordinate of the component's origin.
  - **int getY ()** : Returns the current y coordinate of the component's origin.
  - **Point getLocation ()** : Gets the location of the component in the form of a point specifying the component's top-left corner.
  - **void setLocation (int x, int y):** Moves the component to a new location specified by (x,y).
  - **void setSize (int width, int height):** Resizes the component so that it has new width and height as passed to setSize () method.
  - **void setVisible (boolean b) :** Shows or hides the component depending on the value of parameter b.setVisible(true) will display the component and setVisible (false) hides the component.

- **void setEnabled (boolean b) :** Enables or disables the component, depending on the value of the parameter b.setEnabled (true) will enable the component to function and setEnabled (false) will disable it.
- **void setBounds (int x, int y, int w, int h):** This method allots a rectangular area starting at (x,y) coordinates and with width w and height h. The component is resized to this area before its display. This method is useful to specify the location of the component in the frame.
- After creating a component, we should add the component to the frame. For this purpose, add ( ) method is used.
```
f.add(component);
```
- To remove a component from the frame, we can use remove () method, as:
```
f.remove(component);
```

4.13 Push Button:

- To create a push button with a label, we can create an object to Button class, as:
```
Button b = new Button(); //Button without label
Button b = new Button("Save"); //Button with label
```
- To get the label of the button, use getLabel ():
```
String l = b.getLabel( );
```
- To set the label of the button:
```
b.setLabel("Click Me");
```
- To know the lable of the button the user has clicked use getActionCommand () method of ActionEvent class as follows:
```
String s = ae.getActionCommand();
```
- To know the source object which has been clicked by the user, we can use getSource() method of ActionEvent class as follows:
```
Object obj = ae.getSource();
```
- To handle event handling using button we need pass the event to the listener which will call the appropriate method to handle it. With the push buttons ActionListener is the suitable listener. To handle ActionListener to the button, we can use addActionListener() method as:
```
b.addActionListener(ActionListener obj);
```
- Similarly to to remove action listener from the button, we can use removeActionListener as:
```
b.removeActionListener(ActionListener obj);
```
- As ActionListener is an interface, we cannot create its object and hence cannot pass it to above two methods. Therefore, we should pass object of implementation class of the interface in this case. Normally we use **this** which represents the implementation class object.
- At the beginning before creation of the buttons we need to set a layout manager using setLayout () method. In a program, if we do not want any layout, then we need to pass null to setLayout () as:
```
this.setLayout(null);
```
- The following program creates three push buttons showing three different colors as their label. When a button is clicked, that particular color is set as background color in the frame.

Example:

```
import java.awt.*;
import java.awt.event.*;
class MyButtons extends Frame implements ActionListener
{
     Button b1,b2,b3;
     MyButtons()
     {
          this.setLayout(null);

          b1=new Button("Yellow");
```

```
            b2=new Button("Blue");
            b3=new Button("Pink");

            b1.setBounds(100,100,70,40);
            b2.setBounds(100,160,70,40);
            b3.setBounds(100,220,70,40);

            this.add(b1);
            this.add(b2);
            this.add(b3);
            b1.addActionListener(this);
            b2.addActionListener(this);
            b3.addActionListener(this);
            addWindowListener(new WindowAdapter()
            {
                  public void windowClosing(WindowEvent e)
                  {
                        System.exit(0);
                  }
            });

      }
      public void actionPerformed(ActionEvent ae)
      {
            String str = ae.getActionCommand();
            if(str.equals("Yellow"))
                  this.setBackground(Color.yellow);
            if(str.equals("Blue"))
                  this.setBackground(Color.blue);
            if(str.equals("Pink"))
                  this.setBackground(Color.pink);
      }
      public static void main(String args[])
      {
            MyButtons mb = new MyButtons();
            mb.setTitle("My Buttons");
            mb.setSize(650,400);
            mb.setVisible(true);
      }
}
```

Output:

- The above program can be rewritten with two variations:
    - We have used FlowLayout manager to arrange the components in the frame in a line one after the other. Note that when we use any layout manger there is no need o use setBounds() method to specify the location of the components.
    - Instead of using getActionCommand( ) we have used getSource( ).

Example:

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

```
{
	Button b1,b2,b3;
	MyButtons()
	{
		this.setLayout(new FlowLayout());

		b1=new Button("Yellow");
		b2=new Button("Blue");
		b3=new Button("Pink");

		add(b1);
		add(b2);
		add(b3);

		b1.addActionListener(this);
		b2.addActionListener(this);
		b3.addActionListener(this);

		addWindowListener(new WindowAdapter()
		{
			public void windowClosing(WindowEvent e)
			{
				System.exit(0);
			}
		});

	}
	public void actionPerformed(ActionEvent ae)
	{
		if(ae.getSource()==b1)
			setBackground(Color.yellow);
		if(ae.getSource()==b2)
			this.setBackground(Color.blue);
		if(ae.getSource()==b3)
			this.setBackground(Color.pink);
	}
	public static void main(String args[])
	{
		MyButtons mb = new MyButtons();
		mb.setTitle("My Buttons");
		mb.setSize(650,400);
		mb.setVisible(true);
	}
}
```

4.14 Java 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
1. javax.swing.BoxLayout
2. javax.swing.GroupLayout
3. javax.swing.ScrollPaneLayout
4. javax.swing.SpringLayout etc.

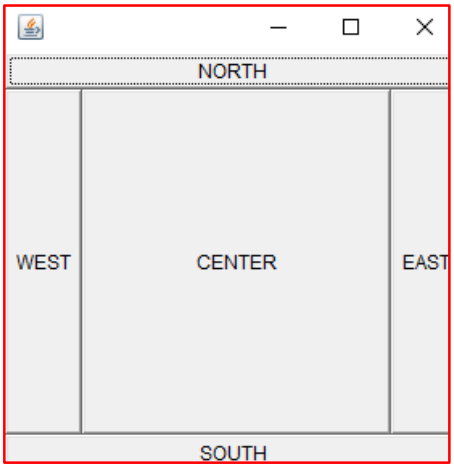
4.14.1 Java BorderLayout:

- 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:
  - public static final int NORTH
  - public static final int SOUTH
  - public static final int EAST
  - public static final int WEST
  - public static final int CENTER
- The following are the constructors of the 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:

```
import java.awt.*;
import java.awt.event.*;
class BLayoutDemo extends Frame
{
      Button b1,b2,b3,b4,b5;
      BLayoutDemo()
      {
            b1=new Button("NORTH");
            b2=new Button("SOUTH");
            b3=new Button("EAST");
            b4=new Button("WEST");
            b5=new Button("CENTER");

            this.add(b1,BorderLayout.NORTH);
            this.add(b2,BorderLayout.SOUTH);
            this.add(b3,BorderLayout.EAST);
            this.add(b4,BorderLayout.WEST);
            this.add(b5,BorderLayout.CENTER);

            addWindowListener(new WindowAdapter()
            {
                  public void windowClosing(WindowEvent e)
                  {
                        System.exit(0);
                  }
            });

      }
      public static void main(String args[])
      {
            BLayoutDemo bld = new BLayoutDemo();
            bld.setTitle("My Buttons");
            bld.setSize(650,400);
            bld.setVisible(true);
      }
}
```

Output:



4.14.2 Java GridLayout:

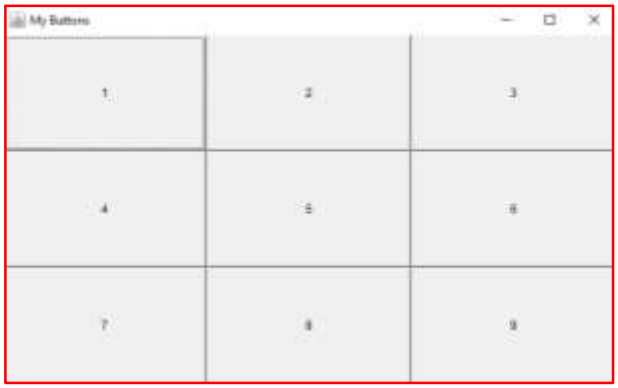
- The GridLayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle.
- The following are the constructors of the GridLayout class
  - **GridLayout():** creates a grid layout with one column per component in a row.
  - **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.
  - **GridLayout(int rows, int columns, int hgap, int vgap):** creates a grid layout with the given rows and columns along with given horizontal and vertical gaps.

Example:

```
import java.awt.*;
import java.awt.event.*;
class GLayoutDemo extends Frame
{
      Button b1,b2,b3,b4,b5,b6,b7,b8,b9;
      GLayoutDemo()
      {
            this.setLayout(new GridLayout(3,3));

            b1=new Button("1");
            b2=new Button("2");
            b3=new Button("3");
            b4=new Button("4");
            b5=new Button("5");
            b6=new Button("6");
            b7=new Button("7");
            b8=new Button("8");
            b9=new Button("9");

            this.add(b1);
            this.add(b2);
            this.add(b3);
            this.add(b4);
            this.add(b5);
            this.add(b6);
            this.add(b7);
            this.add(b8);
            this.add(b9);

            addWindowListener(new WindowAdapter()
            {
                  public void windowClosing(WindowEvent e)
                  {
                        System.exit(0);
                  }
```

```
            });

        }
        public static void main(String args[])
        {
            GLayoutDemo gld = new GLayoutDemo();
            gld.setTitle("My Buttons");
            gld.setSize(650,400);
            gld.setVisible(true);
        }
    }
```

Output:



4.14.3 Java 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.
- The following are the fileds of the FlowLayout class
    - public static final int LEFT
    - public static final int RIGHT
    - public static final int CENTER
    - public static final int LEADING
    - public static final int TRAILING
- The following are the constructors of the FlowLayout class
    - **FlowLayout():** creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap**.**
    - **FlowLayout(int align):** creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap**.**
    - **FlowLayout(int align, int hgap, int vgap):** creates a flow layout with the given alignment and the given horizontal and vertical gap.

Example:

```
import java.awt.*;
import java.awt.event.*;
class FLayoutDemo extends Frame
{
        Button b1,b2,b3,b4,b5,b6,b7,b8,b9;
        FLayoutDemo()
        {


            b1=new Button("1");
            b2=new Button("2");
            b3=new Button("3");
            b4=new Button("4");
            b5=new Button("5");
            b6=new Button("6");
            b7=new Button("7");
```

```
                b8=new Button("8");
                b9=new Button("9");

                this.add(b1);
                this.add(b2);
                this.add(b3);
                this.add(b4);
                this.add(b5);
                this.add(b6);
                this.add(b7);
                this.add(b8);
                this.add(b9);
                this.setLayout(new FlowLayout(FlowLayout.RIGHT));

                addWindowListener(new WindowAdapter()
                {
                      public void windowClosing(WindowEvent e)
                      {
                            System.exit(0);
                      }
                });

          }
          public static void main(String args[])
          {
                FLayoutDemo fld = new FLayoutDemo();
                fld.setTitle("My Buttons");
                fld.setSize(650,400);
                fld.setVisible(true);
          }
    }
```

Output:

4.14.4 Java CardLayout:

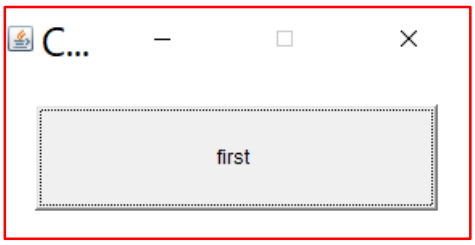
- The CardLayout class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.
- The following are the constructors of the CardLayout class
    - **CardLayout():** creates a card layout with zero horizontal and vertical gap.
    - **CardLayout(int hgap, int vgap):** creates a card layout with the given horizontal and vertical gap.
- The following are the methods of the 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:

```
import java.awt.*;
import java.awt.event.*;
class CardLayoutExample extends Frame implements ActionListener
{
      CardLayout card = new CardLayout(20,20);
      CardLayoutExample()
      {
          setLayout(card);
          Button Btnfirst = new Button("first ");
          Button BtnSecond = new Button ("Second");
          Button BtnThird = new Button("Third");
          add(Btnfirst,"Card1");
          add(BtnSecond,"Card2");
          add(BtnThird,"Card3");
          Btnfirst.addActionListener(this);
          BtnSecond.addActionListener (this);
          BtnThird.addActionListener(this);
             addWindowListener(new WindowAdapter()
            {
                  public void windowClosing(WindowEvent e)
                  {
                        System.exit(0);
                  }
            });
      }
      public void actionPerformed(ActionEvent e)
      {
        card.next(this);
      }

}
   class CardLayoutJavaExample
   {
        public static void main(String args[])
       {
            CardLayoutExample frame = new CardLayoutExample();
            frame.setTitle("CardLayout in Java Example");
            frame.setSize(220,150);
            frame.setResizable(false);
            frame.setVisible(true);
       }
   }
```

Output:



4.14.5 Java GridBagLayout:

- The Java GridBagLayout class is used to align components vertically, horizontally or along their baseline.

- The components may not be of same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells. Each component occupies one or more cells known as its display area. Each component associates an instance of GridBagConstraints. With the help of constraints object we arrange component's display area on the grid. The GridBagLayout manages each component's minimum and preferred sizes in order to determine component's size.

Example:

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

public class GridBagLayoutExample extends Frame
{
    public static void main(String[] args)
   {
        GridBagLayoutExample a = new GridBagLayoutExample();
        a.setTitle("Grid Bag Layout Demo");
        a.setSize(300, 300);
        //setPreferredSize(getSize());
        a.setVisible(true);
    }
    public GridBagLayoutExample()
   {
        GridBagLayout grid = new GridBagLayout();
        GridBagConstraints gbc = new GridBagConstraints();
        setLayout(grid);

        GridBagLayout layout = new GridBagLayout();
        this.setLayout(layout);
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.gridx = 0;
        gbc.gridy = 0;
        this.add(new Button("Button One"), gbc);
        gbc.gridx = 1;
        gbc.gridy = 0;
        this.add(new Button("Button two"), gbc);
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.ipady = 20;
        gbc.gridx = 0;
        gbc.gridy = 1;
        this.add(new Button("Button Three"), gbc);
        gbc.gridx = 1;
        gbc.gridy = 1;
        this.add(new Button("Button Four"), gbc);
        gbc.gridx = 0;
        gbc.gridy = 2;
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.gridwidth = 2;
        this.add(new Button("Button Five"), gbc);

        addWindowListener(new WindowAdapter()
        {
              public void windowClosing(WindowEvent e)
              {
                    System.exit(0);
              }
        });
    }
}
```

Output:

4.15 Check Boxes:

- Check box is a square shaped box which displays an option to the user. The user can select one or more options from a group of check boxes.
- To create a check box, we can create an object to Checkbox class, as:

```
Checkbox cb = new Checkbox(); //Checkbox without label
Checkbox cb = new Checkbox("label"); //Checkbox with label
Checkbox cb = new Checkbox("label",state);
/*if state is true, then the check box appears as if it is
selected by default, else not selected*/
```

- To get the state of a check box, use getState ():

```
boolean b = ch.getState( );// method returns true if selected
```

- To set the label of a check box use the following method:

```
Void setLabel(String label);
```

- To get the label of a check box:

```
String s = ch.getLabel();
```

- The following program creates three check boxes to display Bold, Italic and Underline to the user:

Example:

```
import java.awt.*;
import java.awt.event.*;
class MyCheckBox extends Frame implements ItemListener
{
      String msg="";
      Checkbox c1,c2,c3;
      MyCheckBox()
      {
            setLayout(new FlowLayout());

            c1=new Checkbox("Bold",true);
            c2=new Checkbox("Italic");
            c3=new Checkbox("Underline");

            add(c1);
            add(c2);
            add(c3);

            c1.addItemListener(this);
            c2.addItemListener(this);
            c3.addItemListener(this);

            addWindowListener(new WindowAdapter()
            {
                  public void windowClosing(WindowEvent e)
                  {
                        System.exit(0);
                  }
            });

      }
      public void itemStateChanged(ItemEvent ie)
      {
            repaint();
```

```
        }
        public void paint(Graphics g)
        {
              g.drawString("Current State: ",10,100);
              msg = "Bold: "+ c1.getState();
              g.drawString(msg,10,120);
              msg = "Italic: "+ c2.getState();
              g.drawString(msg,10,140);
              msg = "Underline: "+ c3.getState();
              g.drawString(msg,10,160);
        }
        public static void main(String args[])
        {
              MyCheckBox mc = new MyCheckBox();
              mc.setTitle("My Checkbox");
              mc.setSize(400,400);
              mc.setVisible(true);
        }
  }
```

Output:

4.16 Radio Button:

- A radio button represents a round shaped button, such that only one can be selected from a group of buttons.
- Radio buttons can be created using CheckboxGroup class and Checkbox classes.
- First of all, we should create a CheckboxGroup class object. While creating a radio button, we should pass CheckboxGroup object to the Checkbox class. It represents the group to which the radio button belongs.
- When the same CheckboxGroup object is passed to different radio buttons, then all radio buttons will be considered as belonging to same group and hence the user is allowed to select only one from them.
- To create a radio button, pass CheckboxGroup object to Checkbox class object as:

```
CheckboxGroup cbg = new CheckboxGroup();
Checkbox cb = new Checkbox("label",cbg,state);
/*if state is true, then the radio button appears to be already
selected.*/
```

- To know which radio button is selected by the user:

```
Checkbox cb = cbg.getSelectedCheckbox( );
```

- To know the selected radio button's label:

```
String label = cbg.getSelectedCheckbox( ).getLabel();
```

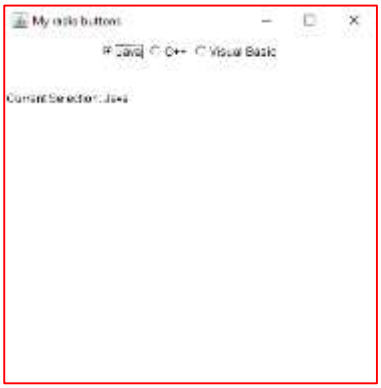
- The following program creates three radio buttons and displays its label when selected:

Example:

```
import java.awt.*;
import java.awt.event.*;
class MyRadio extends Frame implements ItemListener
{
      String msg="";
      Checkbox c1,c2,c3;
```

```
        CheckboxGroup cbg;

        MyRadio()
        {
              setLayout(new FlowLayout());

              cbg=new CheckboxGroup();

              c1=new Checkbox("Java",cbg,true);
              c2=new Checkbox("C++",cbg,false);
              c3=new Checkbox("Visual Basic",cbg,false);

              add(c1);
              add(c2);
              add(c3);

              c1.addItemListener(this);
              c2.addItemListener(this);
              c3.addItemListener(this);

              addWindowListener(new WindowAdapter()
              {
                    public void windowClosing(WindowEvent e)
                    {
                          System.exit(0);
                    }
              });

        }
        public void itemStateChanged(ItemEvent ie)
        {
              repaint();
        }
        public void paint(Graphics g)
        {
              msg = "Current Selection: ";
              msg += cbg.getSelectedCheckbox().getLabel();
              g.drawString(msg,10,100);
        }
        public static void main(String args[])
        {
              MyRadio mr = new MyRadio();
              mr.setTitle("My radio buttons");
              mr.setSize(400,400);
              mr.setVisible(true);
        }
  }
```

Output:



4.17 TextField:

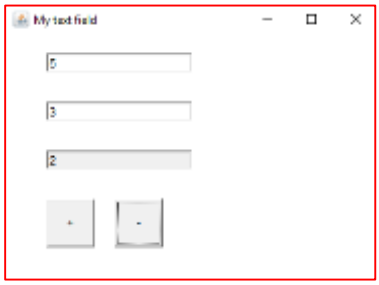
- A TextField represents a long rectangular box where the user can type a single line of text.
- To create a TextField:
  - `TextField tf = new TextField(); //creates a blank text field`
    `TextField tf = new TextField(25);`
    `//creates a blank text field with width 25 character long`
    `TextField tf = new TextField("Hello",25);`
    `//creates a text field of width 25 character with a default value Hello`
- To retrieve text from a TextField:
  `String s = tf.getText();`
- To set a text to a TextField:
  `tf.setText("Text");`
- To hide the text to a TextField:
  `tf.setEchoChar('*');//Hides the text with character *`
- The following program creates two text fields and two push with label '+' and '–'. When the button is pressed it performs the appropriate operation as per the button pressed on the texts entered in the two text fields:

Example:

```
import java.awt.*;
import java.awt.event.*;
public class TextFieldExample extends Frame implements ActionListener
{
    TextField tf1,tf2,tf3;
    Button b1,b2;
    TextFieldExample()
     {
          setLayout(null);

          tf1=new TextField();
     tf1.setBounds(50,50,150,20);

          tf2=new TextField();
     tf2.setBounds(50,100,150,20);

          tf3=new TextField();
     tf3.setBounds(50,150,150,20);
     tf3.setEditable(false);

          b1=new Button("+");
     b1.setBounds(50,200,50,50);

          b2=new Button("-");
     b2.setBounds(120,200,50,50);

          b1.addActionListener(this);
     b2.addActionListener(this);

          add(tf1);add(tf2);add(tf3);add(b1);add(b2);

          addWindowListener(new WindowAdapter()
          {
               public void windowClosing(WindowEvent e)
               {
                    System.exit(0);
               }
          });

     }
     public void actionPerformed(ActionEvent e)
     {
        String s1=tf1.getText();
        String s2=tf2.getText();
```

```
            int a=Integer.parseInt(s1);
            int b=Integer.parseInt(s2);
            int c=0;
            if(e.getSource()==b1)
                c=a+b;
            else if(e.getSource()==b2)
                c=a-b;
            String result=String.valueOf(c);
            tf3.setText(result);
        }
        public static void main(String args[])
        {
            TextFieldExample tf = new TextFieldExample();
            tf.setTitle("My text field");
            tf.setSize(400,400);
            tf.setVisible(true);
        }
    }
```

Output:



4.18TextArea:

- A TextArea is similar to a text field, but it can accommodate several lines of text.
- To create a TextArea:

```
TextArea ta = new TextArea(); //creates an empty text area
TextArea ta = new TextArea(rows,cols);
//creates a blank text area with some rows and cols
TextArea ta = new TextArea("str");
//creates a text area with predefined string
```

- To retrieve text from a TextArea:

```
String s = ta.getText();
```

- To set a text to a TextField:

```
ta.setText("text");
```

- To append the given text to the text areas current text:

```
ta.append("text");
```

- To insert the specified text at the specified position in this text area:

```
ta.insert("text",position);
```

4.19 Label:

- A Label is a constant text that is generally displayed along with a TextField or TextArea.
- When the label is displayed, there would be some rectangular area allotted for the label. In this area, the label is aligned towards right, left, or center as per the alignment constant.
- Label alignment can be either Label.CENTER, Label.LEFT or Label.RIGHT.
- To create a label:

```
Label la = new Label(); //creates an empty label
Label la = new Label("text");//creates label with the specified
text.
Label la = new Label("text");
//creates a label with the specified text with the specified
allignment.
```

- To retrieve text from a Label:

```
String s = la.getText();
```

- To set a text to a Label:

```
la.setText("text");
```

- To get the current alignment of the label:
  `int l = la.getAlignment();`
- To set the alignment for this label to the specified alignment.:
  `la.setAlignment(int alignment);`
- The following program creates a TextArea, a Push Button and two Labels. When we push the button it counts the number of character and words and displays the result as a label.

Example:

```
import java.awt.*;
import java.awt.event.*;
public class TextAreaExample extends Frame implements ActionListener
{
     Label l1,l2;
     TextArea area;
     Button b;
     TextAreaExample()
     {
          setLayout(null);

          l1=new Label();
          l1.setBounds(50,50,100,30);

          l2=new Label();
          l2.setBounds(160,50,100,30);

          area=new TextArea();
          area.setBounds(20,100,300,300);

          b=new Button("Count");
          b.setBounds(100,400,100,30);

          b.addActionListener(this);
          add(l1);add(l2);add(area);add(b);

          addWindowListener(new WindowAdapter()
          {
               public void windowClosing(WindowEvent e)
               {
                    System.exit(0);
               }
          });

     }
     public void actionPerformed(ActionEvent e)
     {
          String text=area.getText();
          String words[]=text.split("\\s");
          l1.setText("Words: "+words.length);
          l2.setText("Characters: "+text.length());
     }
     public static void main(String args[])
     {
          TextAreaExample ta = new TextAreaExample();
          ta.setTitle("My text field");
          ta.setSize(600,600);
          ta.setVisible(true);
     }
}
```

Output:

4.20 Choice Class:

- Choice class is useful to display a choice menu. It is a pop-up list of items and the user can select only one item from the available items.
- To create a Choice menu:
  ```
  Choice ch = new Choice(); //creates an empty choice menu
  ```
- Once a choice menu is created, we should add items to it using add () method, as:
  ```
  ch.add("item");
  ```
- To know the name of the item selected from the Choice menu:
  ```
  String s = ch.getSelectedItem( );
  ```
- Index of items in the Choice menu starts from 0 onwards. To know the index of the currently selected item:
  ```
  int i = ch.getSelectedIndex( ); // returns –1 if no item selected.
  ```
- To get the item string, given the item index number:
  ```
  String item = ch.getItem(int index);
  ```
- To know the number of items in the choice menu:
  ```
  int n = ch.getItemCount( );
  ```
- To remove an item from the choice menu at a specified position:
  ```
  ch.remove(int position);
  ```
- To remove an item from the choice menu:
  ```
  ch.remove(String item);
  ```
- To remove all items from the choice menu:
  ```
  ch.removeAll();
  ```
- The following program creates a TextArea, a Push Button and two Labels. When we push the button it counts the number of character and words and displays the result as a label.

Example:

```
import java.awt.*;
import java.awt.event.*;
class ChoiceExample extends Frame implements ItemListener
{
     String msg;
     Choice ch1,ch2;
     ChoiceExample()
     {
     setLayout(new FlowLayout());

          ch1=new Choice();
     ch1.add("C");
     ch1.add("C++");
     ch1.add("Java");
     ch1.add("PHP");

          ch2=new Choice();
     ch2.add("Turbo C++");
     ch2.add("Spring");
```

```
            ch2.add("Hibernate");
            ch2.add("CodeIgniter");

            add(ch1);
                  add(ch2);

                  ch1.addItemListener(this);
            ch2.addItemListener(this);

                  addWindowListener(new WindowAdapter()
                  {
                        public void windowClosing(WindowEvent e)
                        {
                              System.exit(0);
                        }
                  });
            }
            public void itemStateChanged(ItemEvent ie)
            {
                  repaint();
            }
            public void paint(Graphics g)
            {
                  g.drawString("Selected language: ",20,100);
                  msg=ch1.getSelectedItem();
                  g.drawString(msg,20,120);

                  g.drawString("Selected Framework: ",20,140);
                  msg=ch2.getSelectedItem();
                  g.drawString(msg,20,160);
            }
            public static void main(String args[])
            {
                  ListExample ta = new ListExample();
                  ta.setTitle("My Choice Box");
                  ta.setSize(600,400);
                  ta.setVisible(true);
            }
      }
```

Output:

4.21 List Class:

- A list box presents the user with a scrolling list of text items. The user can select one or more items from the list box.
- To create a list box, we can create an object to List class:

```
List lst = new List();
//creates a list box from which the user can select only one item.
List lst = new List(3);
//creates a list box which displays initially 3 rows. The rest of the rows can be seen by clicking on the scroll button.
List lst = new List(3,true);
//creates a list box which displays initially 3 items. The next parameter true represents that the user can select more than one item from the available items. If it is false, then the user can select only one item.
```

- To add items to the list box, we can use add () method, as:
  `lst.add("item");`
- To get all the selected items from the list box:
  `String s[] = lst.getSelectedItems();`
- To get a single selected items from the list box:
  `String s = lst.getSelectedItem( );`
- To get the selected items' position numbers:
  `int x[] = lst.getSelectedIndexes ( );`
- To get a single selected item's position numbers:
  `int x = lst.getSelectedIndex( );`
- To get the number of visible lines (items) in this list:
  `int x = lst.getRows( );`
- To get all the items available in the list box:
  `String s[] = lst.getItems( );`
- To know how many number of items are there in the list box:
  `int x = lst.getItemCount( );`
- To remove an item at aspecified position from the list:
  `lst.remove(int position);`
- To remove an item whose name is given:
  `lst.remove(String item);`
- To remove all items from the choice menu:
  `lst.removeAll();`
- The following program creates a list box with names of some languages from where the user can select one or more items.

Example:

```
import java.awt.*;
import java.awt.event.*;
class ListDemo extends Frame implements ItemListener
{
     int msg[];
     List lst;
    ListDemo()
     {
     setLayout(new FlowLayout());

          lst=new List(4,true);
     lst.add("English");
     lst.add("Hindi");
     lst.add("Oriya");
     lst.add("Sanskrit");
          lst.add("French");

          add(lst);

          lst.addItemListener(this);

          addWindowListener(new WindowAdapter()
          {
               public void windowClosing(WindowEvent e)
               {
                    System.exit(0);
               }
          });
     }
     public void itemStateChanged(ItemEvent ie)
     {
          repaint();
    }
     public void paint(Graphics g)
     {
          g.drawString("Selected Languages: ",100,200);
          msg=lst.getSelectedIndexes();
```

```
            for(int i=0;i<msg.length;i++)
            {
                  String item = lst.getItem(msg[i]);
                  g.drawString(item,100,220+i*20);
            }
      }
      public static void main(String args[])
      {
            ListDemo ld = new ListDemo();
            ld.setTitle("My Choice Box");
            ld.setSize(400,400);
            ld.setVisible(true);
      }
}
```

Output:

4.22 ScrollBar Class:

- Scrollbar class is useful to create scrollbars that can be attached to a frame or text area.
- Scrollbars are used to select continuous values between a specified minimum and maximum
- Scrollbars can be arranged vertically or horizontaliy.
- To create a scrollbar we can create an object to Scrollbar class, as:

```
Scrollbar sb = new Scrollbar (alignment,start,step,min,max);
Here, alignment: Scrollbar.VERTICAL or Scrollbar.HORIZONTAL
      start: starting value (eg: 0)
      step: step value(eg: 30) //represents scrollbar length
      min: minimum value(eg: 0)
      max: maximum value(eg: 300)
```

- To know the location of a scrollbar, we can use getValue () method that gives the position of the scrollbar in pixels, as:

```
int n = sb.getValue( );
```

- To update the scrollbar position to a new position, we can use setValue () method as

```
sb.setValue(int position);
```

- To get the maximum value of the scrollbar:

```
int x = sb.getMaximum( );
```

- To get the minimum value of the scrollbar:

```
int x = sb.getMinimum( );
```

- To get the alignment of the scrollbar:

```
int x = sb.getOriented( );//method returns 0 if it is aligned
HORIZONTAL else returns 1
```

- The following program creates three vertical scroll bars. Depending on their adjustment it sets the background color of the frame by considering the position as RGB values.

Example:

```
import java.awt.*;
import java.awt.event.*;
class MyScroll extends Frame implements AdjustmentListener
{
      String msg="";
      Scrollbar s1,s2,s3;
      MyScroll()
      {
```

```
        setLayout(null);

            s1=new Scrollbar(Scrollbar.VERTICAL,0,50,0,255);
            s1.setBounds(100,50,30,200);
            s2=new Scrollbar(Scrollbar.VERTICAL,0,50,0,255);
            s2.setBounds(200,50,30,200);
            s3=new Scrollbar(Scrollbar.VERTICAL,0,50,0,255);
            s3.setBounds(300,50,30,200);

            add(s1);
            add(s2);
            add(s3);

            s1.addAdjustmentListener(this);
      s2.addAdjustmentListener(this);
            s3.addAdjustmentListener(this);

            addWindowListener(new WindowAdapter()
            {
                  public void windowClosing(WindowEvent e)
                  {
                        System.exit(0);
                  }
            });
      }
      public void adjustmentValueChanged(AdjustmentEvent ae)
      {
            repaint();
     }
      public void paint(Graphics g)
      {
            int red,green,blue;
            red=s1.getValue();
            green=s2.getValue();
            blue=s3.getValue();
            Color c= new Color(red,green,blue);
            setBackground(c);
      }
      public static void main(String args[])
      {
            MyScroll ms = new MyScroll();
            ms.setTitle("My Scroll bar");
            ms.setSize(400,400);
            ms.setVisible(true);
      }
}
```

Output:

4.23 Panel Class:

- The Panel is a simplest container class. It provides space in which an application can attach any other component. It inherits the Container class.

- It doesn't have title bar.
- The following program demonstrates the use of Panel.

Example:

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

class PanelDemo extends Frame implements ActionListener
{
    Panel p,p1,p2,p3,p4;
    Button b1;
    TextField t;
    Label l1;
    List l;
    Checkbox ch1,ch2;
    CheckboxGroup g;
    Checkbox cc1,cc2;
    Choice c;
    TextArea a;

    PanelDemo()
    {
        p = new Panel();
        p1= new Panel();
        p2= new Panel();
        p3= new Panel();
        p4= new Panel();

        String c1[]= {"UNIX","JAVA","C","C++","CADRE"};

        cc1 = new Checkbox("1",true);
        cc2 = new Checkbox("2");

        b1 = new Button("Hello");
        l1 = new Label("Hello Label");
        t = new TextField(10);
        l = new List();
        g = new CheckboxGroup();
        ch1 = new Checkbox("Radiant",g,true);
        ch2 = new Checkbox("Lambant",g,false);

        a = new TextArea("" ,10,30,TextArea.SCROLLBARS_BOTH);
        c = new Choice();

        String s[] = {"One","Two","Three"};
        String msg;

        p.add(p1);
        p.add(p2);
        p.add(p3);
        p.add(p4);
        add(p);

        p1.setBackground(Color.red);
        p2.setBackground(Color.green);
        p3.setBackground(Color.blue);
        p4.setBackground(Color.cyan);

        p1.add(l1);
        p1.add(b1);
        p1.add(t);

        p2.add(l);
        p2.add(c);

        p3.add(ch1);
```

```
            p3.add(ch2);
            p3.add(cc1);
            p3.add(cc2);

            p4.add(a);

            setLayout(new BorderLayout());

            add(p1,BorderLayout.NORTH);
            add(p2,BorderLayout.SOUTH);
            add(p3,BorderLayout.EAST);
            add(p4,BorderLayout.WEST);

            int j;
            for(j=0;j<c1.length;j++)
            {
               c.add(c1[j]);
            }

            setBackground(Color.pink);
            a.setBackground(Color.white);
            t.setBackground(Color.yellow);

            b1.addActionListener(this);


            int i;
            for(i=0;i<s.length;i++)
            {
               l.add(s[i]);
            }
               addWindowListener(new WindowAdapter()
               {
                     public void windowClosing(WindowEvent e)
                     {
                           System.exit(0);
                     }
               });
        }
        public void actionPerformed(ActionEvent ae)
        {
            String msg = "Hello Silicon";

            if(ae.getSource() == b1)
            {
                t.setText(msg);
            }
        }
         public static void main(String args[])
         {
               PanelDemo pd = new PanelDemo();
               pd.setTitle("My Pane");
               pd.setSize(400,400);
               pd.setVisible(true);
         }
    }
```

Output:

4.24 Handling Keyboard Events:

- To know which key is pressed on the keyboard, we can take the help of KeyListener interface.
  This interface has the following methods:
  - `public void keyPressed (KeyEvent ke)`: This method is called when a key on the keyboard is pressed.
  - `public void keyTyped(KeyEvent ke):` This method is called when a key on the keyboard is typed. This method is applicable to the keys, which can be displayed and generally does not denote the special keys like function keys, control keys, shift keys, etc.
  - `public void keyReleased (KeyEvent ke):` This method is called when a key on the keyboard is released.
- The following program trap the key code and key name typed by the user on the keyboard and display them in a label. The program also counts the number of characters and words entered by the user in the text area. It shows the result immediately after the key is released.

Example:

```
import java.awt.*;
import java.awt.event.*;
public class KeyListenerExample extends Frame implements KeyListener
{
     Label l1,l2;
     TextArea area;
     String msg="";
     KeyListenerExample()
     {
     setLayout(null);
     l1=new Label();
     l1.setBounds(20,50,200,20);

          l2=new Label();
     l2.setBounds(20,100,200,20);

     area=new TextArea();
     area.setBounds(20,140,300, 300);
     area.addKeyListener(this);

     add(l1);
          add(l2);
          add(area);
          addWindowListener(new WindowAdapter()
          {
               public void windowClosing(WindowEvent e)
               {
                    System.exit(0);
               }
          });
     }
     public void keyPressed(KeyEvent e)
```

```
        {
              int keycode = e.getKeyCode();
              msg += "Key code: "+keycode;
              String keyname = e.getKeyText(keycode);
              msg += "   Key pressed: "+ keyname;
              l2.setText(msg);
              msg="";
        }
        public void keyReleased(KeyEvent e)
        {
           String text=area.getText();
           String words[]=text.split("\\s");
           l1.setText("Words:   "+   words.length   +"   Characters:"   +
                        text.length());
        }
        public void keyTyped(KeyEvent e) {}

        public static void main(String[] args)
        {
        KeyListenerExample kl = new KeyListenerExample();
              kl.setSize(400,400);
              kl.setVisible(true);
       }
  }
```

Output:

4.25 Handling Mouse Events:

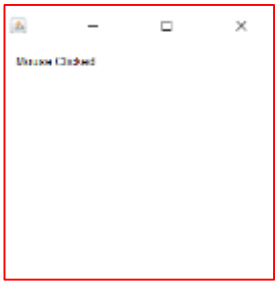
- Mouse events are of two types:
  - MouseListener handles the events when the mouse is not in motion.
  - MouseMotionListener handles the events when mouse is in motion.
- The Java MouseListener is notified whenever you change the state of mouse. It is notified against MouseEvent. The MouseListener interface is found in java.awt.event package.
- The following program demonstates MouseListener interface.

Example:

```
import java.awt.*;
import java.awt.event.*;
class MouseListenerExample extends Frame implements MouseListener
{
      Label l;
      MouseListenerExample()
      {
      addMouseListener(this);
      l=new Label();
      l.setBounds(20,50,100,20);
      add(l);
      setSize(300,300);
      setLayout(null);
      setVisible(true);
```

```
            addWindowListener(new WindowAdapter()
            {
                  public void windowClosing(WindowEvent e)
                  {
                        System.exit(0);
                  }
            });
      }
      public void mouseClicked(MouseEvent e) {
            l.setText("Mouse Clicked");
      }
      public void mouseEntered(MouseEvent e) {
      l.setText("Mouse Entered");
      }
      public void mouseExited(MouseEvent e) {
      l.setText("Mouse Exited");
      }
      public void mousePressed(MouseEvent e) {
      l.setText("Mouse Pressed");
      }
      public void mouseReleased(MouseEvent e) {
      l.setText("Mouse Released");
      }
      public static void main(String[] args) {
            new MouseListenerExample();
      }
}
```

Output:



- The Java MouseMotionListener is notified whenever you move or drag mouse. It is notified against MouseEvent. The MouseMotionListener interface is found in java.awt.event package.
- The following program demonstates MouseMotionListener interface.

Example:

```
import java.awt.*;
import java.awt.event.*;
class    MouseMotionListenerExample    extends    Frame    implements
MouseMotionListener
{
      MouseMotionListenerExample()
      {
      addMouseMotionListener(this);

      setSize(300,300);
      setLayout(null);
      setVisible(true);
            addWindowListener(new WindowAdapter()
            {
                  public void windowClosing(WindowEvent e)
                  {
                        System.exit(0);
                  }
            });
      }
      public void mouseDragged(MouseEvent e)
      {
```

```
            Graphics g=getGraphics();
            g.setColor(Color.BLUE);
            g.fillOval(e.getX(),e.getY(),20,20);
        }
        public void mouseMoved(MouseEvent e) {}

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

Output:

4.26 Handling Multiple Frames:

- In software development, it is necessary to create several frames to display some screens to the user and to accept user input.
- Suppose, we create a frame with the name 'Framel'. When a button like 'Next' is clicked on 'Framel' then it should display 'Frame2'. There is another button 'Back' in Frame2, which when clicked takes us back to the first frame. ·
- To create a new frame 'Frame2', we should write the following code in Framel class:

```
Frame2 f2 = new Frame2();
f2.setSize(400,400);
f2.setVisible(true);
```

- To make Frame2 terminate from memory, we can use dispose( ) method. To do this, write the following code in Frame2 class:

```
this.dispose();
```

- The following program create a frame 'Frame1' with Next and Close button. It also creates another frame 'Frame2' with back button such that when the user clicks Back Button, Frame2 is closed and we see the 'Frame1' only.

  NOTE: First comile Frame2, then complie Frame1 and then run Frame1.

Example:

Frame1

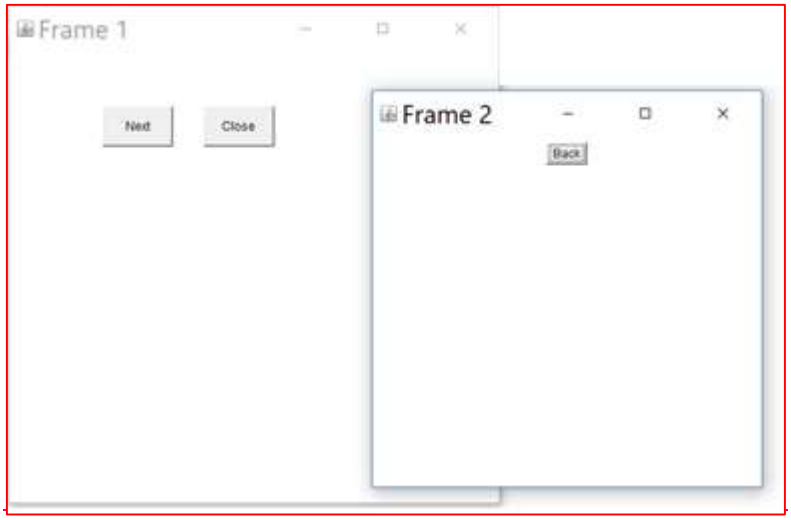
```
import java.awt.*;
import java.awt.event.*;
class Frame1 extends Frame implements ActionListener
{
    Button b1,b2;
    Frame1()
    {
    setLayout(null);
    b1=new Button("Next");
    b2=new Button("Close");

        b1.setBounds(100,100,70,40);
        b2.setBounds(200,100,70,40);

    add(b1);
        add(b2);

        b1.addActionListener(this);
        b2.addActionListener(this);

        addWindowListener(new WindowAdapter()
```

```
            {
                  public void windowClosing(WindowEvent e)
                  {
                        System.exit(0);
                  }
            });
      }
      public void actionPerformed(ActionEvent ae)
      {
            if(ae.getSource()== b1)
            {
                  Frame f2 = new Frame2();
                  f2.setSize(400,400);
                  f2.setVisible(true);
            }
            else
                  System.exit(0);
      }

      public static void main(String[] args)
      {
      Frame1 fl = new Frame1();
            fl.setTitle("Frame 1");
            fl.setSize(500,500);
            fl.setVisible(true);
     }
}
```

Frame2

```
import java.awt.*;
import java.awt.event.*;
class Frame2 extends Frame implements ActionListener
{
      Button b;
      Frame2()
      {
      setLayout(new FlowLayout());
      b=new Button("Back");

      add(b);

            b.addActionListener(this);

            addWindowListener(new WindowAdapter()
            {
                  public void windowClosing(WindowEvent e)
                  {
                        System.exit(0);
                  }
            });
      }
      public void actionPerformed(ActionEvent ae)
      {
            this.dispose();
      }
}
```

Output:



- It is possible to send some data from Framel to Frame2. Suppose we want to pass the roll number (10) and name 'Raj' of a student to Frame2. We can do that at the time of creating Frame2 object by passing the data as shown here:

```
Frame f2 = new Frame2(10,"Raj");
f2.setSize(400,400);
f2.setVisible(true);
```

- To receive the data in Frame2, we should take a parameterized constructor with two parameters which accepts the values coming from Framel as:

```
int rno;
String name;
Frame2(in rno,String name)
{
      this.rno = rno;
      this.name = name;
}
```

4.27 Calculator using AWT:

- The following program shows the development of a simple Calculator using AWT. The calculator can perform the operations like addition, multiplication etc. on two operands only (for eg. 2+3, 5*7 etc.). The application is not designed to handle more than two operands (like 2+3+5, 4+9+7 etc.)

Example:

```
import java.awt.*;
import java.awt.event.*;
class Calculator extends Frame implements ActionListener
{
      int c,n;
      String s1,s2,s3,s4,s5;
      Frame f;
    Button
b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11,b12,b13,b14,b15,b16,b17;
      Panel p;
      TextField tf;
      GridLayout g;
      Calculator()
      {
      p = new Panel();
      setLayout(new FlowLayout());

                  b1 = new Button("0");
      b1.addActionListener(this);
      b2 = new Button("1");
      b2.addActionListener(this);
      b3 = new Button("2");
      b3.addActionListener(this);
```

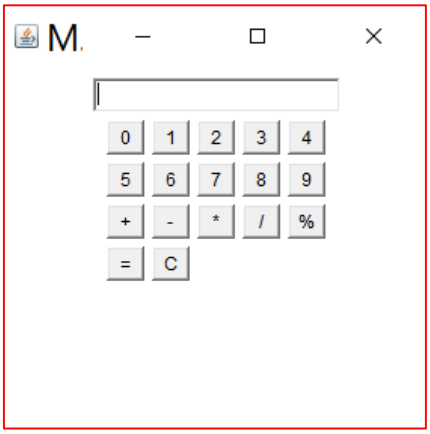
```
b4 = new Button("3");
b4.addActionListener(this);
b5 = new Button("4");
b5.addActionListener(this);
          b6 = new Button("5");
b6.addActionListener(this);
b7 = new Button("6");
b7.addActionListener(this);
b8 = new Button("7");
b8.addActionListener(this);
b9 = new Button("8");
b9.addActionListener(this);
          b10 = new Button("9");
b10.addActionListener(this);
b11 = new Button("+");
b11.addActionListener(this);
b12 = new Button("-");
b12.addActionListener(this);
b13 = new Button("*");
b13.addActionListener(this);
b14 = new Button("/");
b14.addActionListener(this);
b15 = new Button("%");
b15.addActionListener(this);
b16 = new Button("=");
b16.addActionListener(this);
b17 = new Button("C");
b17.addActionListener(this);
tf = new TextField(20);
add(tf);
g = new GridLayout(4,5,5,5);
p.setLayout(g);
          p.add(b1);p.add(b2);p.add(b3);p.add(b4);p.add(b5);
          p.add(b6);p.add(b7);p.add(b8);p.add(b9);
p.add(b10);p.add(b11);p.add(b12);p.add(b13);
          p.add(b14);p.add(b15);p.add(b16);p.add(b17);
          add(p);
      addWindowListener(new WindowAdapter()
      {
            public void windowClosing(WindowEvent e)
            {
                  System.exit(0);
            }
      });

}
public void actionPerformed(ActionEvent e)
{
    if(e.getSource()==b1)
    {
        s3 = tf.getText();
        s4 = "0";
        s5 = s3+s4;
        tf.setText(s5);
    }
    if(e.getSource()==b2)
    {
        s3 = tf.getText();
        s4 = "1";
        s5 = s3+s4;
        tf.setText(s5);
    }
    if(e.getSource()==b3)
    {
        s3 = tf.getText();
        s4 = "2";
```

```
        s5 = s3+s4;
        tf.setText(s5);
    }if(e.getSource()==b4)
    {
        s3 = tf.getText();
        s4 = "3";
        s5 = s3+s4;
        tf.setText(s5);
    }
    if(e.getSource()==b5)
    {
        s3 = tf.getText();
        s4 = "4";
        s5 = s3+s4;
        tf.setText(s5);
    }
    if(e.getSource()==b6)
    {
        s3 = tf.getText();
        s4 = "5";
        s5 = s3+s4;
        tf.setText(s5);
    }
    if(e.getSource()==b7)
    {
        s3 = tf.getText();
        s4 = "6";
        s5 = s3+s4;
        tf.setText(s5);
    }
    if(e.getSource()==b8)
    {
        s3 = tf.getText();
        s4 = "7";
        s5 = s3+s4;
        tf.setText(s5);
    }
    if(e.getSource()==b9)
    {
        s3 = tf.getText();
        s4 = "8";
        s5 = s3+s4;
        tf.setText(s5);
    }
    if(e.getSource()==b10)
    {
        s3 = tf.getText();
        s4 = "9";
        s5 = s3+s4;
        tf.setText(s5);
    }
    if(e.getSource()==b11)
    {
        s1 = tf.getText();
        tf.setText("");
        c=1;

    }
    if(e.getSource()==b12)
    {
        s1 = tf.getText();
        tf.setText("");
        c=2;

    }
    if(e.getSource()==b13)
```

```
        {
            s1 = tf.getText();
            tf.setText("");
            c=3;

        }
        if(e.getSource()==b14)
        {
            s1 = tf.getText();
            tf.setText("");
            c=4;

        }
        if(e.getSource()==b15)
        {
            s1 = tf.getText();
            tf.setText("");
            c=5;

        }
        if(e.getSource()==b16)
        {
            s2 = tf.getText();
            if(c==1)
            {
                n = Integer.parseInt(s1)+Integer.parseInt(s2);
                tf.setText(String.valueOf(n));
            }
            else if(c==2)
            {
                n = Integer.parseInt(s1)-Integer.parseInt(s2);
                tf.setText(String.valueOf(n));
            }
            else if(c==3)
            {
                n = Integer.parseInt(s1)*Integer.parseInt(s2);
                tf.setText(String.valueOf(n));
            }
            if(c==4)
            {
                try
                {
                    int p=Integer.parseInt(s2);
                    if(p!=0)
                    {
                      n = Integer.parseInt(s1)/Integer.parseInt(s2);
      tf.setText(String.valueOf(n));
                     }
                     else
                        tf.setText("infinite");

                }
                catch(Exception i){}
            }
            if(c==5)
            {
                n = Integer.parseInt(s1)%Integer.parseInt(s2);
                tf.setText(String.valueOf(n));
            }
        }
        if(e.getSource()==b17)
        {
            tf.setText("");
        }
    }
```

```
        public static void main(String[] abc)
        {
         Calculator f = new Calculator();
                    f.setTitle("My calculator");
                    f.setSize(300,300);
         f.setVisible(true);
        }
    }
```

Output:



5. SWING (JFC)

5.1 Introduction:

- Java Swing 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.
- Swing is a set of classes that provides more powerful and flexible GUI components than does the AWT.
- Because the AWT components use native code resources, they are referred to as heavyweight.
- Although Swing eliminates a number of the limitations inherent in the AWT, Swing does not replace it.
- Swing is built on the foundation of the AWT. This is why the AWT is still a crucial part of Java.
  Swing also uses the same event handling mechanism as the AWT. Therefore, a basic understanding of the AWT and of event handling is required to use Swing.
- Two of the most important Swing features:
  - **Swing components are lightweight:** This means that they are written entirely in Java and do not map directly to platform-specific peers. The look and feel of each component is determined by Swing, not by the underlying operating system. This means that each component will work in a consistent manner across all platforms.
  - **Swing supports a pluggable look and feel:** It is possible to "plug in" a new look and feel for any given component without creating any side effects in the code that uses that component. The look and feel can be changed dynamically at run time.
- The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

5.2 Difference between AWT and Swing:

Java AWT	Java Swing
AWT components are platform-dependent.	Java swing components are platform-independent.
AWT components are heavyweight.	Swing components are lightweight.

AWT doesn't support pluggablelook and feel (allowing to change the look and feel of the graphical user interface at runtime.).	Swing supports pluggable look and feel.
AWT provides less components than Swing.	Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
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.

5.3 Java Foundation Classes (JFC):

- The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.
- The Java Foundation Classes (JFC) are a graphical framework for building portable Java-based graphical user interfaces (GUIs). JFC consists of the Abstract Window Toolkit (AWT), Swing and Java 2D. Together, they provide a consistent user interface for Java programs, regardless of whether the underlying user interface system is Windows, macOS or Linux.
- JFC is an extension of the original AWT. It contains classes. that are completely portable, since the entire JFC is developed in pure Java.
- Some of the notable features of JFC are as follows:
    - JFC components are light-weight, utilize minimum system resources. Their speed is comparatively good and hence JFC programs execute much faster.
    - JFC components have same look-and-feel on all platforms. Once a component is created, it looks same on any Operating system. So the programmer can be sure of the look of his screen.
    - JFC offers 'pluggable look-and-feel' feature, which allows the programmer to change the look and feel as suited for a platform. Suppose, the programmer wants to display Windows-style push buttons on Windows operating system, and Unix-style buttons on Unix, it is possible.
    - JFC offers a rich set of cpmponents with lots of features.
    - JFC does not replace AWT. JFC is an extension to AWT. All classes of JFC are derived from AWT and hence all the methods available in AWT are also applicable in JFC.

5.4 Swing and MVC:

- A visual component is a composite of three distinct aspects:
    - The way that the component looks when rendered on the screen
    - The way that the component reacts to the user
    - The state information associated with the component.
- All the components of swing follow a model-view-controller (MVC) architecture.
- Model-view-controller (MVC) architecture:
    - **Model:** Represents the data that represents the state of a component. For example in a push button its state means whether the button is pressed or not or whether it is selected or not etc.
    - **View:** determines how the component is displayed on the screen, including any aspects of the view that are affected by the current state of the model.
    - **Controller:** determines how the component reacts to the user. For example, when the user clicks a check box, the controller reacts by changing the model to reflect the user's choice (checked or unchecked). This then results in the view being updated.

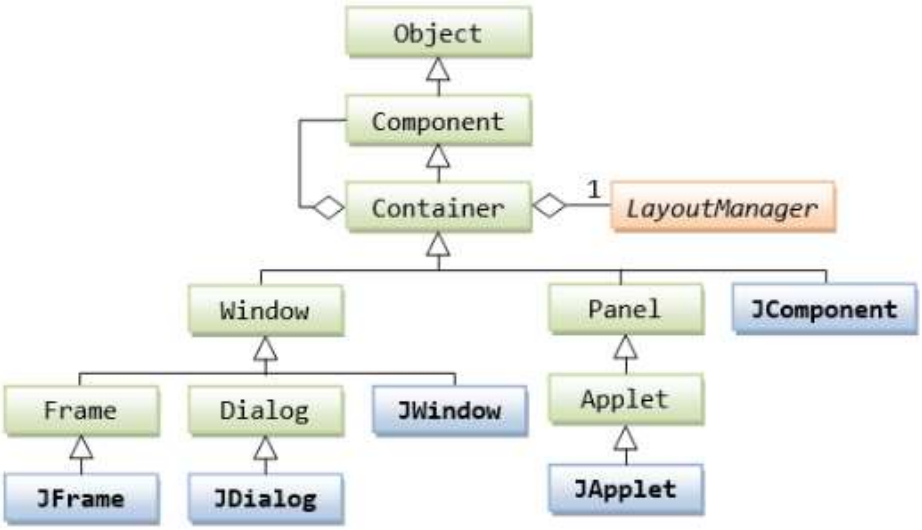
- The goal of MVC architecture is to separate the application object (model), its representation to the user (view), and the way it is controlled by the user (controller).

5.5 Swing Components and their Hierarchy:

- A component is an independent visual control.
- Swing Framework contains a large set of components which provide rich functionalities and allow high level of customization.
- Swing components are derived from the JComponent class.
- JComponent provides the functionality that is common to all components. For example, JComponent supports the pluggable look and feel.
- JComponent inherits the AWT classes Container and Component. Thus, a Swing component is built on and compatible with an AWT component.
- All of Swing's components are represented by classes defined within the package javax.swing.
- Note that all component classes begin with the letter J. For example, the class for a label is JLabel; the class for a push button is JButton; and the class for a scroll bar is JScrollBar.
- They all are derived from JComponent class. This class provides some common functionality like pluggable look and feel, support for accessibility, drag and drop, layout, etc.
- 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.

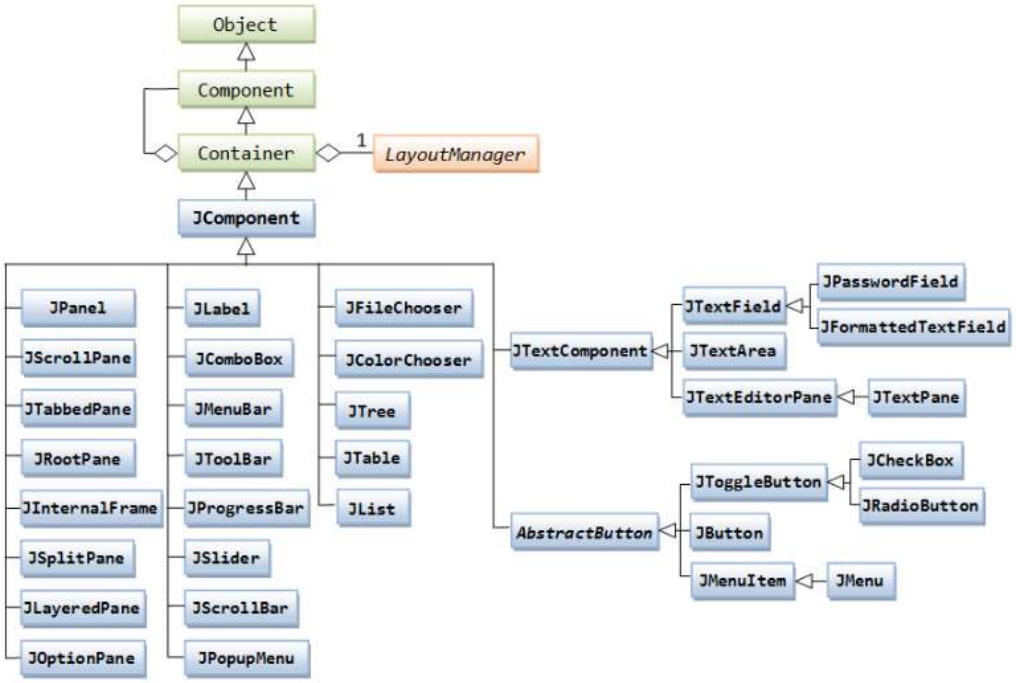
5.6 Swing Containers:

- A container holds a group of components. It provides a space where a component can be managed and displayed.
- A container is a special type of component that is designed to hold other components.
- In order for a component to be displayed, it must be held within a container.
- Containers are of two types:
  - **Top level Containers**
    - It inherits Component and Container of AWT.
    - A top-level container is not contained within any other container.
    - Heavyweight.
    - **Example:** JFrame, JDialog, JApplet, JWindow
    - The containment hierarchy must begin with a top-level container.
    - The one most commonly used for applications is JFrame. The one used for applets is JApplet



  - **Lightweight Containers**
    - It inherits JComponent class.
    - Lightweight
    - It is a general purpose container.

- Lightweight containers are often used to organize and manage groups of related components because a lightweight container can be contained within another container.
- It can be used to organize related components together.
- Example: JPanel



- The following table shows the class names for Swing components (including those used as containers).

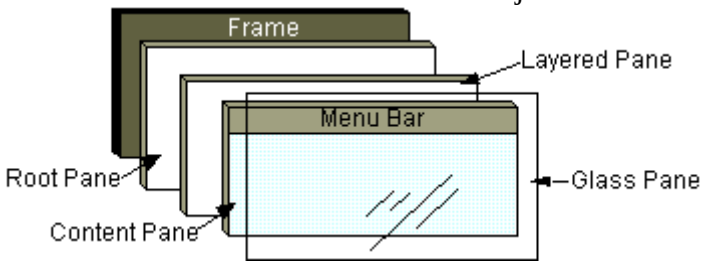
JApplet	JButton	JCheckBox	JCheckBoxMenuItem
JColorChooser	JComboBox	JComponent	JDesktopPane
JDialog	JEditorPane	JFileChooser	JFormattedTextField
JFrame	JInternalFrame	JLabel	JLayeredPane
JList	JMenu	JMenuBar	JMenuItem
JOptionPane	JPanel	JPasswordField	JPopupMenu
JProgressBar	JRadioButton	JRadioButtonMenuItem	JRootPane
JScrollBar	JScrollPane	JSeparator	JSlider
JSpinner	JSplitPane	JTabbedPane	JTable
JTextArea	JTextField	JTextPane	JTogglebutton
JToolBar	JToolTip	JTree	JViewport
JWindow			

5.7 Window Panes:

- A window pane represents a free area of a window where some text or components can be displayed. This free area is called 'window pane'.
- The panes can be imagined like like transparent sheets lying one below the other. So, if we attach any components to any they all will be finally displayed on the screen.
- We have four types of window panes available in javax.swing package.
  - **Root pane:** At the top of the hierarchy is an instance of JRootPane. JrootPane is a lightweight container whose purpose is to manage other panes. It also helps manage the optional menu bar. Root pane and glass pane are used in animation also. To go to the root pane, we can use getRootPane () method of Jframe class which returns an object of JrootPane class.
  - **Glass pane:** This is the the top-level pane. It sits above and completely covers all other panes. It is very close to the monitor's screen. Any component to be displayed in the foreground are attached to this glass pane. The glass pane enables you to manage mouse events that affect the entire container (rather than an individual control. To reach this

glass pane, we use getGlassPane () method of Jframe class. This method returns Component class object.

- **Layered pane:** The layered pane allows components to be given a depth value. This value determines which component overlays another. (Thus, the layered pane lets you specify a Z-order for a component. He layered pane holds the content pane and the optional menu bar. When we want to take several components as a group, we attach them in the layered pane. We can reach this pane by getLayeredPane () method of Jframe class which returns an object of JlayeredPane object.
- **Content pane:**Individual components are attached to this pane. Much of what this pane provide occurs behind the scene. To reach this pane, we can call getContentPane () method of Jframe class returns Container class object.



- In swing, the components are attached to the window panes only. For example, if we want to attach a push button (but object) to the content pane, first of all we should create content pane object by calling getContentPane () method as:

```
JFrame jf = new JFrame();                  // create Jframe object
Container c = jf.getContenetPane();    // creates a content pane
c.add(but);                                 // add button to content pane
```

5.8 Setting the Appearances and Properties of JComponents

- Most of the Swing Components supports these features:
  - Text and icon.
  - Keyboard short-cut (called mnemonics), e.g., activated via the "Alt" key in Windows System.
  - Tool tips: display when the mouse-pointer pauses on the component.
  - Look and feel: customized appearance and user interaction for the operating platform.
- All Jcomponents (such as Jpanel, Jlabel, JtextField and Jbutton) support these set methods to set their appearances and properties:

```
public void setBackground(Color bgColor)
// Sets the background color of this component
public void setForeground(Color fgcolor)
// Sets the foreground (text) color of this component
public void setFont(Font font)
// Sets the font used by this component
public void setBorder(Border border)
// Sets the border for this component
public void setPreferredSize(Dimension dim)
public void setMaximumSize(Dimension dim)
public void setMinimumSize(Dimension dim)
// Sets the preferred, maximum or minimum size of this component.
Public void setOpaque(booleanisOpaque)
// If true (opaque), fill the background with background color;
   // otherwise, enable transparent background.
   // Most of the Jcomponents have default of true, except Jlabel.
Public void setToolTipText(String toolTipMsg)
// Sets the tool-tip message, to be displayed when the mouse-pointer pauses over
the component.
```

- Swing's Jlabel and buttons (AbstractButton subclasses): support both text and icon, which can be specified in the constructor or via the setters.

```
// javax.swing.Jlabel, javax.swing.AbstractButton
public void setText(String strText)
// Set the text
public void setIcon(Icon defaultIcon)
// Set the button's default icon (you can have different icons for "pressed" and
"disabled" states)
public void setHorizontalAlignment(int alignment)
// Set the horizontal alignment of icon and text
   // SwingConstants.RIGHT, SwingConstants.LEFT, SwingConstants.CENTER
public void setVerticalAlignment(int alignment)
// Set the vertical alignment of icon and text,
   // SwingConstants.TOP, SwingConstants.BOTTOM, SwingConstants.CENTER
public void setHorizontalTextPosition(int textPosition)
// Set the horizontal text position relative to icon
   //       SwingConstants.RIGHT,       SwingConstants.LEFT,       SwingConstants.CENTER,
SwingConstants.LEADING, SwingConstants.TRAILING.
public void setVerticalTextPosition(int textPosition)
// Set the vertical text position relative to icon
   // SwingConstants.TOP, SwingConstants.BOTTOM, SwingConstants.CENTER
```

5.9 Creating a Frame in Swing:

- A frame represents a window with a title bar and borders. Frame becomes the basis for creating the screens for an application because all the components go into the frame.
- To create a frame, we have to create an object to JFrame class in swing, as:

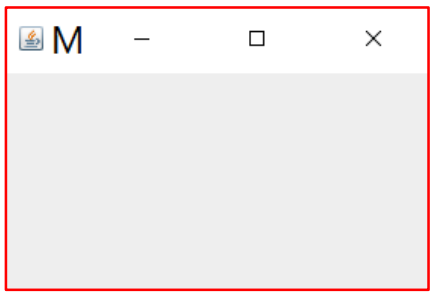
```
JFrame jf = new JFrame() ; //create a frame without any title
JFrame jf = new JFrame ("title"); //create frame with title
```

- The following program create a frame by creating an object to Jframe class:

Example:

```
import javax.swing.*;
class FrameDemo
{
      public static void main(String args[])
      {
            JFrame obj = new JFrame("My Frame");
            obj.setSize(200,200);
            obj.setVisible(true);
      }
}
```

OUTPUT:



- The frame create by the above program will be terminated from memory when the close button is clicked by the user, but the application will not terminate.
- To terminate the application forcibly press Control+C.
- To close the frame, we can take the help of getDefaultCloseOperation() method of Jframe class as shown here:

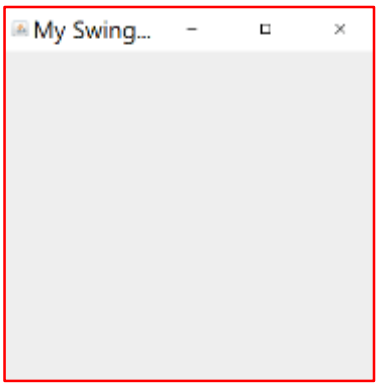
```
getDefaultCloseOperation(constant);
```

- The constant used above can be any one of the following:
  - `JFrame.EXIT_ON_CLOSE:` This closes the application upon clicking on close button.
  - `JFrame.DISPOSE_ON_CLOSE:`Dispose of the frame object, but keep the application running.
  - `JFrame.DO_NOTHING_ON_CLOSE:` This will not perform any operation upon clickingbutton.
  - `JFrame.HIDE_ON_CLOSE:` This hides the frame upon clicking on close button but keep the application running.
- The above program can be rewritten in the following way to show how to terminate an application by clicking on button of the frame.

Example:

```
import javax.swing.*;
class FrameDemo extends JFrame
{
      public static void main(String args[])
      {
            FrameDemo obj = new FrameDemo();
            obj.setTitle("My Swing Frame");
            obj.setSize(400,400);
            obj.setVisible(true);
            obj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      }
}
```

OUTPUT:

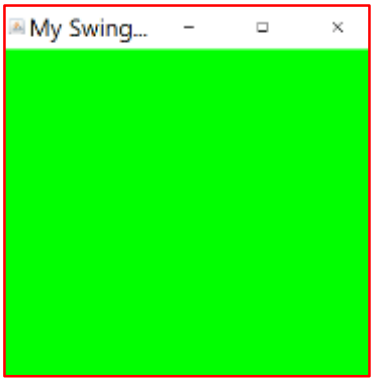


- To set the background color of a frame we should set the background color to content pane. The following program create the frame and display green color in its background:

Example:

```
import javax.swing.*;
import java.awt.*;
class FrameDemo extends JFrame
{
      public static void main(String args[])
      {
            FrameDemo obj = new FrameDemo();
            Container c = obj.getContentPane();
            c.setBackground(Color.green);
            obj.setTitle("My Swing Frame");
            obj.setSize(400,400);
            obj.setVisible(true);
            obj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      }
}
```

OUTPUT:



5.20 Displaying Text in Frame:

- Swing package provides us the following two ways to display text in an application:
  - **paintComponent (Graphics g)** method of JPanel class is used to paint the portion component in swing. We should override this method in our class. In the overridden paintComponentmethod we need to call the paintComponent() method of Jpanel class using **super.paintComponent()** which enable the super class to paint the component's area. The following program shows that.

Example:

```
import javax.swing.*;
import java.awt.*;
class MyPanel extends Jpanel
{
      MyPanel()
      {
            this.setBackground(Color.green);
      }
      public void paintComponent(Graphics g)
      {
            super.paintComponent(g);
            g.setColor(Color.red);
            g.setFont(new Font("Helvetica",Font.BOLD,34));
            g.drawString("Hello Silicon",50,100);
      }
}
class FrameDemo extends JFrame
{
      FrameDemo()
      {
            Container c = this.getContentPane();
            MyPanel mp = new MyPanel();
            c.add(mp);
      }
      public static void main(String args[])
      {
            FrameDemo obj = new FrameDemo();
            obj.setTitle("My Swing Frame");
            obj.setSize(400,400);
            obj.setVisible(true);
            obj.setDefaultCloseOperation(Jframe.EXIT_ON_CLOSE);
      }
}
```

OUTPUT:

- The second way to display some text in swing frame is by using a label. A label represents some constant text to be displayed in the frame. We can use Jlabel class to create a label as:

  Jlabel lbl = new Jlabel("text");

  The following program displays some text in the frame with the help of a label:

Example:

```
import javax.swing.*;
import java.awt.*;
class FrameDemo extends Jframe
{
      Jlabel lbl;
      FrameDemo()
      {
            Container c = this.getContentPane();
            c.setLayout(new FlowLayout());
            c.setBackground(Color.green);
            lbl=new Jlabel("Hello Silicon");
            lbl.setFont(new Font("Helvetica",Font.BOLD,34));
            lbl.setForeground(Color.red);
            c.add(lbl);
      }
      public static void main(String args[])
      {
            FrameDemo obj = new FrameDemo();
            obj.setTitle("My Swing Frame");
            obj.setSize(400,400);
            obj.setVisible(true);
            obj.setDefaultCloseOperation(Jframe.EXIT_ON_CLOSE);
      }
}
```

OUTPUT:

NOTE: If we are using a latest Java version (JDK 5 onwards) no need to use the create content pane object. We could invoke the add( ) method directly on a Jframe instance. Here, getContentPane( ) first obtains a reference to content pane, and then add( ) adds the component to the container linked to this pane. The above program can be written as the following way:

Example:

```
import javax.swing.*;
import java.awt.*;
```

```
class FrameDemo extends JFrame
{
	JLabel lbl;
	FrameDemo()
	{
		setLayout(new FlowLayout());
		setBackground(Color.green);
		lbl=new JLabel("Hello Silicon");
		lbl.setFont(new Font("Helvetica",Font.BOLD,34));
		lbl.setForeground(Color.red);
		add(lbl);
	}
	public static void main(String args[])
	{
		FrameDemo obj = new FrameDemo();
		obj.setTitle("My Swing Frame");
		obj.setSize(400,400);
		obj.setVisible(true);
		obj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	}
}
```

5.21 JLabel Class:

- Jlabel is a class of java Swing . Jlabel is used to display a short string or an image icon.
- Jlabel can display text, image or both. Jlabel is only a display of text or image and it cannot get focus.
- Jlabel is inactive to input events such a mouse focus or keyboard focus.
- By default labels are vertically centered but the user can change the alignment of label.
- Constructors of Jlabel class are:

  Jlabel() : creates a blank label with no text or image in it.
  Jlabel(String s) : creates a new label with the string specified.
  Jlabel(Icon i) : creates a new label with a image on it.
  Jlabel(String s, Icon i, int align) : creates a new label with a string, an image and a specified horizontal alignment
- Commonly used methods of the class are:

  getIcon() : returns the image that that the label displays
  setIcon(Icon i) : sets the icon that the label will display to image i
  getText() : returns the text that the label will display
  setText(String s) : sets the text that the label will display to string s
- The following program creates a blank label and add text to it.

Example:

```
import javax.swing.*;
import java.awt.*;
class FrameDemo extends JFrame
{
	JLabel lbl;
	FrameDemo()
	{
		setLayout(new FlowLayout());
		setBackground(Color.green);
		lbl=new JLabel();
		lbl.setFont(new Font("Helvetica",Font.BOLD,34));
		lbl.setText("Label Demo");
		add(lbl);
	}
	public static void main(String args[])
	{
		FrameDemo obj = new FrameDemo();
		obj.setTitle("My Swing Frame");
		obj.setSize(400,400);
		obj.setVisible(true);
		obj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	}
```

```
}
```

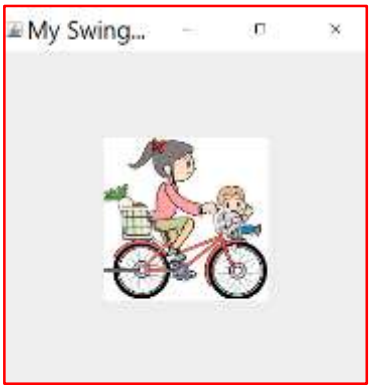
Output:

- The following program creates a blank label and an image to it.

Example:

```
import javax.swing.*;
import java.awt.*;
class FrameDemo extends JFrame
{
      JLabel lbl;
      FrameDemo()
      {
            // create a new image icon
      ImageIcon i = new ImageIcon("Cycle.gif");

      // create a label to display image
      lbl = new JLabel(i);
            add(lbl);
      }
      public static void main(String args[])
      {
            FrameDemo obj = new FrameDemo();
            obj.setTitle("My Swing Frame");
            obj.setSize(400,400);
            obj.setVisible(true);
            obj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      }
}
```

Output:



- The following program creates a blank label, add a image and string to it.

Example:

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
class FrameDemo extends JFrame
{
      JLabel lbl;
      FrameDemo()
```

```
        {
            // create a new image icon
           ImageIcon i = new ImageIcon("Cycle.gif");

           // create a label to display image
        lbl     =     new     JLabel("new     image     text",     i,
    SwingConstants.HORIZONTAL);

            add(lbl);
        }
        public static void main(String args[])
        {
            FrameDemo obj = new FrameDemo();
            obj.setTitle("My Swing Frame");
            obj.setSize(400,400);
            obj.setVisible(true);
            obj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
    }
```

Output:

5.22 JButton Class:

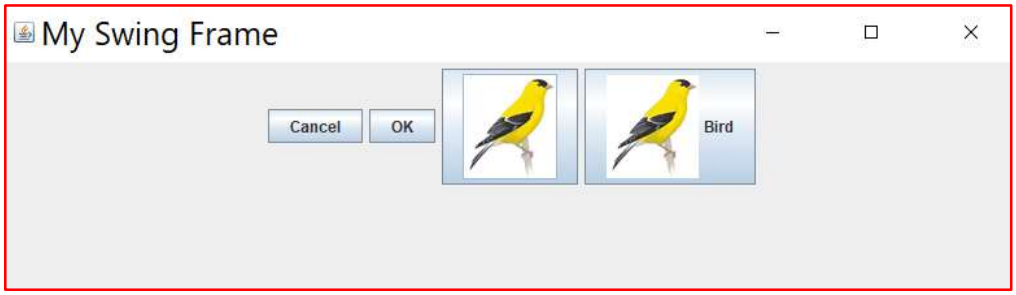
- The class **JButton** is an implementation of a push button. This component has a label and generates an event when pressed. It can also have an Image.
- Constructors of JButton class are:
  - **JButton () :** Creates a button with no set text or icon.
  - **JButton(Icon icon) :** Creates a button with an icon.
  - **JButton(String text) :** Creates a button with the text.
  - **JButton(String text, Icon icon) :** Creates a button with an initial text and an icon.
- Commonly used methods of the class JButton are:
  - **void setText(String s):** It is used to set specified text on button.
  - **String getText():** It is used to return the text of the button.
  - **Void setEnabled(86oolean b):** It is used to enable or disable the button.
  - **Void setIcon(Icon b):** It is used to set the specified Icon on the button.
  - **Icon getIcon():** It is used to get the Icon of the button.
  - **Void setMnemonic(int a):** It is used to set the mnemonic on the button.
  - **Void addActionListener(ActionListener a):** It is used to add the action listener to this object.
  - **setMnemonic(int mnemonic):** Set the keyboard mnemonic (i.e., the alt short-cut key).
- The following program creates four different buttons using four different constructors of JButton class.

Example:

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
class ButtonDemo extends JFrame
{
```

```
        JButton bl,b2,b3,b4;
        ButtonDemo()
        {
              setLayout(new FlowLayout());
              // create a new image icon
           ImageIcon i = new ImageIcon("Bird.png");

           // create a label to display image
           bl = new JButton();
              b2 = new JButton("OK");
              b3 = new JButton(i);
              b4 = new JButton("Bird",i);

              bl.setText("Cancel");
              add(bl);
              add(b2);
              add(b3);
              add(b4);
        }
        public static void main(String args[])
        {
              ButtonDemo obj = new ButtonDemo();
              obj.setTitle("My Swing Frame");
              obj.setSize(800,400);
              obj.setVisible(true);
              obj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
  }
```

Output:



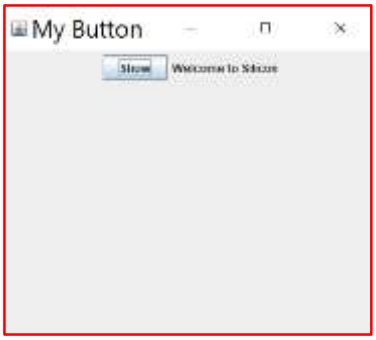
- The following program shows event handling of JButton class.

Example:

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
class ButtonDemo extends JFrame
{
      JButton b;
      JLabel l;
      ButtonDemo()
      {
            setLayout(new FlowLayout());
            b = new JButton("Show");
            l=new JLabel();
            add(b);
            add(l);

            b.addActionListener(new ActionListener()
            {
                  public void actionPerformed(ActionEvent e)
                  {
            l.setText("Welcome to Silicon");
            }
            });
      }
```

```
        public static void main(String args[])
        {
              ButtonDemo obj = new ButtonDemo();
              obj.setTitle("My Button");
              obj.setSize(400,400);
              obj.setVisible(true);
              obj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
    }
```

Output:



5.23 JTextField Class:

- The object of a JTextField class is a text component that allows the editing of a single line text. It inherits JTextComponent class and uses the interface SwingConstants.

- Constructors of JtextField class are:

  JTextField() : constructor that creates a new TextField.

  JTextField(int columns) : constructor that creates a new empty TextField with specified number of columns.

  JTextField(String text) : constructor that creates a new empty text field initialized with the given string.

  JTextField(String text, int columns) : constructor that creates a new empty textField with the given string and a specified number of columns.

  JTextField(Document doc, String text, int columns) : constructor that creates a textfield that uses the given text storage model and the given number of columns.

- Commonly used methods of the class JtextField are:

  setColumns(int n) : set the number of columns of the text field.
  setFont(Font f) : set the font of text displayed in text field.
  addActionListener(ActionListener l) : set an ActionListener to the text field.
  Int getColumns() : get the number of columns in the textfield.
  getText(): get the content of the text field.
  setText(): set the content of the text field.
  setToolTipText(String s): show the string s on the tool tip.
  selectAll(): Select all text
  setSelectionStart(int start): Start the selection from index start
  setSelectionEnd(int end): Ends the selection at index end
  setEditable(Boolean b): Enabling/disabling the textfield
  setHorizontalAlignment(position): Set horizontal alignment of text. Position can be JtextField.LEFT, JtextField.CENTER, JtextField.RIGHT, JtextField.LEADING, JtextField.TRAILING

- The following program creates a Textfield and a button. When you are typing some text in the text field and hit Enter, it should show the message in a dialog. If the field is empty, the OK button is disabled:

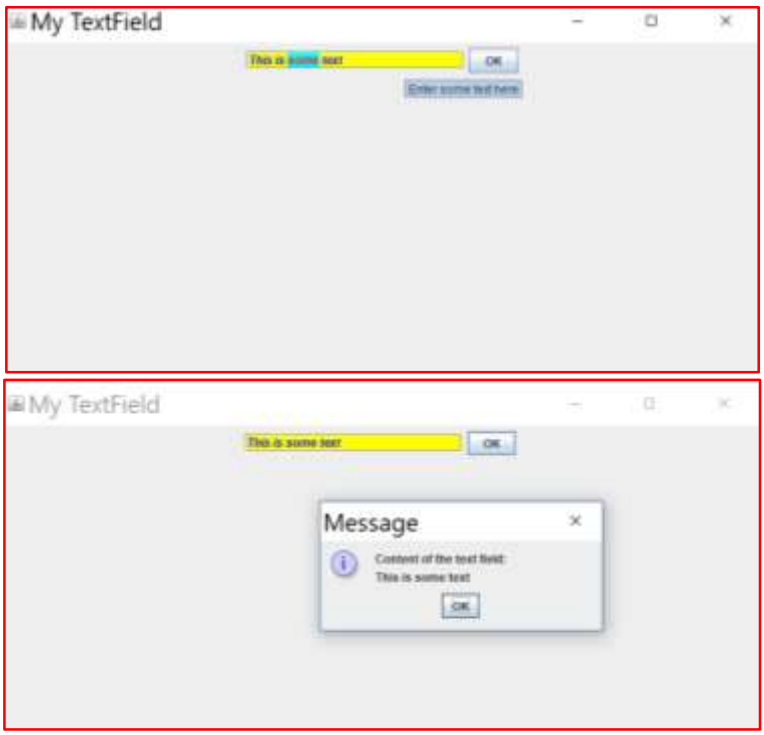
Example:

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
```

```
class TextFieldDemo extends JFrame
{
     JTextField textField;
     JButton button;
     TextFieldDemo()
     {
          setLayout(new FlowLayout());
          textField = new JTextField("This is some text", 20);
          button = new JButton("OK");
          // customizes appearance: font, foreground, background
          textField.setFont(new java.awt.Font("Arial", Font.ITALIC |
Font.BOLD, 12));
          textField.setForeground(Color.BLUE);
          textField.setBackground(Color.YELLOW);

          // customizes text selection
          textField.setSelectionColor(Color.CYAN);
          textField.setSelectedTextColor(Color.RED);

          // sets initial selection
          textField.setSelectionStart(8);
          textField.setSelectionEnd(12);

          //adding tooltip
          textField.setToolTipText("Enter some text here");

          // adds event listener which listens to Enter key event
          textField.addActionListener(new ActionListener()
          {
               public void actionPerformed(ActionEvent event)
{

     JOptionPane.showMessageDialog(TextFieldDemo.this,"You    entered
text:\n" + textField.getText());
               }
          });

          // adds key event listener
          textField.addKeyListener(new KeyAdapter()
          {
               public void keyReleased(KeyEvent event)
               {
                    String content = textField.getText();
                    if (!content.equals(""))
                    {
                         button.setEnabled(true);
                    } else
                    {
                         button.setEnabled(false);
                    }
               }
          });

          // adds action listener for the button
          button.addActionListener(new ActionListener()
          {
               public void actionPerformed(ActionEvent event)
               {

     JOptionPane.showMessageDialog(TextFieldDemo.this,"Content     of
the text field:\n" + textField.getText());
               }
          });

          add(textField);
          add(button);
```

```
        }
        public static void main(String args[])
        {
              TextFieldDemo obj = new TextFieldDemo();
              obj.setTitle("My TextField");
              obj.setSize(800,400);
              obj.setVisible(true);
              obj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
    }
```

Output:



5.24 JTextArea Class:

- JTextArea is a plain text area for displaying multiple lines of editable text, unformatted.
- All the texts are in the same font.
- Constructors of JTextArea class are:

  JTextArea(): constructs a new blank text area .
  JTextArea(String s): constructs a new text area with a given initial text.
  JTextArea(int row, int column): constructs a new text area with a given number of rows and columns.
  JTextArea(String s, int row, int column): constructs a new text area with a given number of rows and columns and a given initial text.
- Commonly used methods of the class JTextArea are:

  append(String s): appends the given string to the text of the text area.
  getLineCount(): get number of lines in the text of text area.
  setFont(Font f): sets the font of text area to the given font.
  setColumns(int c): sets the number of columns of the text area to given integer.
  setRows(int r): sets the number of rows of the text area to given integer.
  getColumns(): get the number of columns of text area.
  getRows(): get the number of rows of text area.
  append(String str): append the str to the end of the document

replaceRange(String str, int startPos, int endPos): replace with the str from startPos to endPos position

insert(String str, int pos): insert the str after the specified position

- Unlike TextArea component in AWT, if all the text cannot be displayed in the available space in the component, scrollbars are not automatically added. In order to add scrollbars, you must insert it into a ScrollPane.
- The following program creates four buttons, a text area and a label. When we click the submit button, the application should count the number of lines entered in the text area. By clicking on the button plain, bold or italic it applies the design on the text of text area.

Example:

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
class ButtonDemo extends JFrame implements ActionListener
{
     JButton b1,b2,b3,b;
     JLabel l1;
     JTextArea jt;
     ButtonDemo()
     {
          setLayout(new FlowLayout());

     l1 = new JLabel("0 lines");

     b = new JButton("submit");
     b1 = new JButton("plain");
     b2 = new JButton("italic");
     b3 = new JButton("bold");

          jt = new JTextArea("please write something ", 10, 10);

          add(b);
          add(b1);
          add(b2);
          add(b3);
          add(l1);
          add(jt);

          b.addActionListener(this);
     b1.addActionListener(this);
     b2.addActionListener(this);
     b3.addActionListener(this);
     }
     public void actionPerformed(ActionEvent e)
     {
     String s = e.getActionCommand();
          l1.setText(s);

          if (s.equals("submit"))
               l1.setText(jt.getLineCount() + " lines");
     else if (s.equals("bold"))
          {
               Font f = new Font("Serif", Font.BOLD, 15);
               jt.setFont(f);
     }
     else if (s.equals("italic"))
          {
     Font f = new Font("Serif", Font.ITALIC, 15);
               jt.setFont(f);
     }
     else if (s.equals("plain"))
          {
               Font f = new Font("Serif", Font.PLAIN, 15);
```

```
                    jt.setFont(f);
                }
            }

            public static void main(String args[])
            {
                  ButtonDemo obj = new ButtonDemo();
                  obj.setTitle("My TextArea");
                  obj.setSize(400,400);
                  obj.setVisible(true);
                  obj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            }
      }
```

Output:

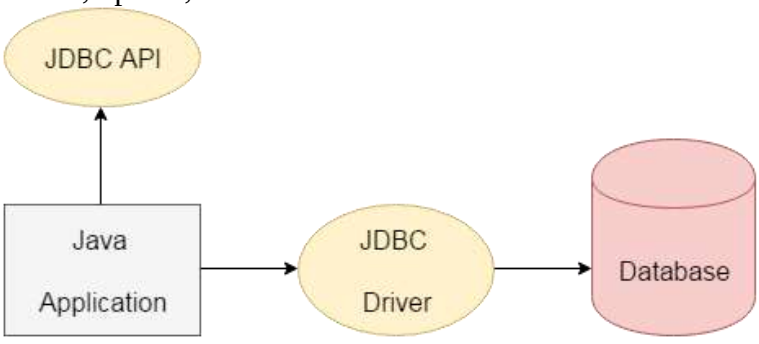
- It is common to wrap a JTextArea inside a JScrollPane, so as to scroll the text area (horizontally or vertically). To do so, allocate a JScrollPane with the JTextArea as the argument.

```
JTextArea tArea = new JTextArea(...);  // Allocate JTextArea
JScrollPane tAreaScrollPane = new JScrollPane(tArea); // Allocate JScrollPane which wraps the JTextArea
tAreaScrollPane.setVerticalScrollBarPolicy(...);  // Configure vertical scroll bar
tAreaScrollPane.setHorizontalScrollBarPolicy(...);//  Configure  horizontal  scroll bar
```

6. JDBC

6.1 Introduction:

- JDBC stands for Java Database Connectivity.
- JDBC is a Java API to connect and execute the query with the database.
- It is a part of JavaSE (Java Standard Edition).
- JDBC API uses JDBC drivers to connect with the database.
- There are four types of JDBC drivers:
  - JDBC-ODBC Bridge Driver
  - Native Driver
  - Network Protocol Driver
  - Thin Driver
- We can use JDBC API to access tabular data stored in any relational database. By the help of JDBC API, we can save, update, delete and fetch data from the database.



- The current version of JDBC is 4.3.
- The **java.sql** package contains classes and interfaces for JDBC API.
- **Why Should We Use JDBC?**
  - Before JDBC, ODBC API was the database API to connect and execute the query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).
  - We can use JDBC API to handle database using Java program and can perform the following activities:
    - Connect to the database
    - Execute queries and update statements to the database
    - Retrieve the result received from the database.

6.2JDBC Driver:

- JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:

6.2.1 JDBC-ODBC Bridge Driver

- The JDBC-ODBC bridge driver uses ODBC driver to connect to the database.
- The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls.
- This is now discouraged because of thin driver.
- Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.

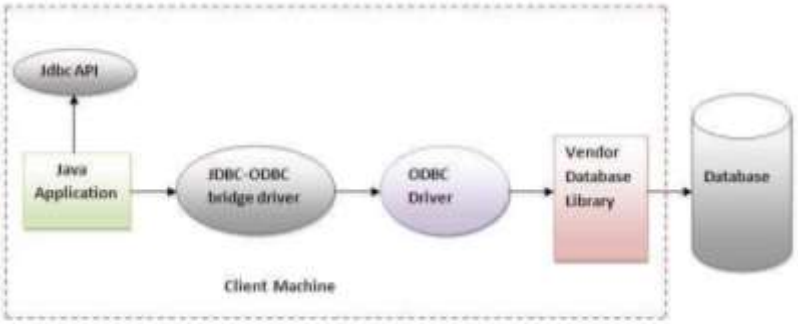


Figure- JDBC-ODBC Bridge Driver

- **Advantages:**
  - easy to use.
  - can be easily connected to any database.
- **Disadvantages:**
  - Performance degraded because JDBC method call is converted into the ODBC function calls.
  - The ODBC driver needs to be installed on the client machine.

6.2.2 Native-API Driver

- The Native API driver uses the client-side libraries of the database.
- The driver converts JDBC method calls into native calls of the database API.
- It is not written entirely in java.

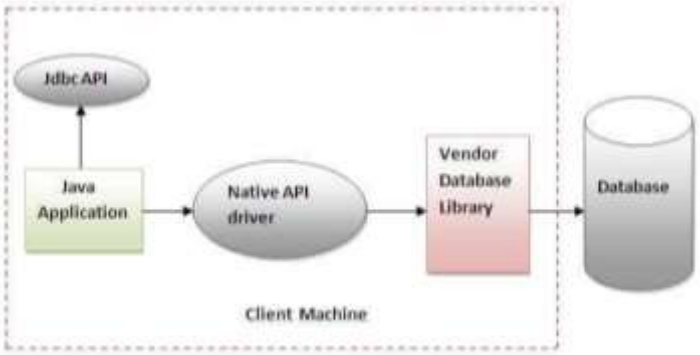


Figure- Native API Driver

- **Advantages:**
  - performance upgraded than JDBC-ODBC bridge driver..
- **Disadvantages:**
  - The Native driver needs to be installed on the each client machine.
  - The Vendor client library needs to be installed on client machine.

6.2.3 Network Protocol Driver

- The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol.
- It is fully written in java.

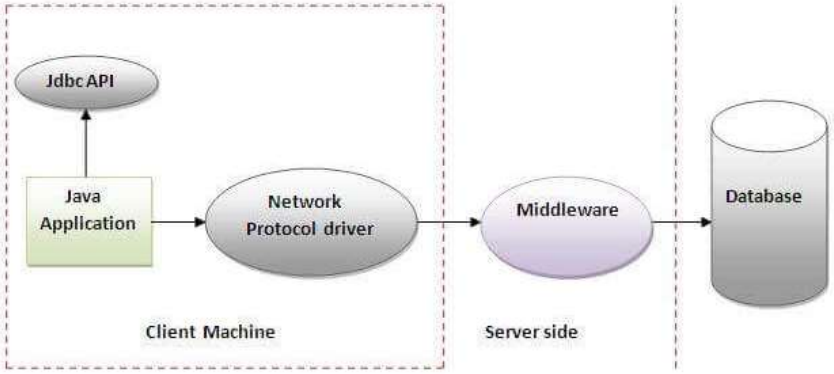


Figure- Network Protocol Driver

- **Advantages:**
  - No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.
- **Disadvantages:**
  - Network support is required on client machine.
  - Requires database-specific coding to be done in the middle tier.
  - Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

6.2.4 Thin Driver

- The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver.
- It is fully written in Java language.

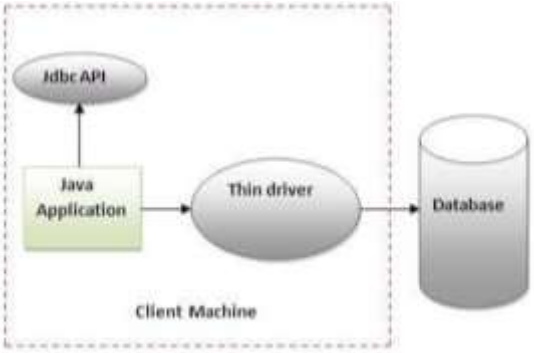


Figure- Thin Driver

- **Advantages:**
  - Better performance than all other drivers.
  - No software is required at client side or server side.
- **Disadvantages:**
  - Drivers depend on the Database.

6.3Java Database Connectivity:

- There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:

6.3.1 Register the Driver class

- The forName() method of Class class is used to register the driver class. This method is used to dynamically load the driver class.
- **Syntax** of forName() method

```
public static void forName(String className)throws ClassNotFoundException
```

- **Example** to register the OracleDriver class:

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

6.3.2 Create the connection object

- The getConnection() method of DriverManager class is used to establish connection with the database.
- **Syntax** of getConnection()method:

```
1) public static Connection getConnection(String url)throws SQLException
2) public static Connection getConnection(String url,String name,String password)throws SQLException
```

- **Example**establish connection with the Oracle database:

```
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","password");
```

6.3.3 Create the Statement object

- The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.
- **Syntax**of createStatement()method:

```
public Statement createStatement()throws SQLException
```

- **Example**to create the statement object

```
Statement stmt=con.createStatement();
```

6.3.4 Execute the query

- The executeQuery() method of Statement interface is used to execute queries to the database.
- This method returns the object of ResultSet that can be used to get all the records of a table.
- **Syntax** of executeQuery()method

```
public ResultSet executeQuery(String sql)throws SQLException
```

- **Example** to execute query:

```
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
{
        System.out.println(rs.getInt(1)+" "+rs.getString(2));
```

```
}
```

6.3.5Close the connection object

- By closing connection object statement and ResultSet will be closed automatically.
- The close() method of Connection interface is used to close the connection.
- **Syntax** of close() method

```
public void close()throws SQLException
```

- **Example** to to close connection:

```
con.close();
```

6.4Java Database Connectivity with Oracle:

- In this example, we are using Oracle 10g as the database.
- To connect java application with the oracle database, we need to know following information for the oracle database:
  - **Driver class:** The driver class for the oracle database is **oracle.jdbc.driver.OracleDriver**.
  - **Connection URL:** The connection URL for the oracle10G database is **jdbc:oracle:thin:@localhost:1521:xe** where jdbc is the API, oracle is the database, thin is the driver, localhost is the server name on which oracle is running, we may also use IP address, 1521 is the port number and xe is the Oracle service name. You may get all these information from the tnsnames ora file.
  - **Username:** The default username for the oracle database is **system**.
  - **Password:** It is the password given by the user at the time of installing the oracle database.
- **Create a Table**
  - Before establishing connection, let's first create a table in oracle database.
  - Following is the SQL query to create a table.

```
create table Student(roll number(10),name varchar2(40),age number(3));
```

- **Program 1:** In this example, we are connecting to an Oracle database and getting data from **emp** table. Here, **scott** and **tiger** are the username and password of the Oracle database. The description of the emp table is as follows:

Name	Type
EMPNO	NUMBER(4)
ENAME	VARCHAR2(10)
JOB	VARCHAR2(9)
MGR	NUMBER(4)
HIREDATE	DATE
SAL	NUMBER(7,2)
COMM	NUMBER(7,2)
DEPTNO	NUMBER(2)
AGE	NUMBER

```
import java.sql.*;
class OracleCon
{
      public static void main(String args[])
      {
            try
            {
                  //step1 load the driver class
                  Class.forName("oracle.jdbc.driver.OracleDriver");


                  //step2 create  the connection object
                  Connection con=DriverManager.getConnection
("jdbc:oracle:thin:@192.168.1.2:1521:silicon","scott","tiger");

//step3 create the statement object that can be used to execute SQL queries.
                  Statement stmt=con.createStatement();
```

```
      //# is used to execute SELECT query. It returns the object of
ResultSet
                  ResultSet rs=stmt.executeQuery("select * from emp");


                  //step4 execute query
                  while(rs.next())
                  System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"   "+
rs.getString(3));

                  //step5 close the connection object

                  con.close();

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

      }
}
```

- **Program 1:** In this example, we are connecting to an Oracle database and inserting data into**emp** table.

```
            import java.sql.*;
            import java.io.*;
            class RS
            {
            public static void main(String args[])throws Exception
            {
                  Class.forName("oracle.jdbc.driver.OracleDriver");
                  Connection
            con=DriverManager.getConnection("jdbc:oracle:thin:@192.168.1.2:1
            521:silicon","scott","tiger");

                  PreparedStatement                  ps=con.prepareStatement
            ("insert into emp values(?,?,?,?,?,?,?,?,?)");
                  ps.setInt(1,1111);
                  ps.setString(2,"Ajay");
                  ps.setString(3,"RAJ");
                  ps.setInt(4,8000);
                  ps.setString(5,"01.01.2019");
                  ps.setInt(6,5000);
                  ps.setInt(7,100);
                  ps.setInt(8,20);
                  ps.setInt(9,20);
                  int i=ps.executeUpdate();
                  con.close();
                  }
            }
```