Object Oriented Programming Using Java
Code: PCS3I102

3rd Semester B. Tech (CSE)

3rd Semester B. Tech (CSE)

(2018-19)

Bapuji Rao, B.Sc(Hons.), DISM, M.Sc(IT), M.Tech(CS), (Ph.D)

Mobile: +91 9437825480

IGIT, Sarang, Dhenkanal Dist., Odisha.

MODULE-3

Syllabus

Module 3: -

Chapter 1:- IO Streams (java.io package)

Introduction, Byte Stream and Character Stream, Files and Random Access Files, Serialization, Collection Frame Work (java.util), Introduction, Util Package interfaces, List, Set, Map etc, List interfaces and its classes, Setter interfaces and its classes.

Chapter 2:-Applet

Introduction, Life Cycle of an Applet, GUI with an Applet, Abstract Window Toolkit (AWT), Introduction to GUI, Description of Components and Containers, Component/Container hierarchy, Understanding different Components/Container classes and their constructors, Event Handling, Different mechanisms of Event Handling, Listener Interfaces, Adapter classes.

Module 4: -

Chapter 1:- Swing (JFC)

Introduction Diff b/w awt and swing, Components Hierarchy, Panes, Individual Swings Components JLabel, JButton, JTextField, JTextArea.

Chapter 2:- JavaFX

Getting started with JavaFX, Graphics, User Interface Components, Effects, Animation, and Media, Application Logic, Interoperability, JavaFX Scene Builder 2, Getting Started with scene Builder. Working with scene Builder.

FILES

To create a file in the disk, four things to be decided.

- Suitable name for a file.
- Data type to be stored.
- Purpose of the file (reading or writing or updating).
- Method of creation of file.

Naming a File

- It consists of string of character.
- The length of the file name and the characters are dependent on the OS on which the java program is executed.
- The file name may consists of both primary and an optional period with secondary or extension name.
- **Examples: ABC.Dat Text.doc student.txt Salary Numbers.dat**

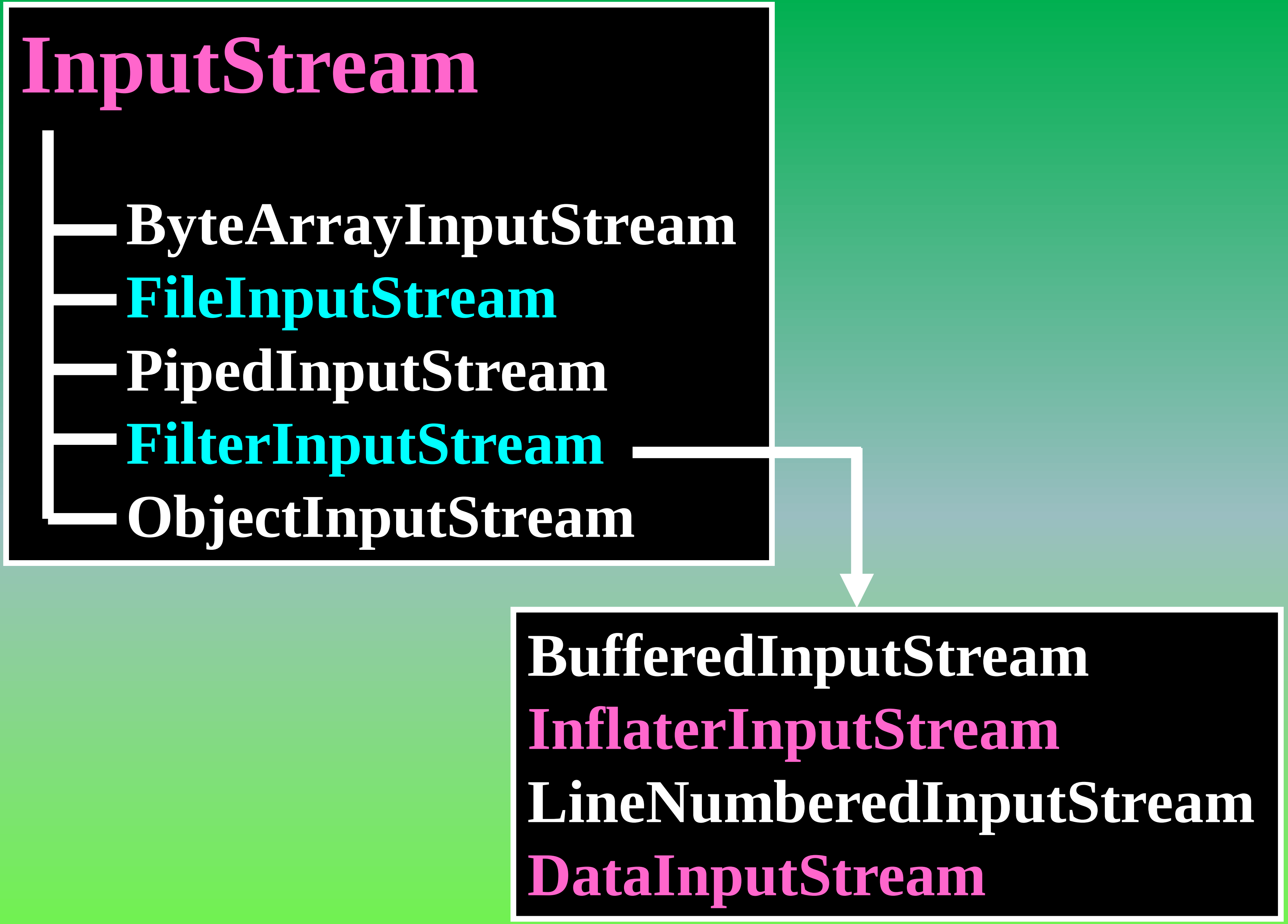
To create a file, following steps to be followed:

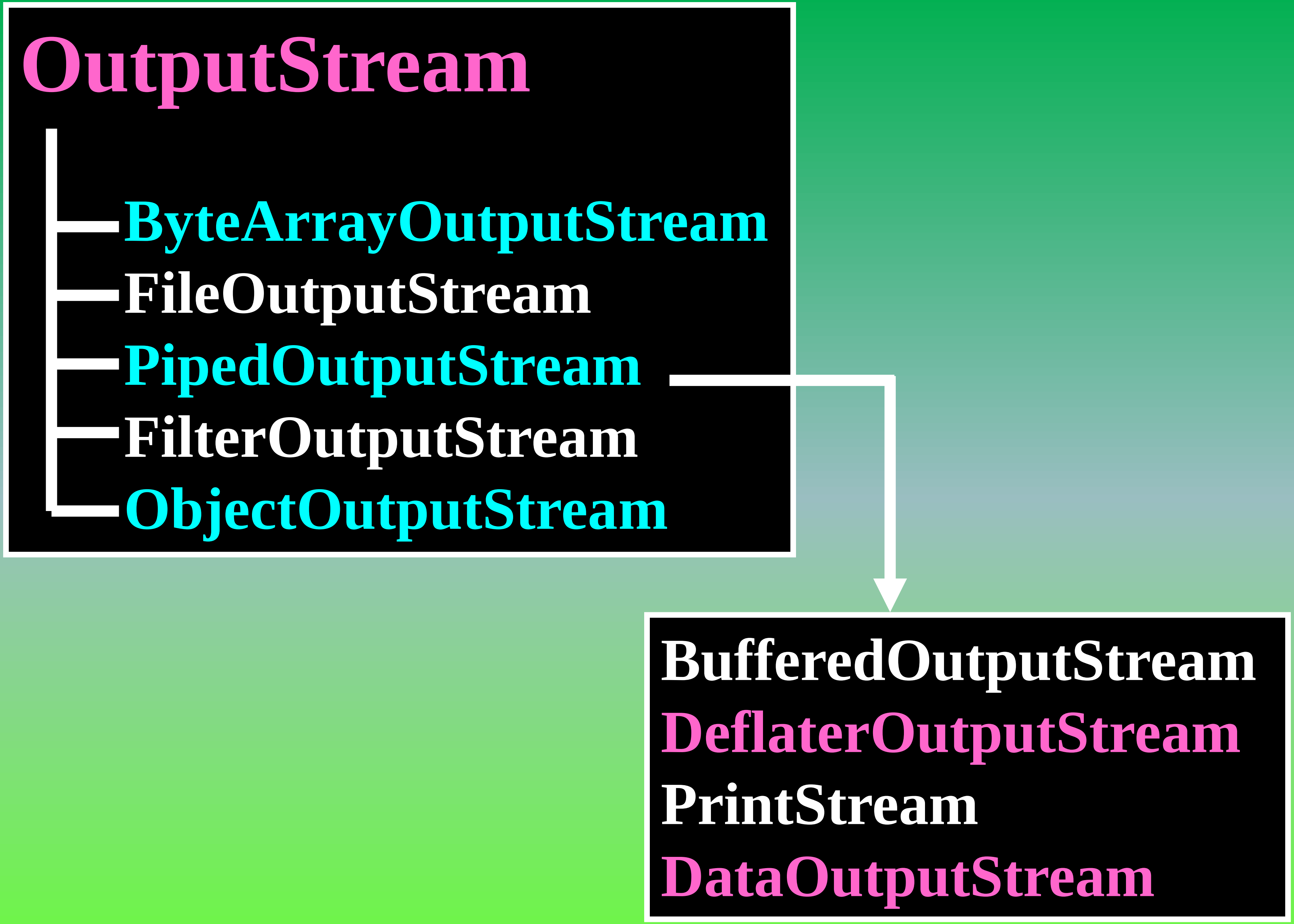
- To create a file, it must be opened first. This is done by creating a ***file stream***, and then link to the file name.
- A ***file stream*** can be defined by using the classes of **Reader/InputStream** for **reading data from** a file.
- Similarly the classes of **Writer/ OutputStream** for **writing data to** file.

Byte Streams

- It is to handle any characters like ***text***, ***image***, ***audio***, and ***video files***.

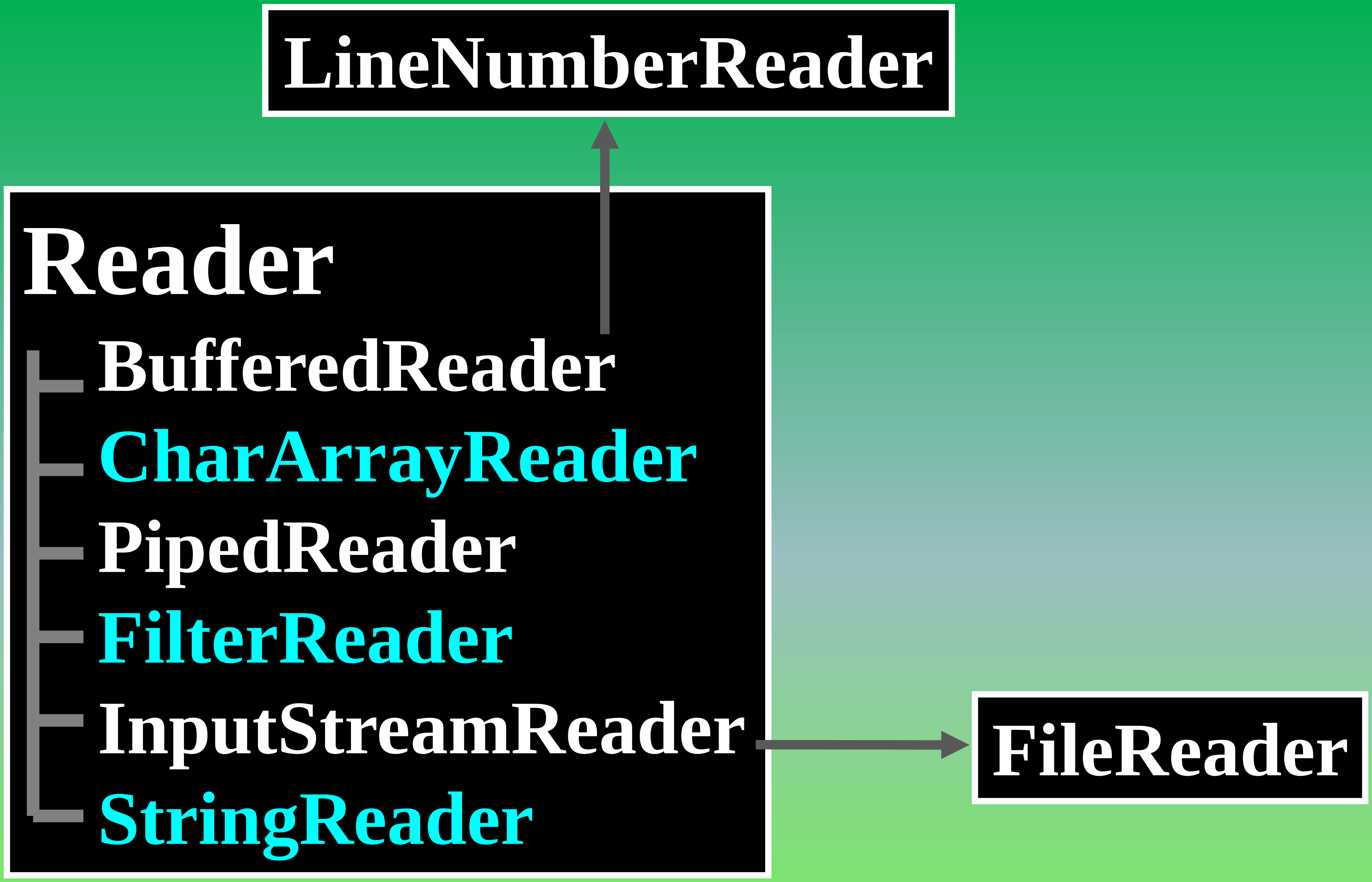
- The important classes of byte stream are **InputStream** and **OutputStream** for reading and writing purpose.

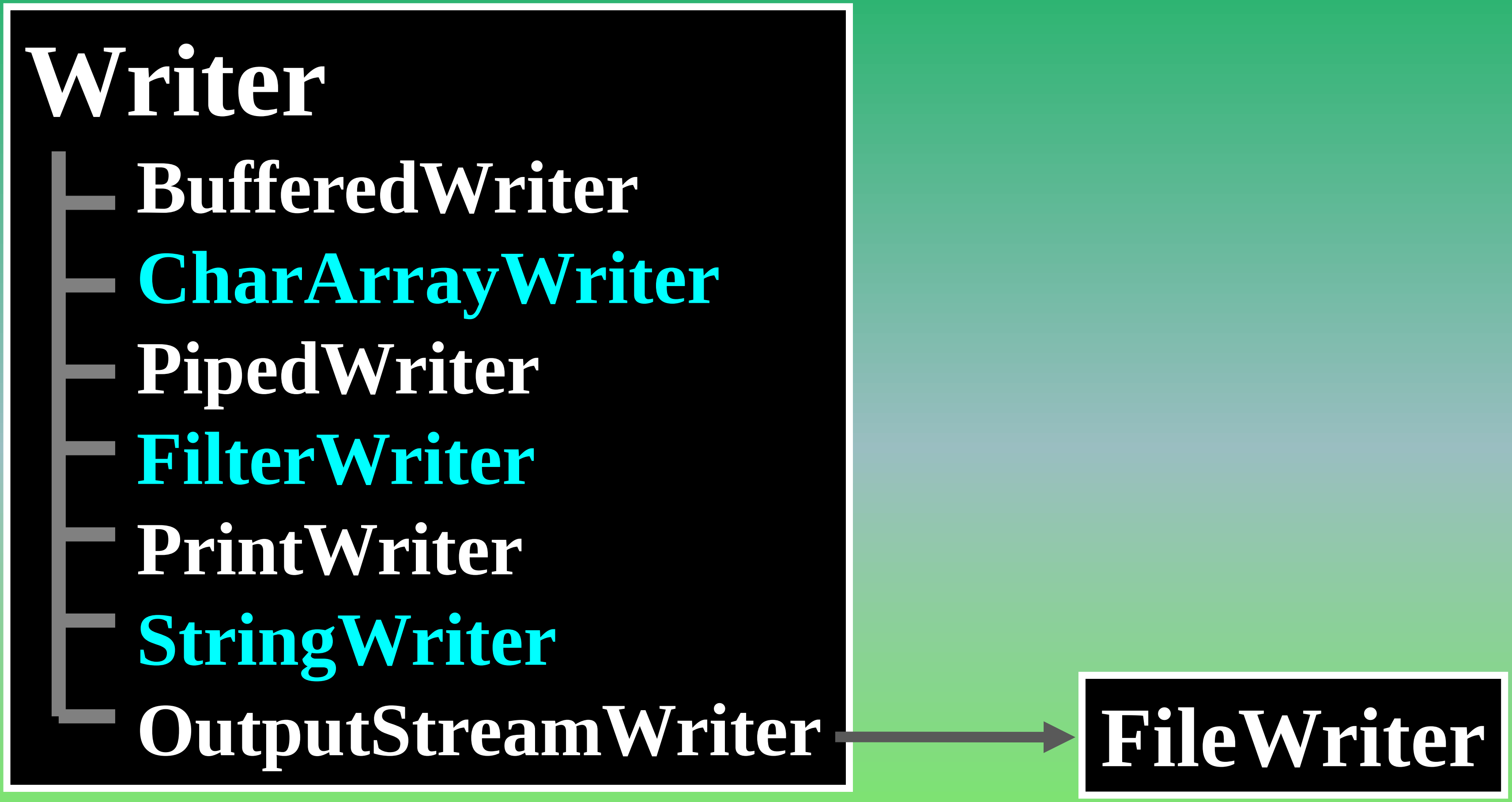

InputStream
ByteArrayInputStream
FileInputStream
PipedInputStream
FilterInputStream
ObjectInputStream
BufferedInputStream
InflaterInputStream
LineNumberedInputStream
DataInputStream


OutputStream
ByteArrayOutputStream
FileOutputStream
PipedOutputStream
FilterOutputStream
ObjectOutputStream
BufferedOutputStream
DeflaterOutputStream
PrintStream
DataOutputStream

Character or Text Streams

- It is to store and retrieve data in the form of **characters (or text**) only.
- It cannot handle image, video, and audio files.
- The important classes of character stream are **Reader** and **Writer** for reading and writing purpose.


LineNumberReader
Reader
BufferedReader
CharArrayReader
PipedReader
FilterReader
InputStreamReader
StringReader
FileReader


Writer
BufferedWriter
CharArrayWriter
PipedWriter
FilterWriter
PrintWriter
StringWriter
OutputStreamWriter
FileWriter

Two ways a file can be opened for writing or reading purpose.

i. Direct Approach

```
FileOutputStream  fileObjectName;
try
{
  fileObjectName = new FileOutputStream(“FileName”);
}
catch (IOException objectName)
{
      // statement(s);
}
```

(or)

```
class class_name
{
public static void main() throws IOException
{
 FileOutputStream  fileObjecjName =
                         new FileOutputStream("FileName");
 --------
 --------
}
```

ii. Indirect Approach

```
File fileObject = new File(“fileName”);

FileOutputStream  objectName;

try
{
 objectName = new FileOutputStream(fileObject);
}
catch (IOException exceptionObject)
{
     // statement(s);
}
```

(or)

```
class class_name
{
public static void main(String args[ ]) throws IOException
{
File fileObject = new File(“fileName”);

FileOutputStream  objectName =
                              new FileOutputStream(fileObject);
--------------
}
}
```

Writing and Reading Character data in/from a file

i. write( ): To write a character in a file.

Syntax: **stream_object.write(char value);**

```
Examples:   (i) obj.write(‘m’);
            (ii) char d = ‘z’;
                 obj.write(d);
```

ii. read( ): To read a character from a file.

Syntax: **int stream_object.read( )**

```
Example:    int a;
            a = obj.read();
```

Checking End of File

- **read( )** method is used to check for end of file.
- If it returns **-1** then the stream object is reached at the end of file.
- If it is not reached at the end of file then the return value is either **0** or **+ve value**.

Syntax: (i) **while(stream_object.read()!=-1)**

```
{
  // statement(s);
}
```

(ii) while(1)

```
{
  // statement(s);
  if( stream_object.read()==-1)
  {
   System.out.print("End of file");
   System.exit(1);
  }
```

close( ): To close a file.

Syntax: **stream_object.close( );**

Writing Primitive data in a file

i. writeInt( ): To write an integer data.

Syntax: stream_object.writeInt(int value);

Examples: (a) obj.writeInt(10);

(b) int n=5; obj.writeInt(n);

(c) obj.writeInt(n+3);

ii. writeDouble( ): To write a double value.

Syntax: stream_object.writeDouble(double value);

Writing Primitive data in a file

iii. writeChar( ): To write a character data.

Syntax: stream_object.writeChar(char value);

iv. writeBoolean( ): To write a boolean value.

Syntax: **stream_object.writeBoolean(boolean value);**

v. writeBytes( ): To write a string data.

Syntax: **stream_object.writeBytes(String value);**

Reading primitive data from a file

i. readInt( ): To read an integer data.

Syntax: **int stream_object.readInt( );**

Example: int a = obj.readInt( );

ii. readDouble( ): To read a double value.

Syntax: **double stream_object.readDouble( );**

iii. readChar( ): To read a character data.

Syntax: **char stream_object.readChar( );**

iv. readBoolean( ): To read a boolean value.

Syntax: **boolean stream_object.readBoolean( );**

v. readLine( ): To read a string data.

Syntax: **String stream_object.readLine( );**

Input and Output Exception

i. **EOFException:** An end of file or end of stream has been reached unexpectedly during input.

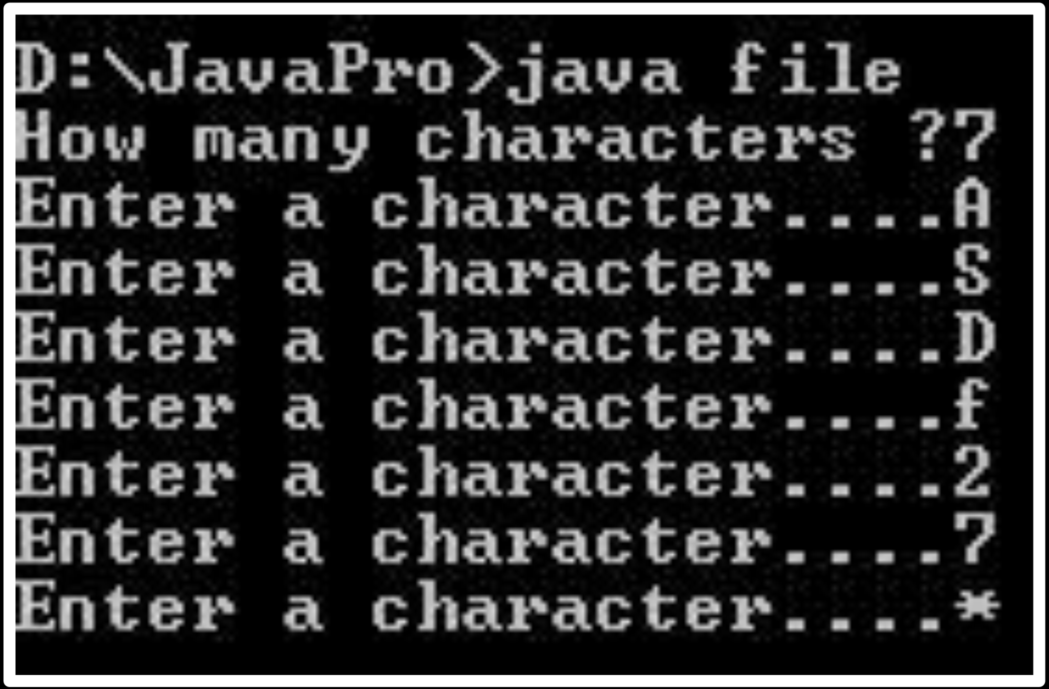
ii. **FileNotFoundException:** Informs about unavailability of file.

iii. **IOException:** An I/O exception has been occurred.

iv. **InterruptedIOException:** The I/O operation as been interrupted.

```
// Read n characters and store in a file called "char.txt"
// using FileWriter method(Character type).

import java.io.*;
 class file
 {
 public static void main(String args[ ]) throws IOException
 {
  DataInputStream in = new DataInputStream(System.in);
  FileWriter fobj = new  FileWriter("char.txt");

    System.out.print("How many characters ?");
    int n = Integer.parseInt(in.readLine());
    for(int i=0;i<n;i++)
    {
      System.out.print("Enter a character....");
      char ch = (char)System.in.read();
      in.skip(2);  // increment 2 bytes to position the object “in”
      fobj.write(ch);  // for picking the next character
     }
      fobj.close();
  }
 }
```

Output



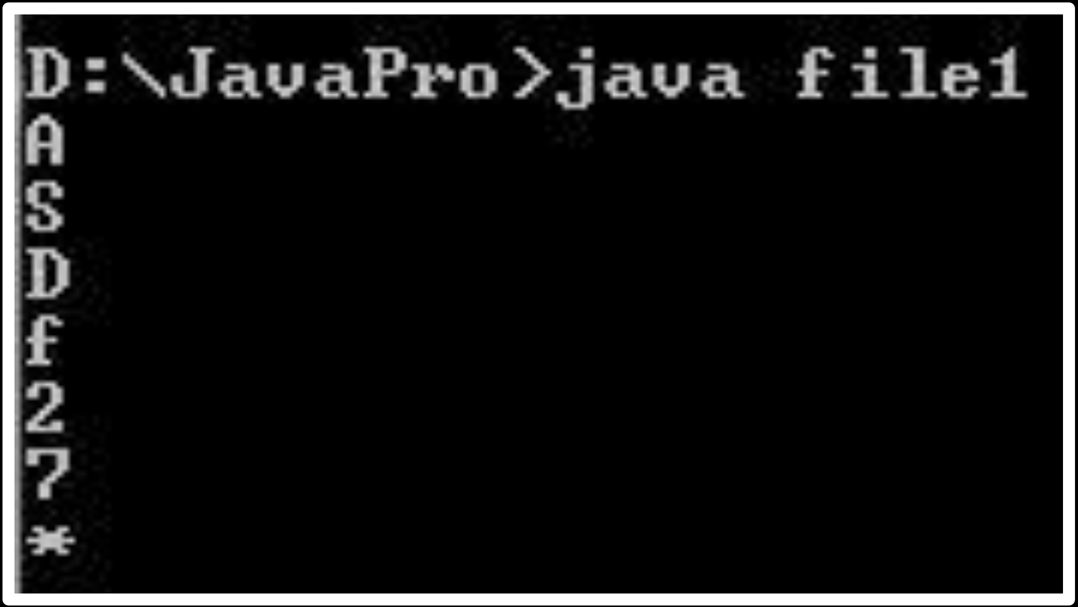
```
D:\JavaPro>java file
How many characters ?7
Enter a character....A
Enter a character....S
Enter a character....D
Enter a character....f
Enter a character....2
Enter a character....7
Enter a character....*
```

char.txt
ASDf27*

// Read the contents of the file called "char.txt" using FileReader
// method(Character type).

```
 import java.io.*;
 class file1
 {
 public static void main(String args[ ]) throws IOException
 {
   DataInputStream in = new DataInputStream(System.in);
   FileReader fobj = new  FileReader("char.txt");
   int ch;
   while((ch=fobj.read())!=-1)       or     ch=fobj.read();
   {                                         while(1)
     System.out.println((char)ch);          {
    }                                       if(fobj==-1) System.exit(1);
     fobj.close();                          System.out.println((char)ch);
 }                                          ch=fobj.read();
                                            }
```

Output

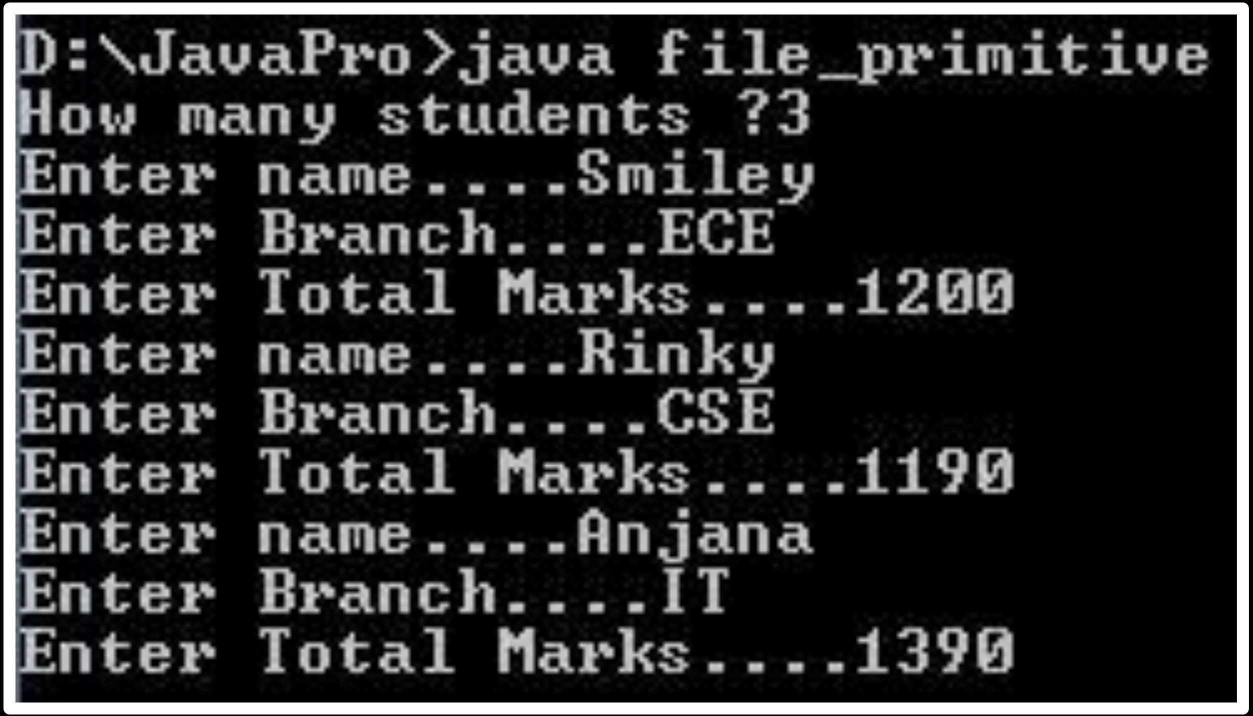


```
D:\JavaPro>java file1
A
S
D
f
2
7
*
```

```
// Program to store primitive data in a file.
import java.io.*;
class file_primitive
{
public static void main(String args[ ]) throws IOException
{
DataInputStream in = new DataInputStream(System.in);
FileOutputStream fobj = new FileOutputStream("student.txt");
DataOutputStream obj = new DataOutputStream(fobj);

System.out.print("How many students ?");
int n = Integer.parseInt(in.readLine());

for(int i=1;i<=n;i++)
{
   System.out.print("Enter name....");
   String name = in.readLine( );
```

```
    System.out.print("Enter Branch....");
    String b = in.readLine( );
    System.out.print("Enter Total Marks....");
    int mark = Integer.parseInt(in.readLine( ));

    obj.writeBytes(name);
    obj.writeBytes("\n");
    obj.writeBytes(b);
    obj.writeChar('\n');
    obj.writeInt(mark);
    obj.writeChar('\n');
}
    obj.close();
    fobj.close();
}
}
```

Output



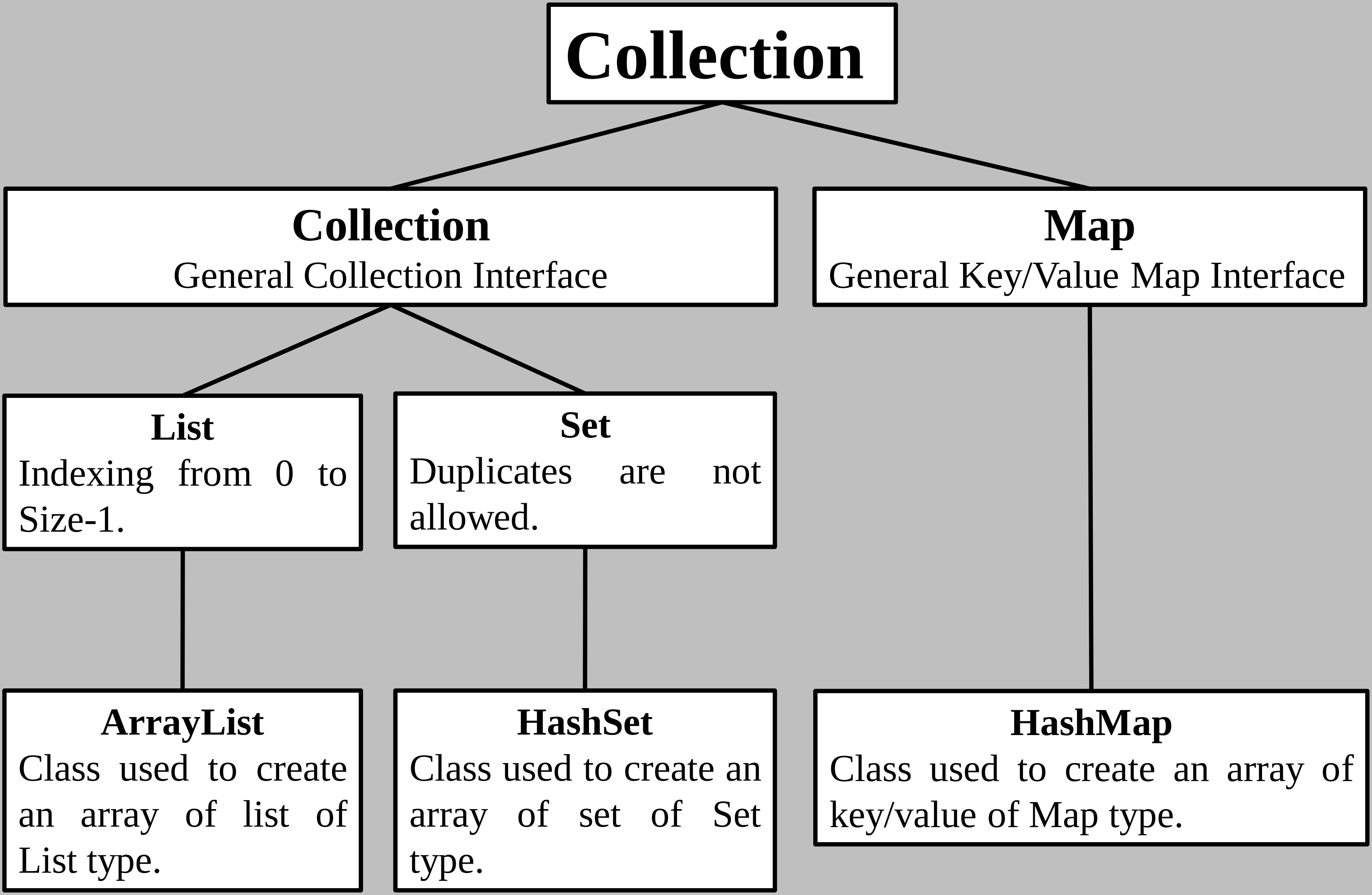
```
D:\JavaPro>java file_primitive
How many students ?3
Enter name....Smiley
Enter Branch....ECE
Enter Total Marks....1200
Enter name....Rinky
Enter Branch....CSE
Enter Total Marks....1190
Enter name....Anjana
Enter Branch....IT
Enter Total Marks....1390
```

// Program to read primitive data from a file.

```
import java.io.*;
import java.lang.*;
public class file_primitive1
{
  public static void main(String args[ ]) throws IOException
  {
  FileInputStream fobj = new FileInputStream("student.txt");
  DataInputStream obj = new DataInputStream(fobj);
  System.out.println("NAME\tBRANCH\tTOTAL");
  String st;
while((st=obj.readLine())!=null)
{
   String a = obj.readLine();
   int c = obj.readInt();
   System.out.println(st+"\t"+a+"\t"+c);
}
    obj.close();
    fobj.close();
 }
}
```

Java Collections

- List
- Set
- Map



Collection

Collection
General Collection Interface

Map
General Key/Value Map Interface

List
Indexing from 0 to Size-1.

Set
Duplicates are not allowed.

ArrayList
Class used to create an array of list of List type.

HashSet
Class used to create an array of set of Set type.

HashMap
Class used to create an array of key/value of Map type.

Lists

- The List is probably the single most useful and widely used type of Collection.

- List is a general interface.

- **ArrayList** and **LinkedList** are implementing classes.

- **ArrayList** class is the best general purpose List.

- A List is a linear structure where each element is accessed by an index number 0, 1, 2, ... len-1 (like an array).

- Lists only **store objects**, like **String**, **Integer**, **Float** etc., but not primitives like int, char, float etc.

Syntax:

List<Type> ObjectReference = new ArrayList<Type>( );

List Methods

i) **add(Object obj)**

ii) **add(int index, Object obj)**

iii) **get(int index)**

iv) **set(int index, Object obj)**

v) **remove(int index)**

vi) **size( )**

i. **add(Object obj):** Adds a new element at the end of the list.

ii. **add(int index, Object obj):** Adds a new element into the list at the given index, shifting any existing elements at greater index positions over to make a space.

iii. **get(int index):** Returns the element at the given index.

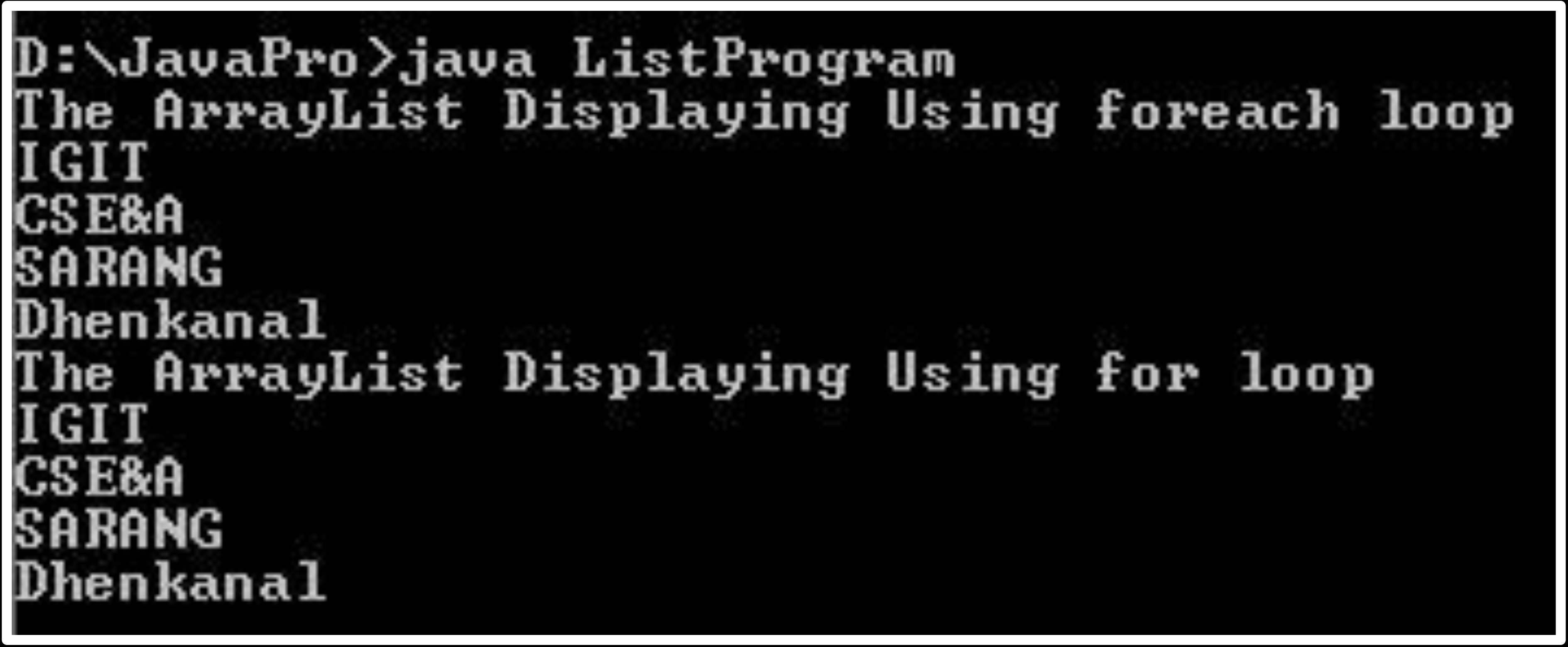
iv. set(int index, Object obj): Sets the element at the given index in a list.

v. remove(int index): Removes the element at the given index, shifting any elements at greater index positions down to take up the space.

vi. size( ): Returns the size of Lists or any collection.

```
// List example
import java.util.*;
class ListProgram
{
public static void main(String args[])
{
List<String> words = new ArrayList<String>();

words.add("IGIT");
words.add("CSE&A");
words.add("SARANG");
words.add("Dhenkanal");

int len = words.size();
System.out.println("The ArrayList Displaying Using for…each loop");

for (String str : words)
{
    System.out.println(str);
}

System.out.println("The ArrayList Displaying Using for loop");

for (int i=0; i<len; i++)
{
    String str = words.get(i);
    System.out.println(str);
}
}
}
```

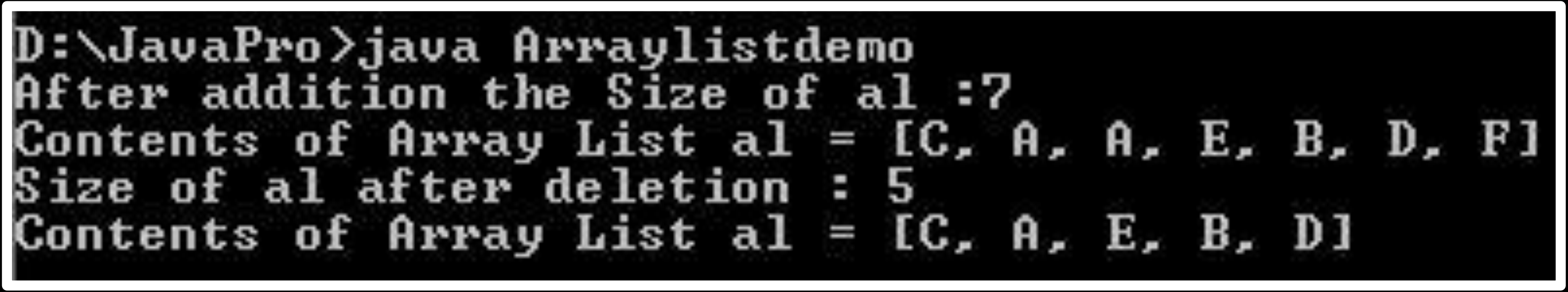
Output



```
D:\JavaPro>java ListProgram
The ArrayList Displaying Using foreach loop
IGIT
CSE&A
SARANG
Dhenkanal
The ArrayList Displaying Using for loop
IGIT
CSE&A
SARANG
Dhenkanal
```

```
import java.util.*;
        class Arraylistdemo
        {
        public static void main(String args[])
        {
        ArrayList al=new ArrayList( );
        al.add("C");
        al.add("A");
        al.add("E");
        al.add("B");
        al.add("D");
        al.add("F");
        al.add(1,"A");
System.out.println("After addition the Size of al :" + al.size( ));

 // To display arraylist
    System.out.println("Contents of Array List al = " + al);

 // To remove elements from arraylist
    al.remove("F");
    al.remove(2);
    System.out.println("Size of al after deletion : " + al.size( ));
    System.out.println("Contents of Array List al = " + al);
   }
}
```

Output

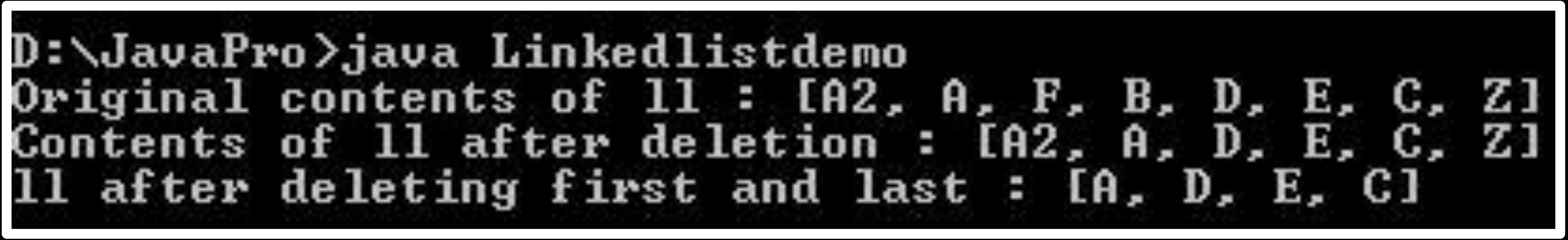


```
D:\JavaPro>java Arraylistdemo
After addition the Size of al :7
Contents of Array List al = [C, A, A, E, B, D, F]
Size of al after deletion : 5
Contents of Array List al = [C, A, E, B, D]
```

// Linked list example

```
import java.util.*;
class Linkedlistdemo
{
   public static void main(String args[ ])
  {    // To create linked list
       LinkedList Ll=new LinkedList( );
       Ll.add("F");
       Ll.add("B");
       Ll.add("D");
       Ll.add("E");
       Ll.add("C");
       Ll.addLast("Z");
       Ll.addFirst("A");
       Ll.add(0,"A2");
System.out.println("Original contents of Ll : "+Ll);

// To remove elements from linked list
  Ll.remove("F");
  Ll.remove(2);
  System.out.println("Contents of Ll after deletion : "+Ll);

  Ll.removeFirst();
  Ll.removeLast();
  System.out.println(“Ll after deleting first and last : "+Ll);
 }
}
```

Output



```
D:\JavaPro>java Linkedlistdemo
Original contents of ll : [A2, A, F, B, D, E, C, Z]
Contents of ll after deletion : [A2, A, D, E, C, Z]
ll after deleting first and last : [A, D, E, C]
```

Sets

Set is a Collection, like the List, but with the added constraint that each element can be in the Set once.

Syntax:

Set<type> ObjectRef = new HashSet<type>( );

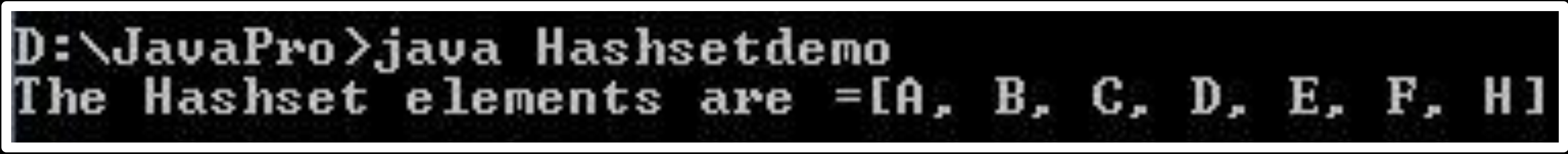
(or) HashSet<type> ObjectRef = new HashSet<type>( );

The various methods are:

- **add( )**
- **contains( )**
- **addAll( )**
- **retainAll( )**
- **containsAll( )**
- **iterator( )**

```
// Set example
import java.util.*;
class Hashsetdemo
{
        public static void main(String args[])
        {
        Set hs =new HashSet( );  // (or)  HashSet hs =new HashSet( );
        hs.add("B");
        hs.add("A");
        hs.add("D");
        hs.add("E");
        hs.add("C");
        hs.add("F");
        hs.add("H");
        System.out.println("The Hashset elements are =" + hs);
        }
}
```

Output



```
D:\JavaPro>java Hashsetdemo
The Hashset elements are =[A, B, C, D, E, F, H]
```

APPLET

- Applets are small java programs that are primarily used in internet computing.
- It can be transported over the internet from one computer to another.
- It can run using **applet viewer** or **any web browser** that supports java.
- An applet can perform arithmetic operation and play interactive games.

- There are two types of applets, local and remote applet.

- **Local Applet:** An applet developed locally and stored in a local system is known as local applets. These applets does not require internet.

- **Remote Applet:** A remote applet is developed by someone else and stored on a remote computer connected to the internet. To locate and load, we must know its address i.e. **Uniform Resource Locator (URL)** which must be specified in the applet's HTML document as:

CODEBASE = http://web-site/applet-name

How Applet differ from Applications

- Applets do not use **main ( )**. It automatically calls certain methods of Applet class to start and execute the applet code.
- It cannot run independently. It needs web page using **HTML tag**.
- It cannot read data from as well as write to the files.
- It cannot communicate with other servers on the network.
- It cannot run any program from the local computer.
- It cannot include library files from other languages such as C or C++.

Steps to write Applets

i. Build an applet code (**.java file**)

ii. Create an executing applet(**.class**)

iii. Design a web page using HTML tags.

iv. Prepare **<APPLET>** tag and include into the web page.

v. Create HTML file (**.HTML file**)

vi. Test the applet code.

Building Applet Code

- Applet code must include two classes namely **Applet** and **Graphics**.
- Applet class present in **java.applet** package and Graphics class in **java.awt** package. Every applet has a life cycle.
- **java.applet** package provide life and behavior to the applets through its methods **init( ), start( ),** and **paint( )**.
- When the applet begins, the **AWT** (Abstract Window Toolkit) calls the above 3 methods in that sequence.
- When an applet is terminated, the following sequence of method calls takes place: **stop( ), destroy( )**.

(i) init( ): It is the first method to be called. The variable initialization must be done. It calls only once during the run time of your applet.

(ii) start( ): It executes after **init( )**. It is also called to restart an applet. To display HTML document again and again, **start( )** method is called that many times.

(iii) paint( ): To show the applets output.

```
public void paint (Graphics object)
{
        // statement(s);
 }
```

(iv) stop( ):

- It is called when the web browser leaves the HTML document containing the applet by moving to another page.

- It is used to suspend threads that don’t need to run when the applet is not visible. Again to restart an applet by calling **start( )**.

(v) destroy( ): To remove the applet completely from the memory.

(vi) update( ): To redraw a window need to call update( ). By default **update( )** fills an applet with the default background color and calls **paint ( )** for displaying of applet output.

```
public void update (Graphics  object)
{
  object.update( );    // it calls paint( ) method
}
```

(vii) drawString( ): It outputs a string at a specified row and column position.

```
object.drawString(String,  column_value,  row_value);
```

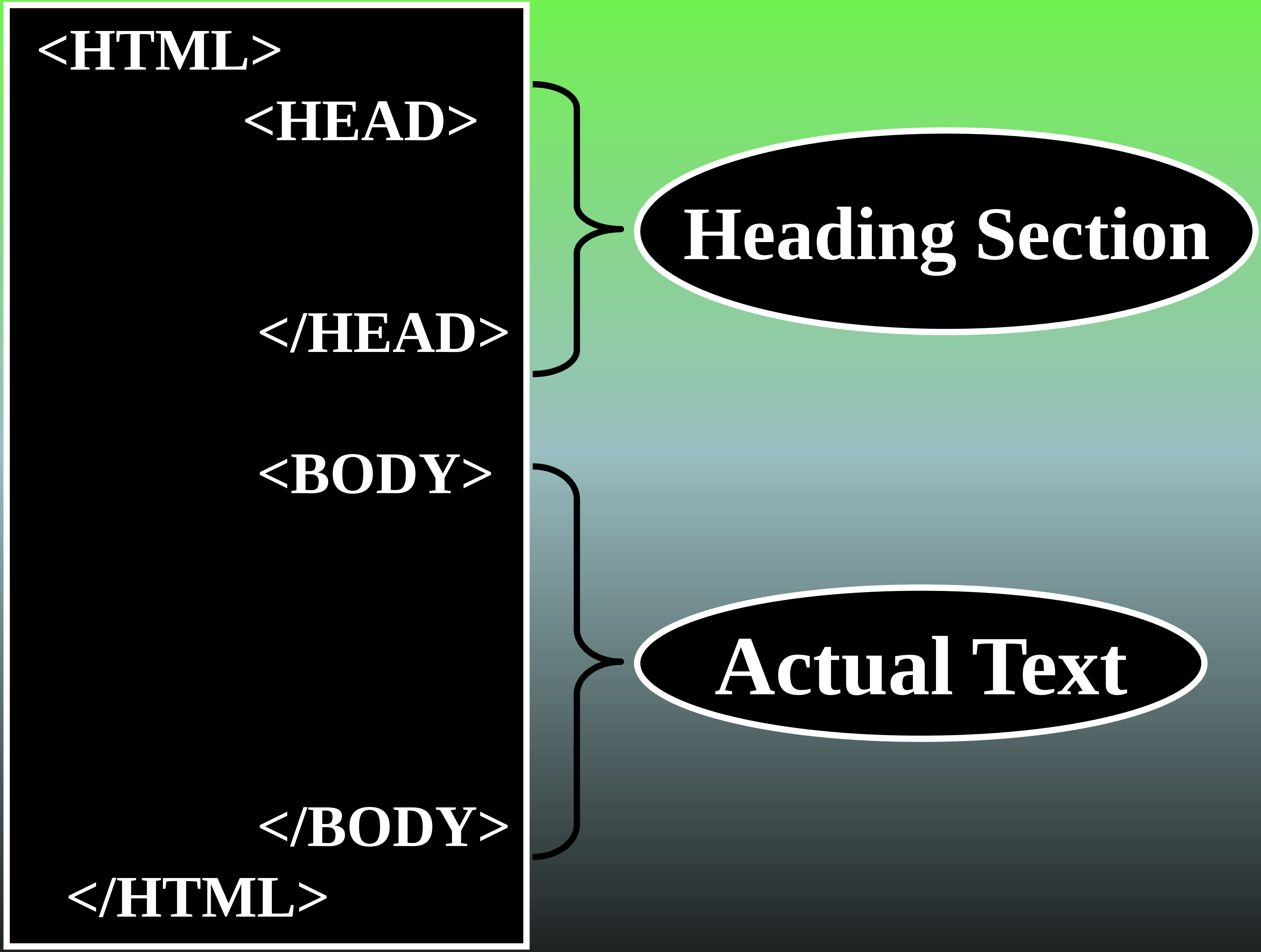
```
// Applet example
import java.awt.*;
import java.applet.*;

public class my_applet extends Applet
{
   public void paint(Graphics g)
    {
      g.drawString("HELLO APPLET", 50, 50);
    }
}
```

- Save the above applet as **my_applet.java**

- Compile to get the byte code as **my_applet.class**

- Finally, the byte code becomes the **applet** to be attached in a HTML document.

Structure of HTML Document



```
<HTML>
        <HEAD>


         </HEAD>

         <BODY>





         </BODY>
  </HTML>
```

<APPLET> </APPLET> : To add the applet into the html tag for displaying the applet.

Syntax:

<Applet CODE="Applet Name" **Width** =Value **Height**=Value> **</Applet>**

```
 <! MY FIRST APPLET PROGRAM >
<HTML>
    <HEAD>
            <H1> WEL COME TO MY APPLET </H1>
    </HEAD>
    <BODY>
 <APPLET CODE=my_applet.class WIDTH=500 HEIGHT=500>
 </APPLET>
    </BODY>
 </HTML>
```

Steps to Run the above Applet

Save as **my_applet.html**

Run it through a **browser**

(or)

At the command prompt:

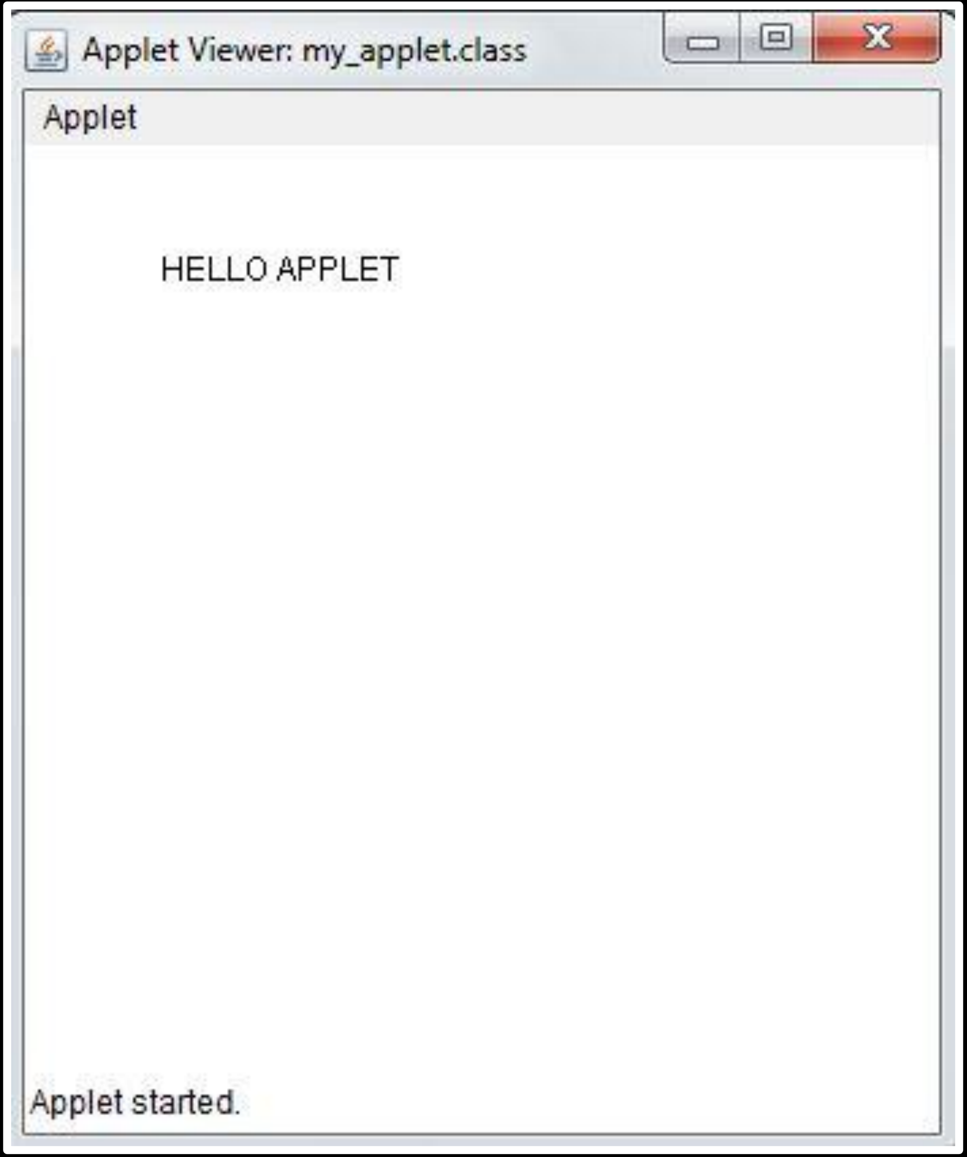
```
WD:\> appletViewer  my_applet ↵
```

Note: WD means Java current working directory

```
D:\JavaPro\Applets>appletviewer my_applet.html
```

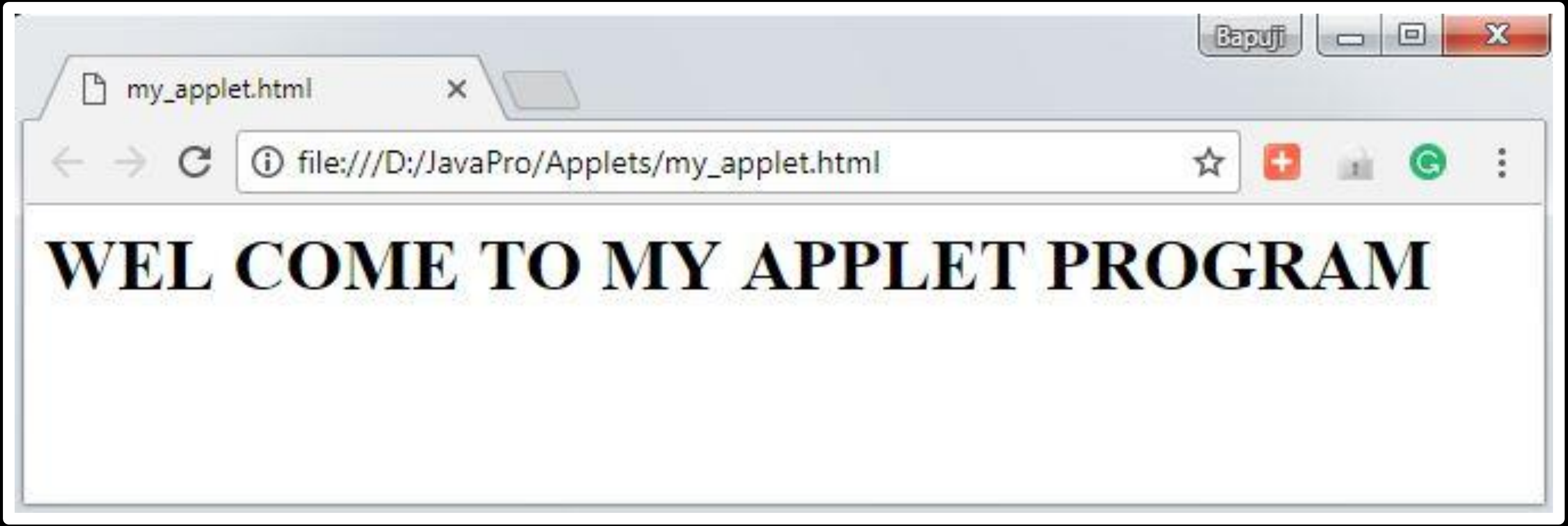
Current Java working directory (WD)

Output



```
Applet Viewer: my_applet.class
Applet

HELLO APPLET

Applet started.
```

Output



WEL COME TO MY APPLET PROGRAM

Methods of Applet Background

(1) setBackground( ): To set the background color of the applet.

Syntax: void setBackground (Color.color_name);

(2) getBackground( ): To get the background color of the applet.

Syntax: Color getBackground( )

The class **Color** defines the following 13 color constants:

red , green , black , white , blue , cyan , pink , yellow , gray , lightGray , darkGray, orange, magenta

Methods of Applet Foreground

(1) setForeground( ): To set the foreground color of the applet i.e. the text color and the line, rectangle's etc. drawing color.

Syntax: void setForeground(Color.color_name);

(2) getForeground( ): To get the foreground color of the applet.

Syntax: Color getForeground( );

```
//  To display text with different colors (my_applet4.java)
  import java.awt.*;
  import java.applet.*;
public class my_applet4 extends Applet
{   Color p;
   public void paint(Graphics g)
    { p = new Color(10,100,100);
      setForeground(p);
      g.drawString("HELLO APPLET",50,50);
      g.drawString("HELLO APPLET",300,50);
      g.drawString("HELLO APPLET",550,50);
    }
  }
```

my_applet4.html

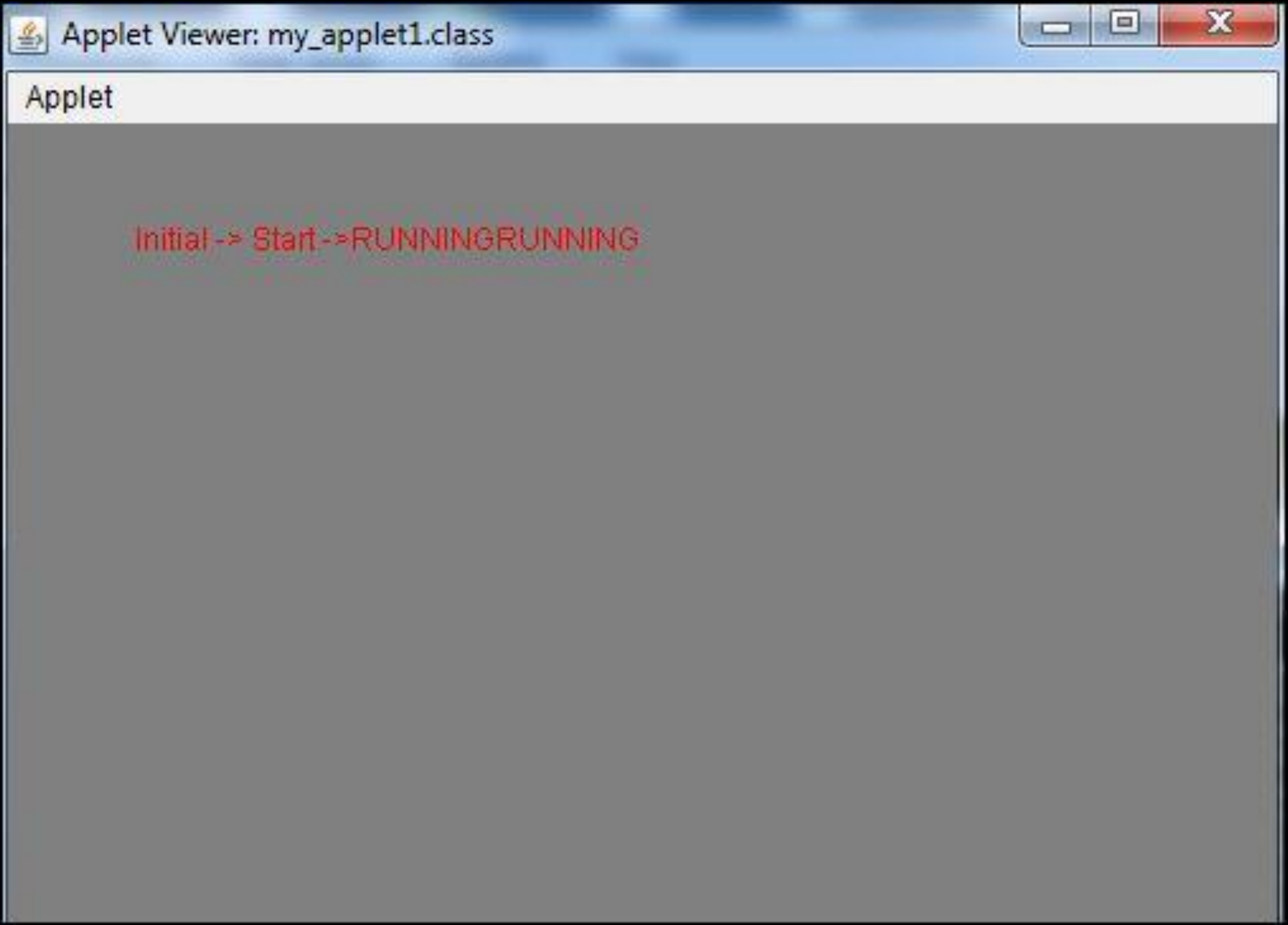
```
<HTML>
        <! Applet Program to display different text colours>
    <HEAD>
     <TITLE>  WEL COME TEXT COLOUR APPLETS </TITLE>
     </HEAD>
    <BODY>
                <CENTER>
                 <H1> APPLET TEXT COLOR PROGRAM </H1>
                </CENTER>
          <BR>
      <CENTER>
 <APPLET CODE="my_applet4.class" WIDTH=800 HEIGHT=400>
        </APPLET>
      </CENTER>
    </BODY>
   </HTML>
```

```
// A sample applet that sets the foreground and background colors
// and outputs a string. (my_applet1.java)
   import java.awt.*;
   import java.applet.*;
   public class my_applet1 extends Applet
   {
        String st;
   public void init( )
   {
          st="Initial ->";
                    setBackground(Color.gray);
          setForeground(Color.red);
    }
public void start( )
{
          st = st + " Start ->";
 }
     public void paint( Graphics a)
     {
          st = st + "RUNNING";
                a.drawString(st,50,50);
     }
}
```

my_applet1.html

```
<HTML>
   <APPLET CODE="my_applet1.class"
                                              WIDTH=500 HEIGHT=800>
   </APPLET>
   </HTML>
```

Output



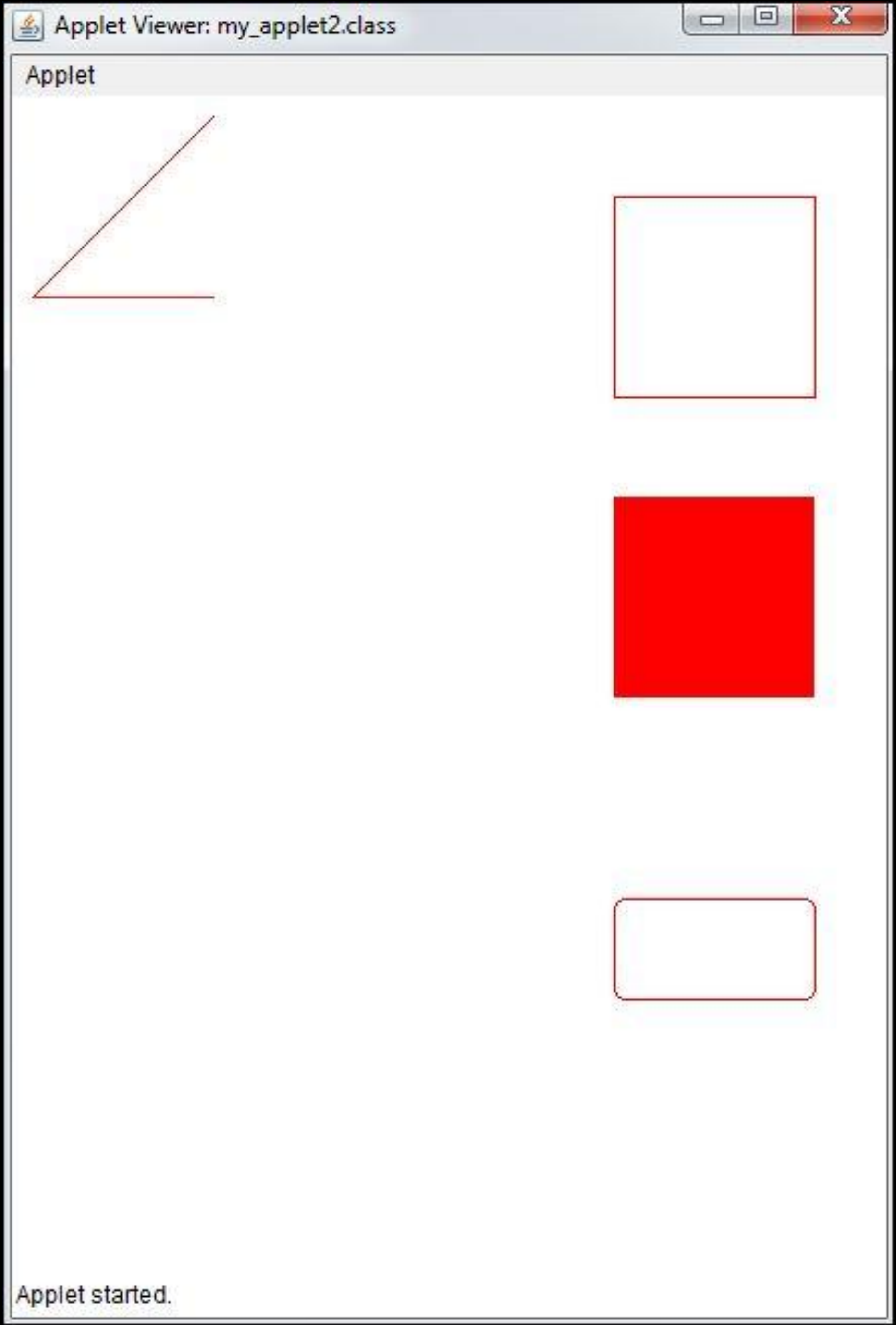
```
Initial -> Start->RUNNINGRUNNING
```

```
//  Applet to draw lines, boxes, rounded boxes, ovals, and arcs
//                        (my_applet2.java)
   import java.awt.*;
   import java.applet.*;
   public class my_applet2 extends Applet
   {
  public void paint( Graphics a)
  {
      setForeground(Color.red);
      a.drawLine(10,100,100,100);
      a.drawLine(10,100,10,100);
      a.drawLine(10,100,100,10);
      a.drawRect(300,50,100,100);
      a.fillRect(300,200,100,100);
      a.drawRoundRect(300,400,100,50,10,10);
       }
   }
```

my_applet2.html

```
<HTML>
     <HEAD>
    <TITLE>  WEL COME TO  APPLET  WHICH DRAWS
                                          LINE, BOX, OVAL, ARC     </TITLE>
    </HEAD>
   <BODY>
 <APPLET CODE="my_applet2.class" WIDTH=800 HEIGHT=800>
      </APPLET>
     </BODY>
   </HTML>
```

Output



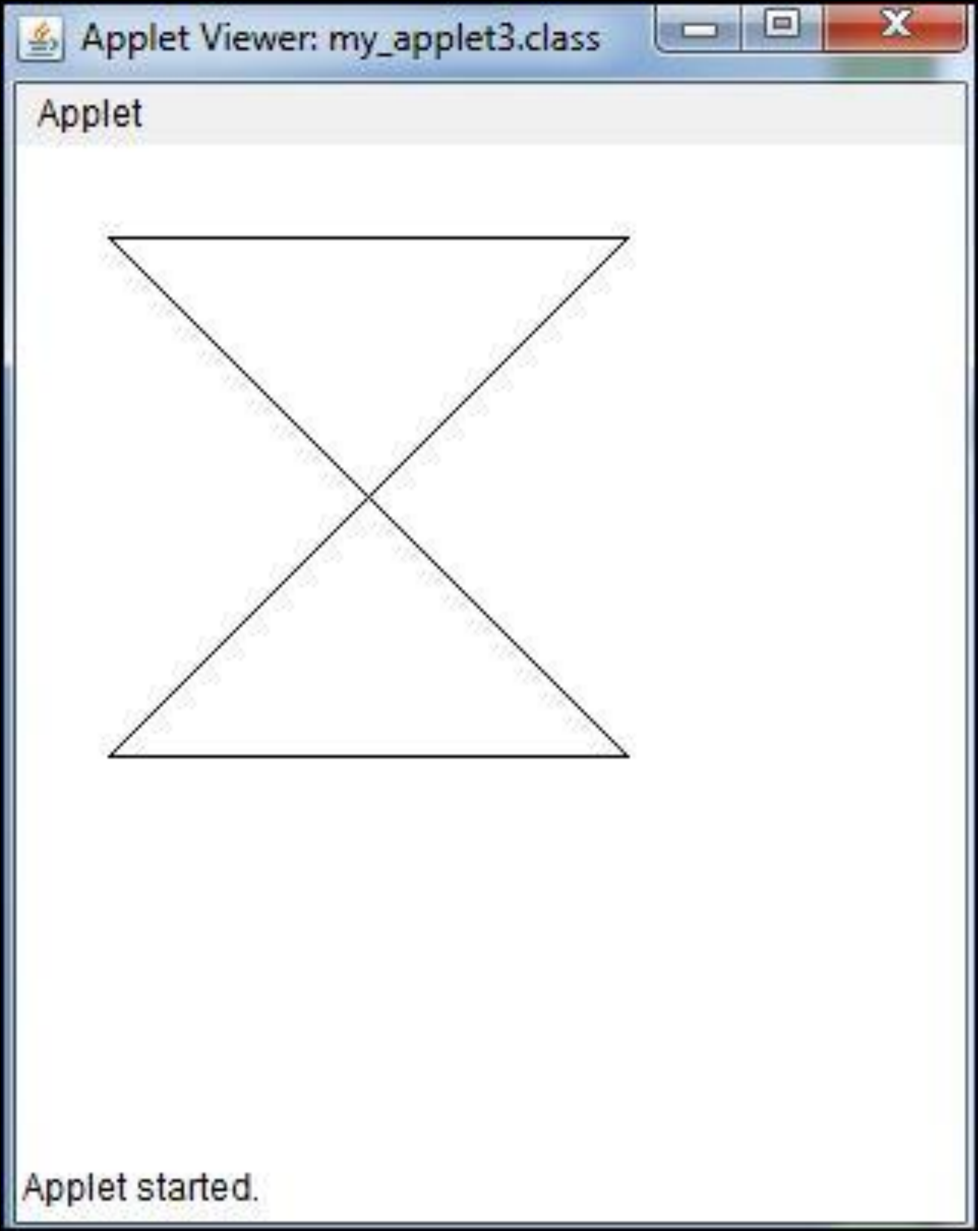
Applet started.

```
//  applet to draw polygons. (my_applet3.java)
   import java.awt.*;
   import java.applet.*;
   public class my_applet3 extends Applet
   {


     public void paint( Graphics a)
        {
            int x[]={30,200,30,200,30};
            int y[]={30,30,200,200,30};
            int ns=5;

            a.drawPolygon(x,y,ns);
        }
   }
```

my_applet3.html

```
<HTML>
     <HEAD>
    <TITLE>  WEL COME TO  APPLET  WHICH DRAWS A
                                                                    POLYGON</TITLE>
    </HEAD>
   <BODY>
<APPLET CODE="my_applet3.class" WIDTH=800 HEIGHT=800>
       </APPLET>
     </BODY>
  </HTML>
```

Output



AWT CONTROLS

The environment where the user can interact with an application through graphics or images is called **GUI (Graphics User Interface)**.

GUI has the following advantages:

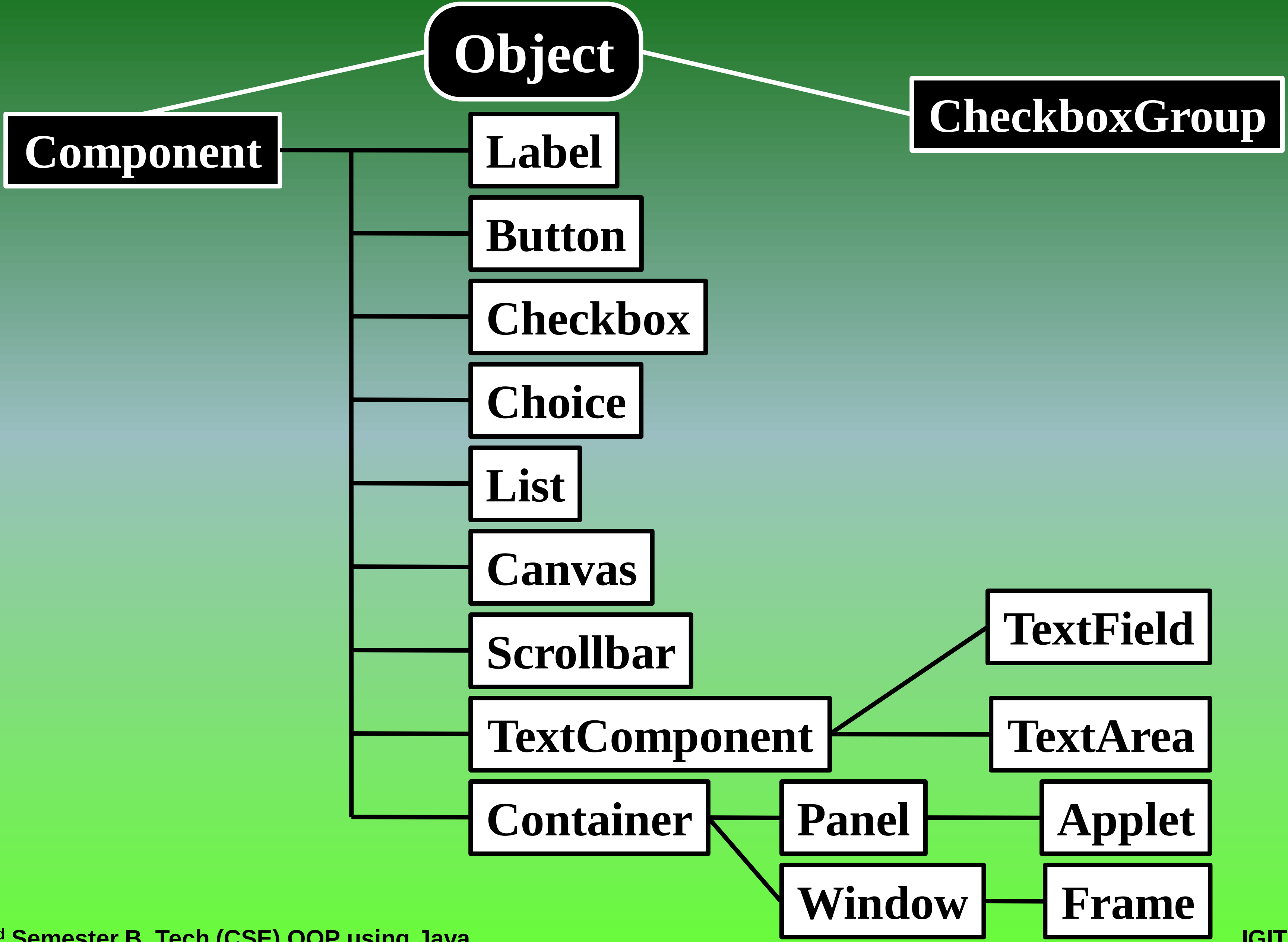
i. It is user-friendly
ii. It adds attraction and beauty to any application by adding pictures, colors, menus, animation, etc.
iii. It is possible to simulate the real life objects using GUI.
iv. It helps to create graphical components like push buttons, radio buttons, check boxes, etc.

AWT

Abstract Window Toolkit (AWT) represents a class library to develop applications using GUI.

The java.awt package has classes and interfaces to develop GUI.

Classes of AWT



Component: A Component represents an object which is displayed pictorially on the screen.

Window: A window represents any imaginary rectangular area on the screen without any border or title bar.

Frame: A frame represents a window with title bar and border.

Creation of Frame:

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

Set the Frame Size:

```
f.setSize(400, 200);
```

The frame’s width is 400pixels and height is 200 pixels.

Visibility of Frame:

```
f.setVisible(true);
```

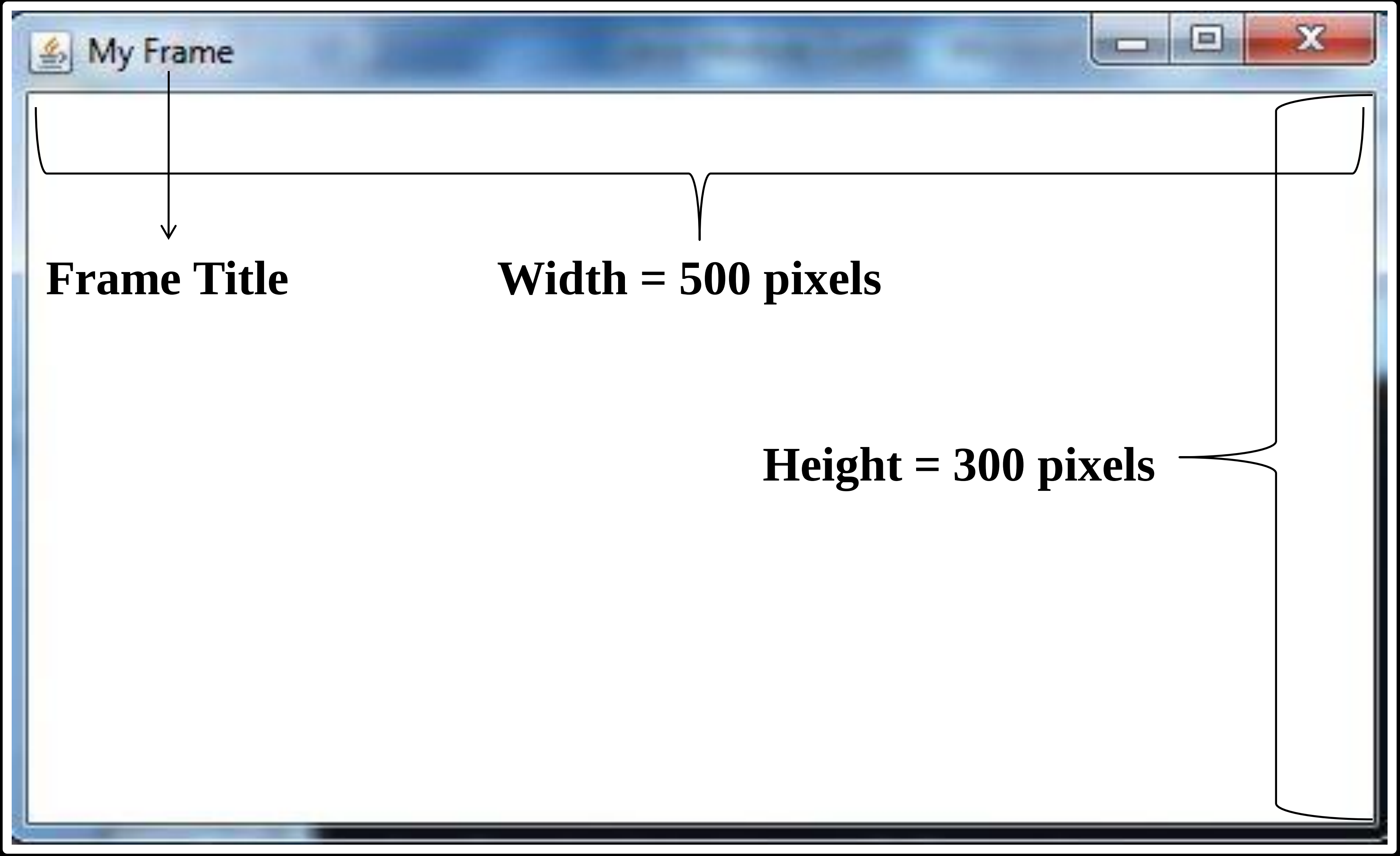
It displays the frame with defined size.

```
// Frame creation and display
import java.awt.*;
class MyFrame
{
public static void main(String args[])
{
    Frame f = new Frame("My Frame");
    f.setSize(500, 300);
    f.setVisible(true);
}
}
```

```
// Frame creation and display (Method-1)
import java.awt.*;
class MyFrame1 extends Frame
{
   MyFrame1(String st)
   {
      super(st);
    }
   public static void main(String args[])
  {
    MyFrame1 f = new MyFrame1("My Frame");
    f.setSize(500, 300);
    f.setVisible(true);
}
}
```

```
// Frame creation and display (Method-2)
import java.awt.*;
class MyFrame2 extends Frame
{  MyFrame2(String st)
   {
      super(st);
    }
}
class FrameCreation
{
   public static void main(String args[])
  {
    MyFrame2 f = new MyFrame2("My Frame");
    f.setSize(500, 300);
    f.setVisible(true);
}
}
```

Output

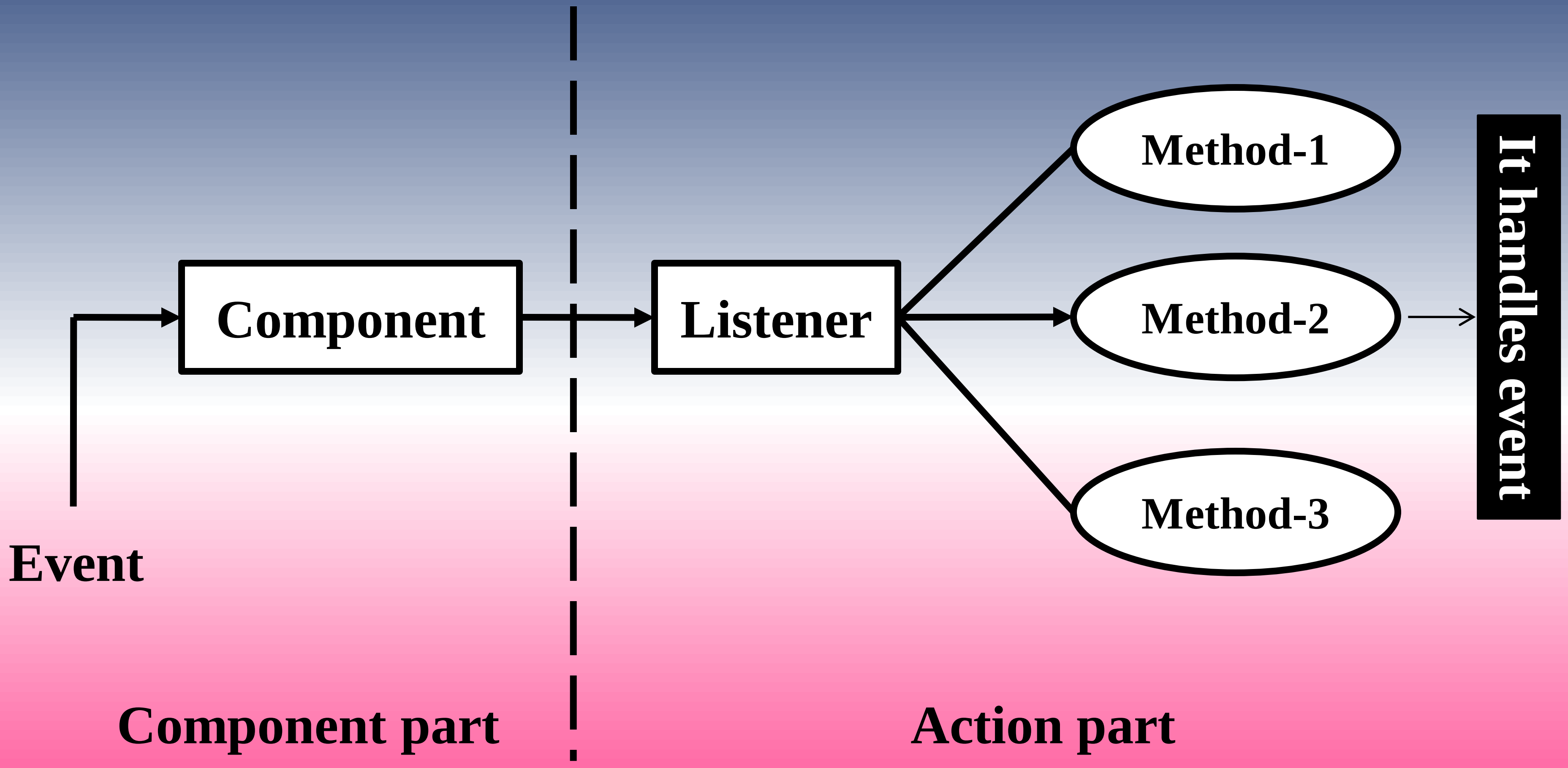


```
// Frame creation, display, and close using windowAdapter class
import java.awt.*;
import java.awt.event.*;
 class  CloseClass extends WindowAdapter
{
  public void windowClosing(WindowEvent e)
 {
     System.exit(0);
  }
}
```

```
class FrameDisplayClose
{
   public static void main(String args[])
  {
    Frame f = new Frame("My Frame, To Close Press Close Button");
    f.setSize(600, 300);
    f.setVisible(true);
    // to close frame
    f.addWindowListener(new CloseClass());
   }
}
```

Event Delegation Model

- An **event** represents a specific action done on a component.
- Clicking, double clicking, typing data inside the component, mouse over, etc. are the examples of events.
- When an event is generated on the component, the component will not know about it because it cannot listen to the event.
- To let the component to understand an event generated on it, there must be some **listener** to be added to the components.
- A listener will have some abstract methods which need to be implemented by the programmer.


Component
Listener
Method-1
Method-2
Method-3
It handles event
Event
Component part
Action part

The following methods present in the **WindowListener** interface:

i. public void **windowActivated**(windowEvent e**)**

ii. public void **windowClosed(**windowEvent e**)**

iii. public void **windowClosing(**windowEvent e**)**

iv. public void **windowDeactivated(**windowEvent e**)**

v. public void **windowDeiconified(**windowEvent e**)**

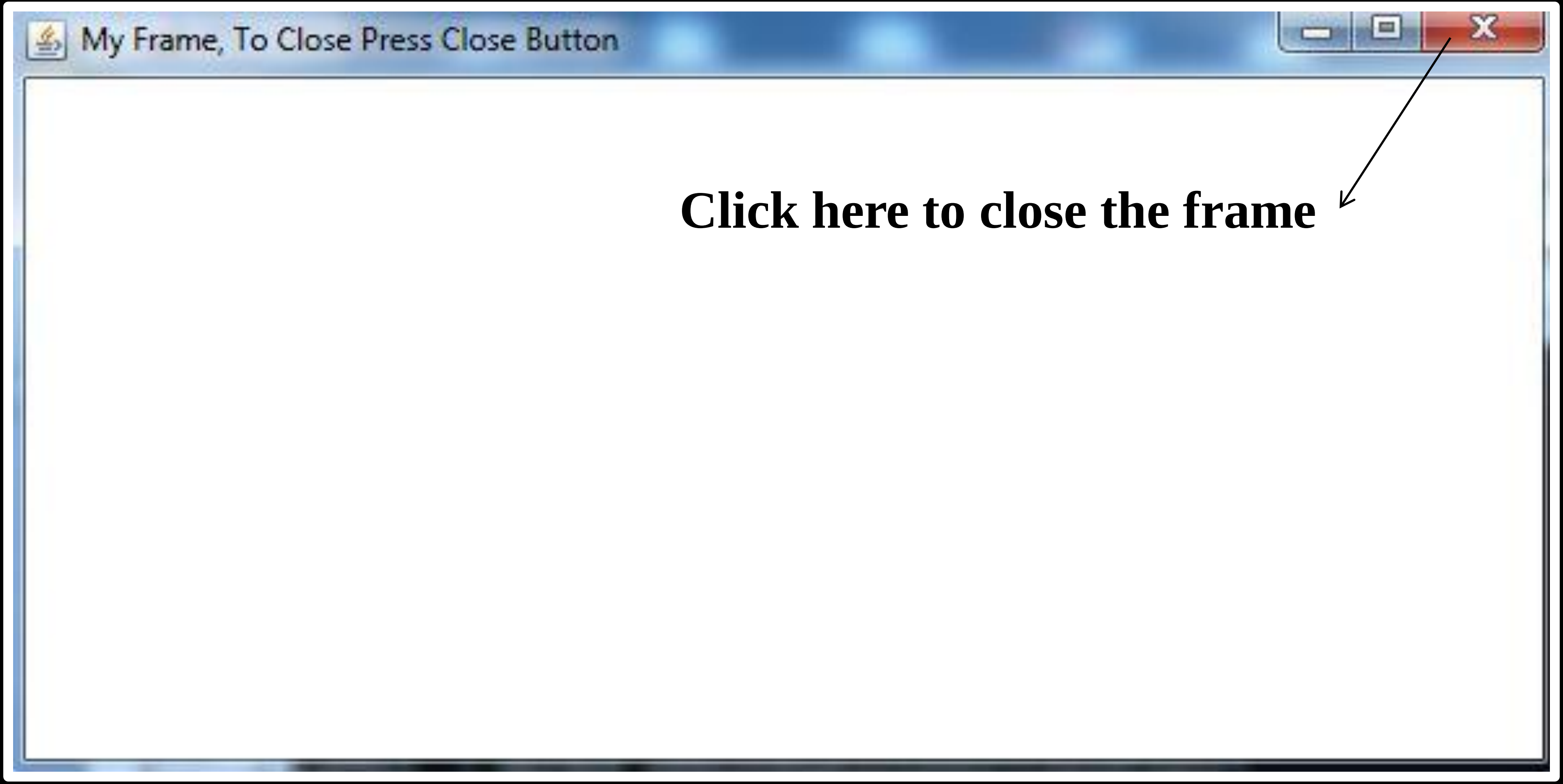
vi. public void **windowIconified**(windowEvent e**)**

vii. public void **windowOpened(**windowEvent e**)**

```
// Frame creation, display, and close using WindowListener class
import java.awt.*;
import java.awt.event.*;
 class CloseClass1 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)  {  }
}
```

```
class FrameDisplayClose1
{
   public static void main(String args[])
  {
    Frame f = new Frame("My Frame, To Close Press Close Button");
    f.setSize(600, 300);
    f.setVisible(true);
    // to close frame
    f.addWindowListener(new CloseClass1());
   }
}
```

Output


My Frame, To Close Press Close Button
Click here to close the frame

The following are the methods of **Component** class.

(i) add( ): To include a control in the window.

Syntax: add (Component object);

(ii) remove( ): To remove a control from the window.

Syntax: remove (Component object);

(iii) removeAll( ): To remove all controls from the window.

Syntax: removeAll( );

COMPONENTS

(1) Label: To display a label. The Label class has three constructors:

(i) Label object = new Label( );
(ii) Label object = new Label(String st);
(iii) Label object = new Label(String st, int alignment);

The values of alignment are:

Label.LEFT	- 0
Label.RIGHT	- 1
Label.CENTER	- 2

These are Label class constants. Default alignment of text in a Label is **LEFT**.

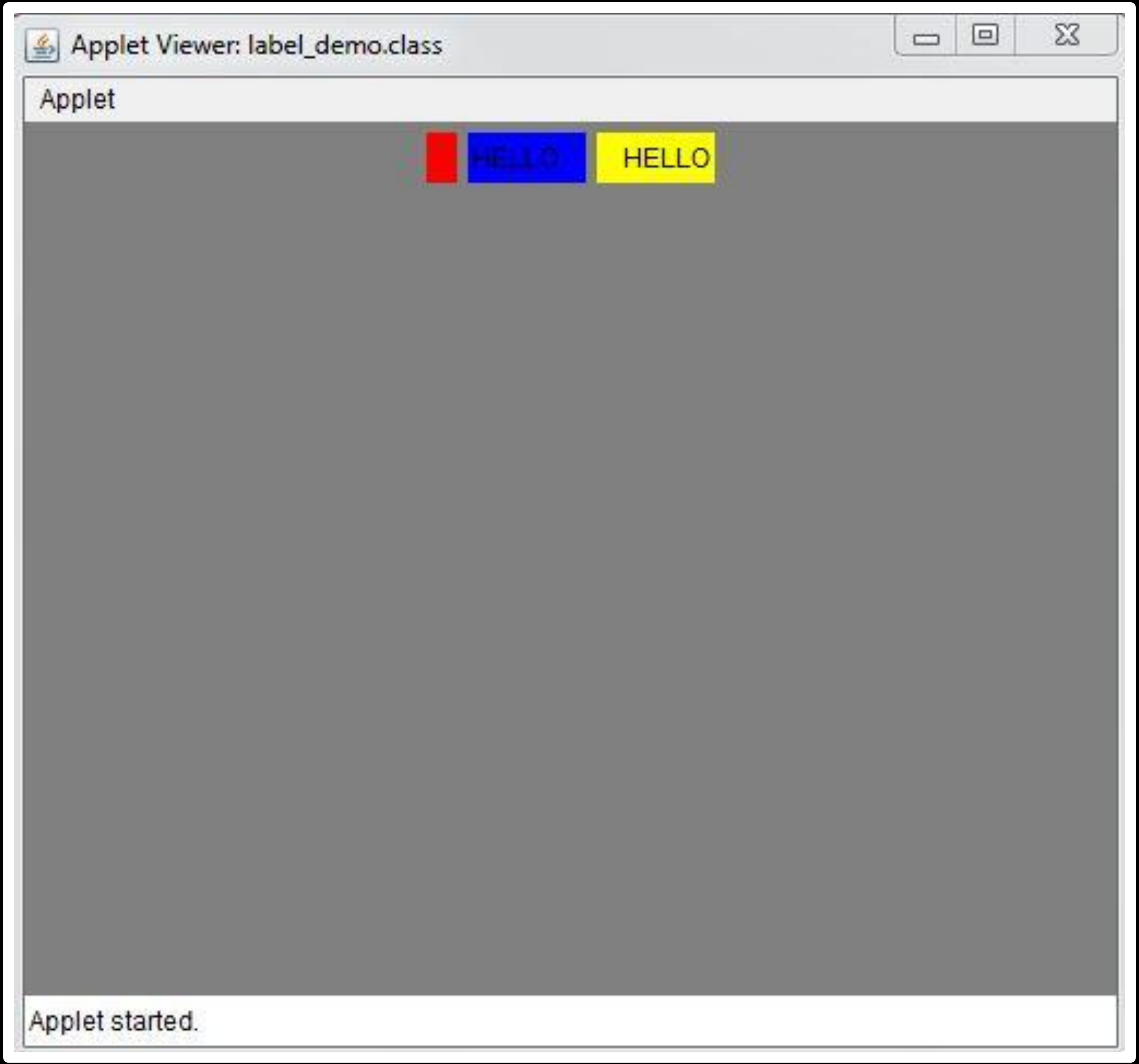
// To display 3 labels with background colour **(label_demo.java)**

```
   import java.awt.*;
   import java.applet.*;
   public class label_demo extends Applet
   {       Label lab1,lab2,lab3;
           public void init()
          {    setBackground(Color.gray);
               lab1 = new Label();
               lab2 = new Label("HELLO");
               lab3 = new Label("HELLO",Label.RIGHT);
               lab1.setBackground(Color.red);
               lab2.setBackground(Color.blue);
               lab3.setBackground(Color.yellow);
               add(lab1);
               add(lab2);
               add(lab3);
          }        }
```

label_demo.html

```
<HTML>
        <BODY>
        <H1> LABEL DISPLAY</H1>
 <APPLET CODE=“label_demo.class” width=“500” height=“400”>
 </APPLET>
        </BODY>
       </HTML>
```

Output



Applet started.

(2) Button: To create a push button. It contains a label and generate an event when it is pressed. It has two constructors:

(i) Button object = new Button();
  Empty button without label

(ii) Button object =new Button(String);
  Button with a label name

ActionListener:

- It is an interface.
- First of all the button must be registered.
- So that the action can be sent to the listener.

Syntax:

```
button_object.addActionListener(this);
```

actionPerformed( ): It receives an object of ActionEvent kind on which an action is performed. Then inside it there is a method **getActionCommand( )** which returns the label value of that particular button which was pushed earlier.

Syntax:

```
        public void actionPerformed(ActionEvent object)
        {
            String st = object.getActionCommand( );
            // statement(s);
        }
```

// display three buttons and display message accordingly while pushing a particular button (**button_demo.java**).

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class button_demo extends Applet  implements
ActionListener
{
      String st= " ";
      Button y, n, m;
      public void init()
      {
           y = new Button("YES");
           n = new Button("NO");
          m = new Button("MAY BE");
```

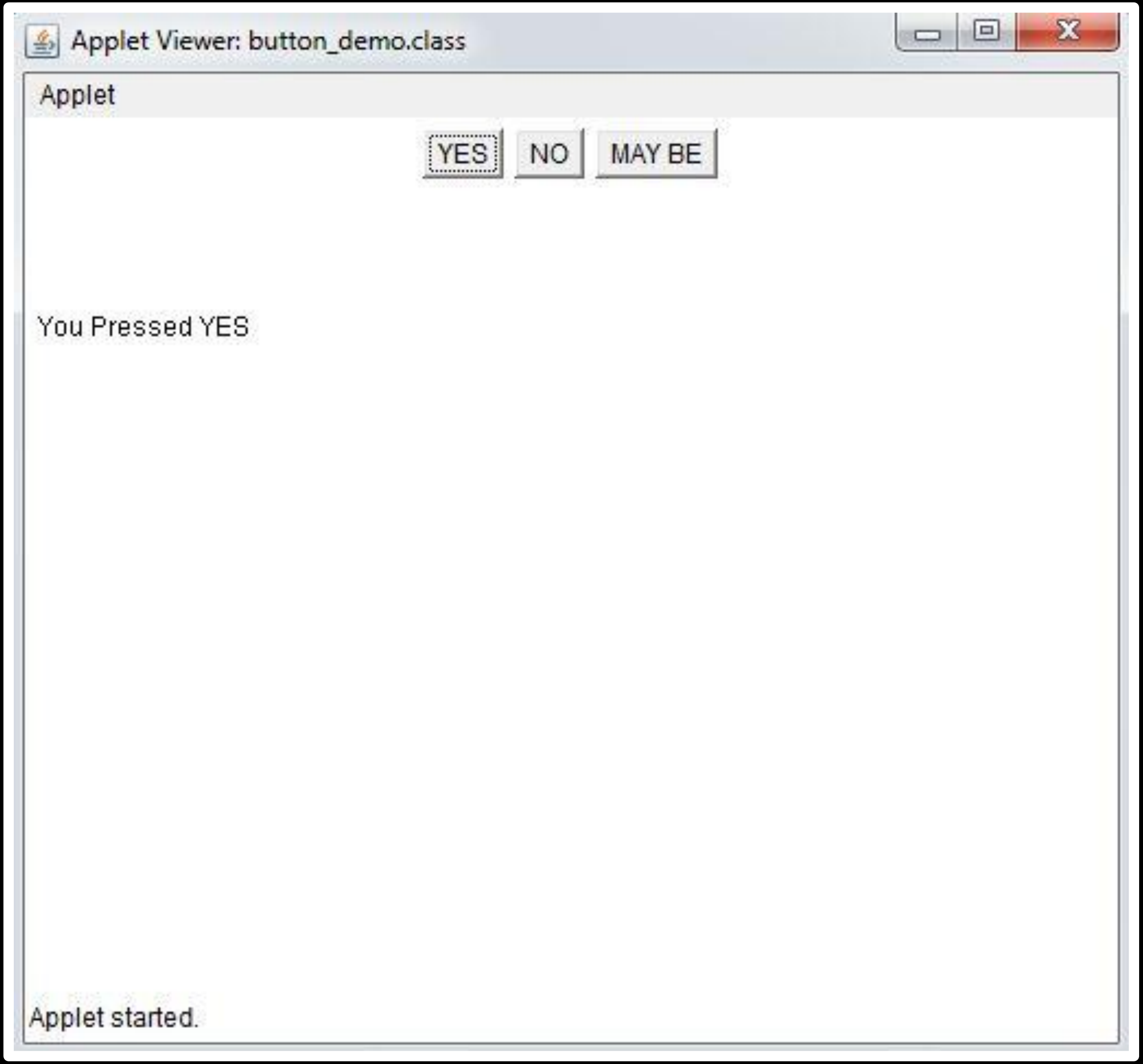
```
    add(y);
    add(n);
    add(m);
    // To register for action event
    y.addActionListener(this);
    n.addActionListener(this);
    m.addActionListener(this);
  }
```

```
public void actionPerformed(ActionEvent ae)
{
       String a = ae.getActionCommand();

       if(a.equals("YES") ) st = "You Pressed YES";

       if(a.equals("NO") ) st = "You Pressed NO";

       if(a.equals("MAY BE") ) st = "You Pressed MAY BE";

       repaint(); // calling paint() for display
  }
public void paint(Graphics g)
{
    g.drawString(st,6,100);
 }
}
```

button_demo.html

```
<HTML>
    <BODY>
        <CENTER>
                        <H1> BUTTON DEMO </H1>
        </CENTER>
        <APPLET CODE=button_demo.class width=500 height=400>
        </APPLET>
    </BODY>
</HTML>
```

Output



```
You Pressed YES
```

```
Applet started.
```

```
// display three buttons and display colour of back  ground
// accordingly while pushing a particular button
//                              (button_demo1.java)
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class button_demo1 extends Applet  implements
ActionListener
{
    String st= " ";
    Button r,g,b;
    public void init()
   {
             r = new Button("RED");
             g = new Button("GREEN");
             b = new Button("BLUE");
```

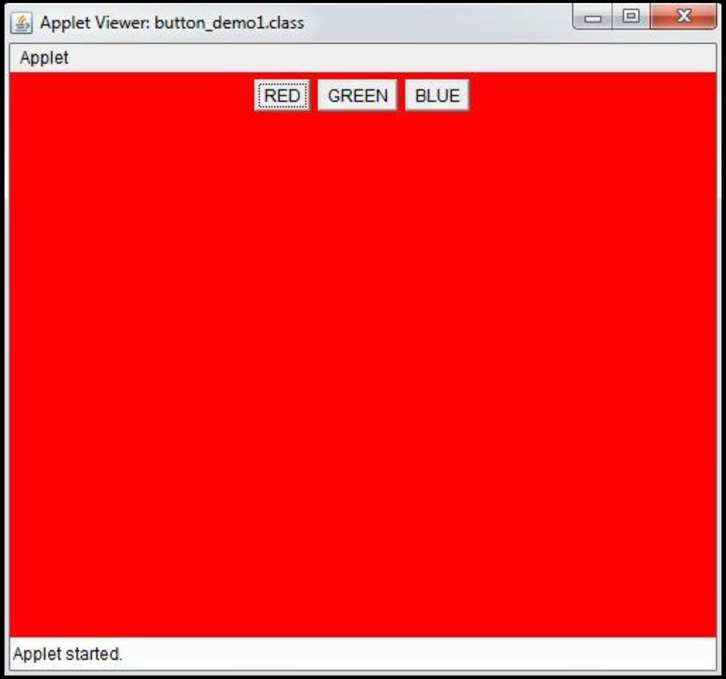
```
                 add(r);
                 add(g);
                 add(b);

                // to register for action event
                r.addActionListener(this);
                g.addActionListener(this);
                b.addActionListener(this);
     }
```

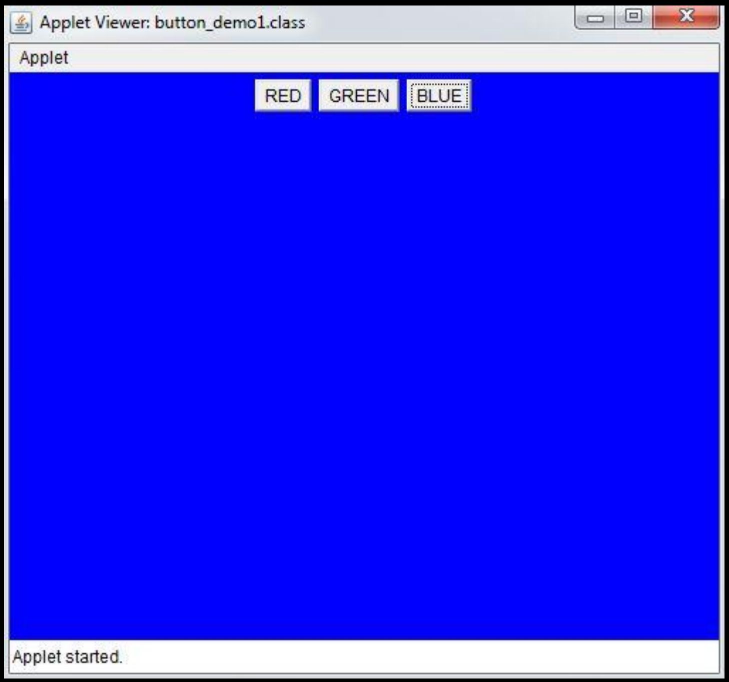
```
public void actionPerformed(ActionEvent ae)
{
   String a = ae.getActionCommand();
   if(a.equals("RED") )  setBackground(Color.red);

   if(a.equals("GREEN") ) setBackground(Color.green);

   if(a.equals("BLUE") )   setBackground(Color.blue);

   repaint(); // calling paint() for display
 }
 // no paint method needs because there is no displaying
 // of any string or graphics.
}
```

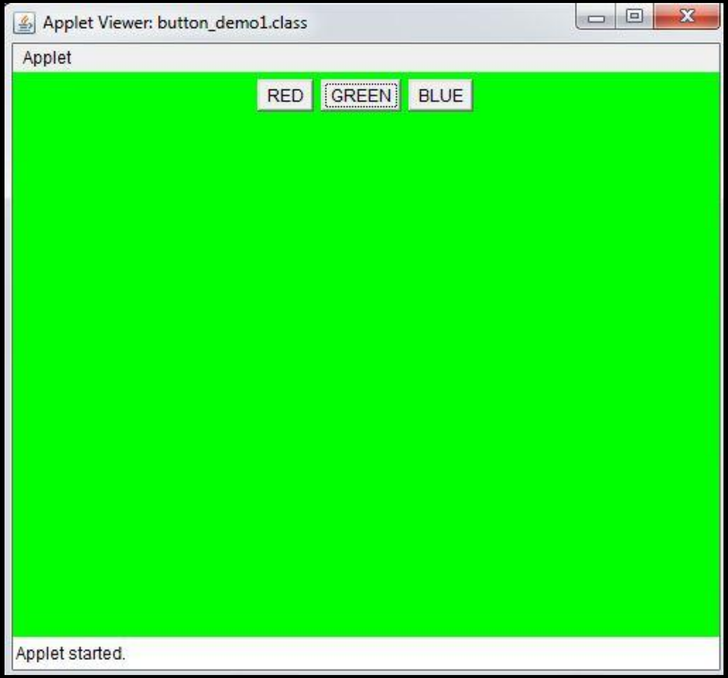
button_demo1.html

```
<HTML>
      <BODY>
          <CENTER>
   <H1> BUTTON DEMO WITH BACKGROUND COLOUR </H1>
          </CENTER>
   <APPLET CODE=button_demo1.class width=500 height=400>
   </APPLET>
      </BODY>
</HTML>
```

Output


Applet Viewer: button_demo1.class
Applet
RED
GREEN
BLUE
Applet started.


Applet Viewer: button_demo1.class
Applet
RED
GREEN
BLUE
Applet started.


Applet Viewer: button_demo1.class
Applet
RED
GREEN
BLUE
Applet started.

(3) Check Boxes:

A check box is a control that is used to turn an option **on** or **off**.

It consists of a small box that can either contain a check mark or not.

It has three constructors :

(i) Checkbox object = new Checkbox( );

(ii) Checkbox object = new Checkbox (String);

(iii) Checkbox object = new Checkbox (String, boolean);

It has the following methods and Interface:

(i) **boolean getState( ):** To get the current state of the checkbox.

(ii) **void setState(boolean):** To change the state of a particular check box.

(iii) **String getLabel( ):** To get the current label associated with the checkbox.

(iv) **void setLabel(String):** To change the label of check box.

(v) ItemListener: It is an interface. First of all the check boxes must be registered. So that the action can be sent to the listener.

Syntax: **Checkbox_object.addItemListener (this);**

(vi) itemStateChanged( ): It receives an object of **ItemEvent** kind on which an action is performed.

Syntax:

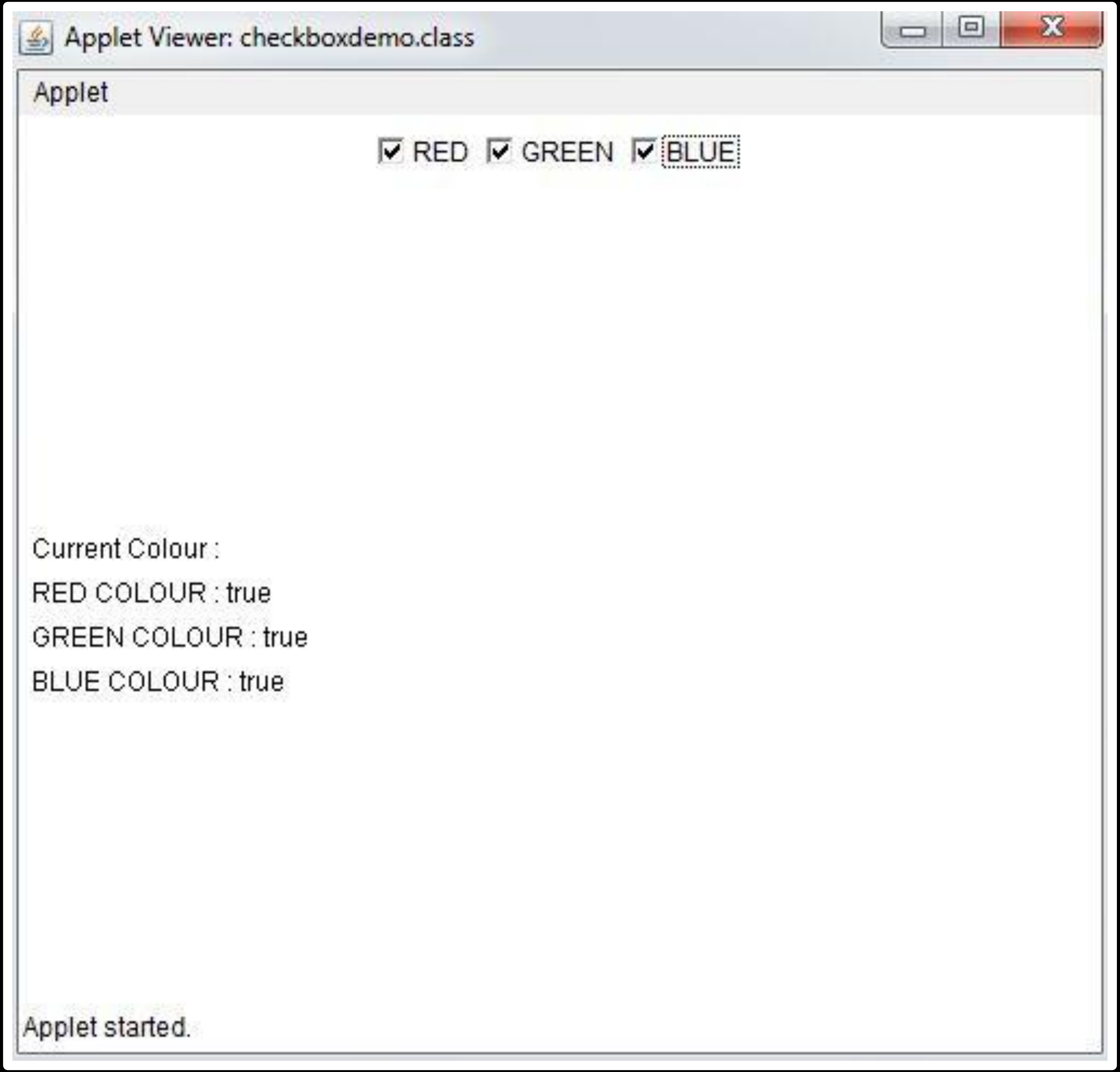
```
 public void itemStateChanged (ItemEvent   object)
{
 // statement(s);
 }
```

```
//selection of one or more than one checkboxes
//                    (checkboxdemo.java)
 import java.awt.*;
 import java.applet.*;
 import java.awt.event.*;
 public class checkboxdemo extends Applet implements
                                          ItemListener
 {
     String st;
     Checkbox r, g ,b;
     public void init()
     {
         r = new Checkbox("RED", false);
         g = new Checkbox("GREEN", true);
         b = new Checkbox("BLUE");
```

```
    add(r);
    add(g);
    add(b);

    // registration for event i.e click effect
    r.addItemListener(this);
    g.addItemListener(this);
    b.addItemListener(this);
  }
```

```
public void itemStateChanged(ItemEvent ie)
{
    repaint();
}
public void paint(Graphics gr)
{
    st = "Current Colour : ";
    gr.drawString(st, 6, 200);
    st = "RED COLOUR : " + r.getState();
    gr.drawString(st, 6, 220);
    st = "GREEN COLOUR : " + g.getState();
    gr.drawString(st, 6, 240);
    st = "BLUE COLOUR : " + b.getState();
    gr.drawString(st, 6, 260);
 }
}
```

Output



```
Current Colour :
RED COLOUR : true
GREEN COLOUR : true
BLUE COLOUR : true
```

```
Applet started.
```

(4) Check Box Group:

It allows to group a set of check boxes. So that at a time only one check box can be checked. So the check boxes as a group is called 'radio buttons'.

It has only one constructor to create a check box group.

Syntax:

Checkbox object = new Checkbox (String, CheckboxGroup, boolean);

Example: **CheckboxGroup cbg;**
boolean val = true;

Checkbox cb = new Checkbox("Colour" ,cbg, val);

It creates a checkbox object called **cb** whose label is **'Colour'** and group object is **cbg**. The initial state of the check box is **'true'**.

It has the same methods and Interface of Checkbox class and a method called **getSelectedCheckbox( )** which returns the selected check box reference, which is to be invoked through CheckboxGroup object.

Syntax:

Return value Checkbox kind:

```
    CheckboxGroup_object.getSelectedCheckbox( )
```

Return value String kind:

```
    String  Checkbox_object.getLabel( )
```

(OR)

Return value String kind:

```
   CheckboxGroup_object.getSelectedCheckbox( ).getLabel( )
```

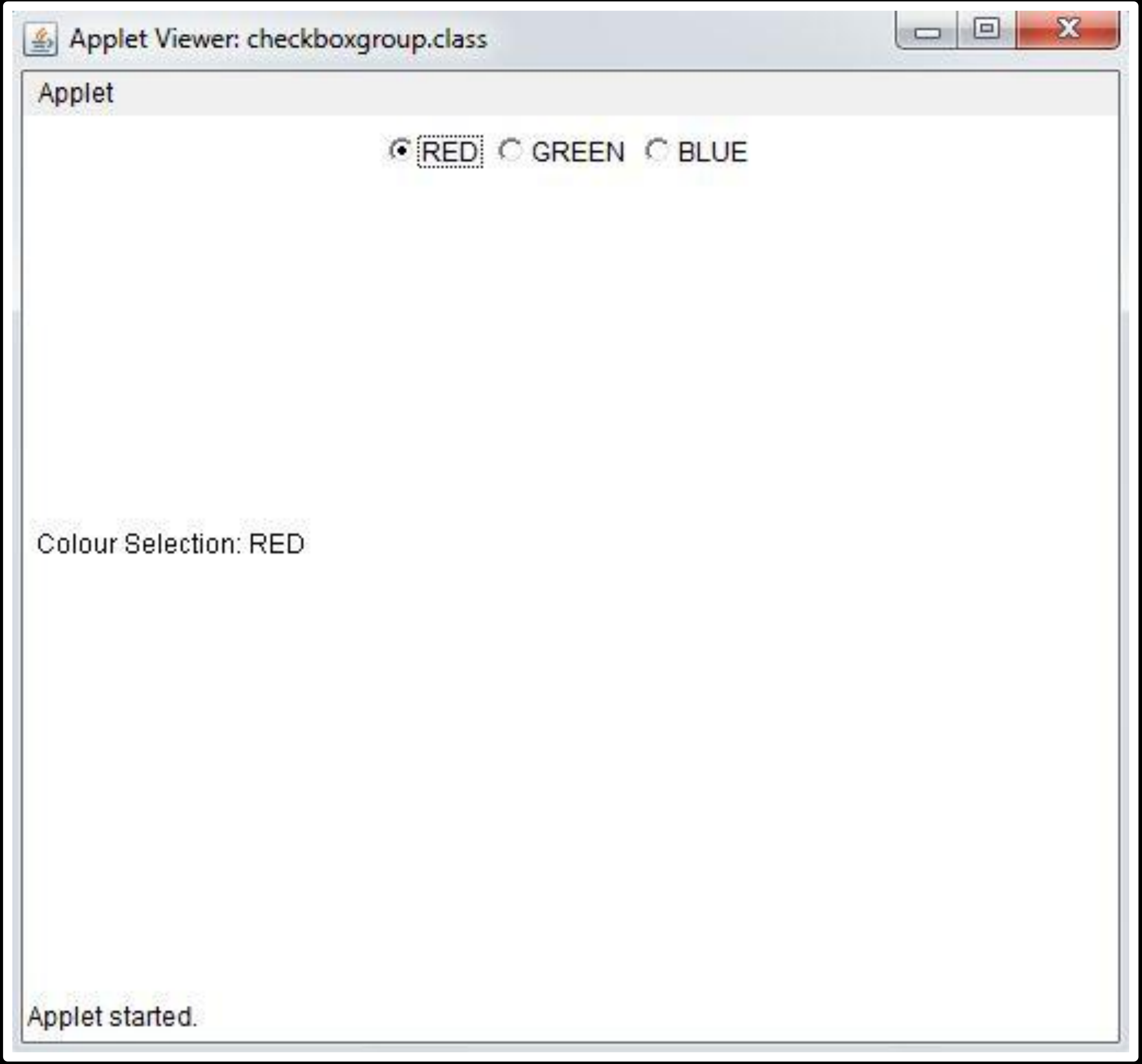
```
 // selection of one checkbox at a time by using
// CheckboxGroup class (checkboxgroup.java).
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class checkboxgroup extends Applet implements
                                                ItemListener
{
        String st;
        Checkbox r, g, b;
        CheckboxGroup cbg;
        public void init()
        {
           cbg = new CheckboxGroup();
        r = new Checkbox("RED", cbg, false);
        g = new Checkbox("GREEN", cbg, true);
        b = new Checkbox("BLUE", cbg, false);
        add(r);
        add(g);
        add(b);

        // registration for event i.e click effect
        r.addItemListener(this);
        g.addItemListener(this);
        b.addItemListener(this);
    }
```

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

 public void paint(Graphics gr)
 {
    st = "Colour Selection: ";
    st += cbg.getSelectedCheckbox().getLabel();
    gr.drawString(st,6,200);
  }
}
```

checkboxgroup.html

```
<HTML>
    <BODY>
       <H1> CHECK BOX GROUP DISPLAY</H1>
       <APPLET CODE=checkboxgroup.class width=500 height=400>
       </APPLET>
    </BODY>
</HTML>
```

Output


Applet Viewer: checkboxgroup.class
Applet
[● RED] [○ GREEN] [○ BLUE]

```
Colour Selection: RED
```

```
Applet started.
```

```
// selection of one checkbox at a time by using
// CheckboxGroup class and change the background colour
// of the applet (checkboxgroup1.java).
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class checkboxgroup1 extends Applet implements
                                                    ItemListener
{
    String st;
    Checkbox r, g, b;
    CheckboxGroup cbg;
    public void init()
    {
        cbg = new CheckboxGroup();
```

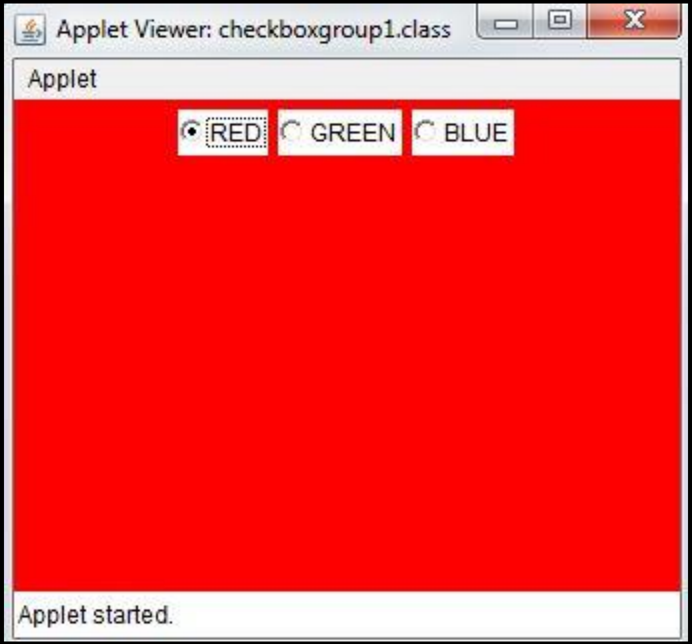
```
     r = new Checkbox("RED", cbg, false);
     g = new Checkbox("GREEN", cbg, false);
     b = new Checkbox("BLUE", cbg, false);
     add(r);
     add(g);
     add(b);

    // registration for event i.e click effect
    r.addItemListener(this);
    g.addItemListener(this);
    b.addItemListener(this);
 }
```

```
public void itemStateChanged(ItemEvent ie)
{
    Checkbox cb = cbg.getSelectedCheckbox();

    st = cb.getLabel();

    if(st.equals("RED")) setBackground(Color.red);

    if(st.equals("GREEN")) setBackground(Color.green);

    if(st.equals("BLUE")) setBackground(Color.blue);
  }
}
```

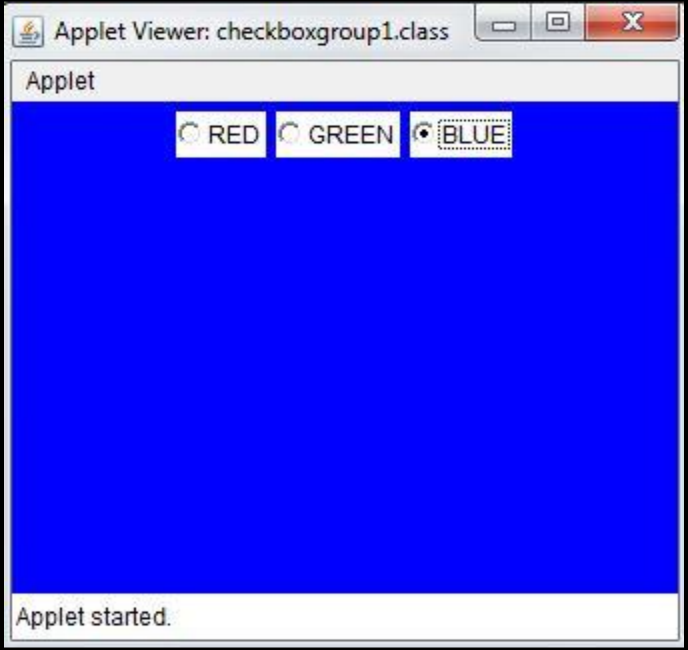
checkboxgroup1.html

```
<HTML>
     <BODY>
     <CENTER>
           <H1> CHECK BOX GROUP SELECTION AND CHANGE
                              OF APPLET BACKGROUND COLOUR</H1>
      </CENTER>
      <APPLET CODE=checkboxgroup1.class width=500 height=400>
      </APPLET>
      </BODY>
</HTML>
```

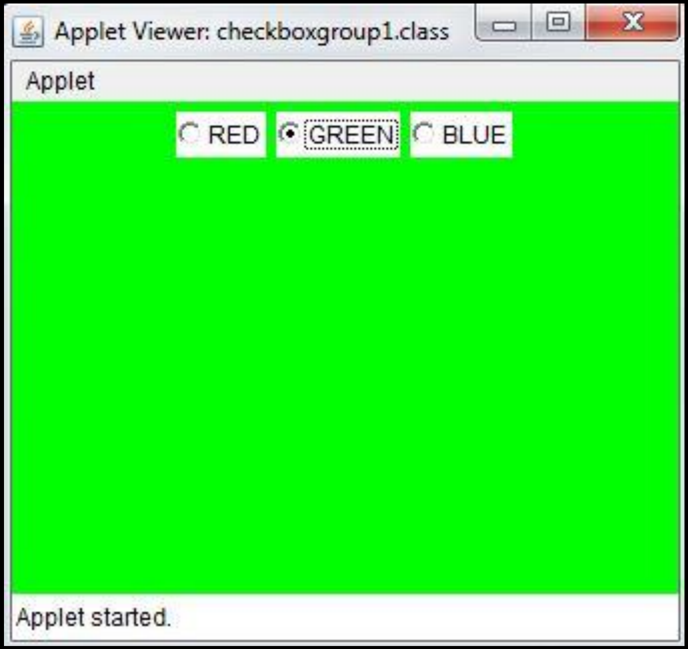
Output



Applet started.



Applet started.



Applet started.

(5) List box:

The List class provides **a multiple choice** and **scrolling selection list of items**.

The Choice class allows to display one item at a time. Where as List class allows to display a list or set of items.

It also allows to select multiple selection of items at a time.

It has 3 constructors used to create a list box.

(i) List object = new List( ); It allows to select one item at a time.

(ii) List object = new List(int no_of_rows); It allows to show no_of_rows items visible at a time. Remaining items can be scrolled to view when needed. And allows to select one item at a time.

(iii) List object =
new List (int no_of_rows, boolean multi_select);

If **multi_select** is **true** then the user can select more than one items at a time.

The method **add( )** used for adding items in the list box.

(a) List_object.add(String): It adds an item at the end of the list.

(b) List_object.add(String name, int index): It adds an item at the specified position specified by 'index'. The initial index starts from 0. To add item at the end, specify -1 as the index value.

// selection of item(s) from list box (**listemo.java**).

```
 import java.awt.*;
 import java.applet.*;
 import java.awt.event.*;
 public class listdemo extends Applet implements
                                                      ActionListener
 {
     String st;               // in Choice it is ItemListener
     List os, lan;            // but in List it is ActionListener
   public void init( )
  {
    // 4 rows and multiselection is true
    os  = new List(4, true);
    lan = new List(3);   // to displays 3 no. of items
```

```
// adding 4 items in the list os
     os.add("DOS");
     os.add("LINUX");
     os.add("WINDOWS");
     os.add("UNIX");
 // adding 5 items in the list lan
     lan.add("Java");
     lan.add("C");
     lan.add("C++");
     lan.add("C#");
     lan.add("VB");
 // add both the lists to the window
     add(os);
     add(lan);
 // registration for event i.e click effect
   os.addActionListener(this);
   lan.addActionListener(this);     }
```

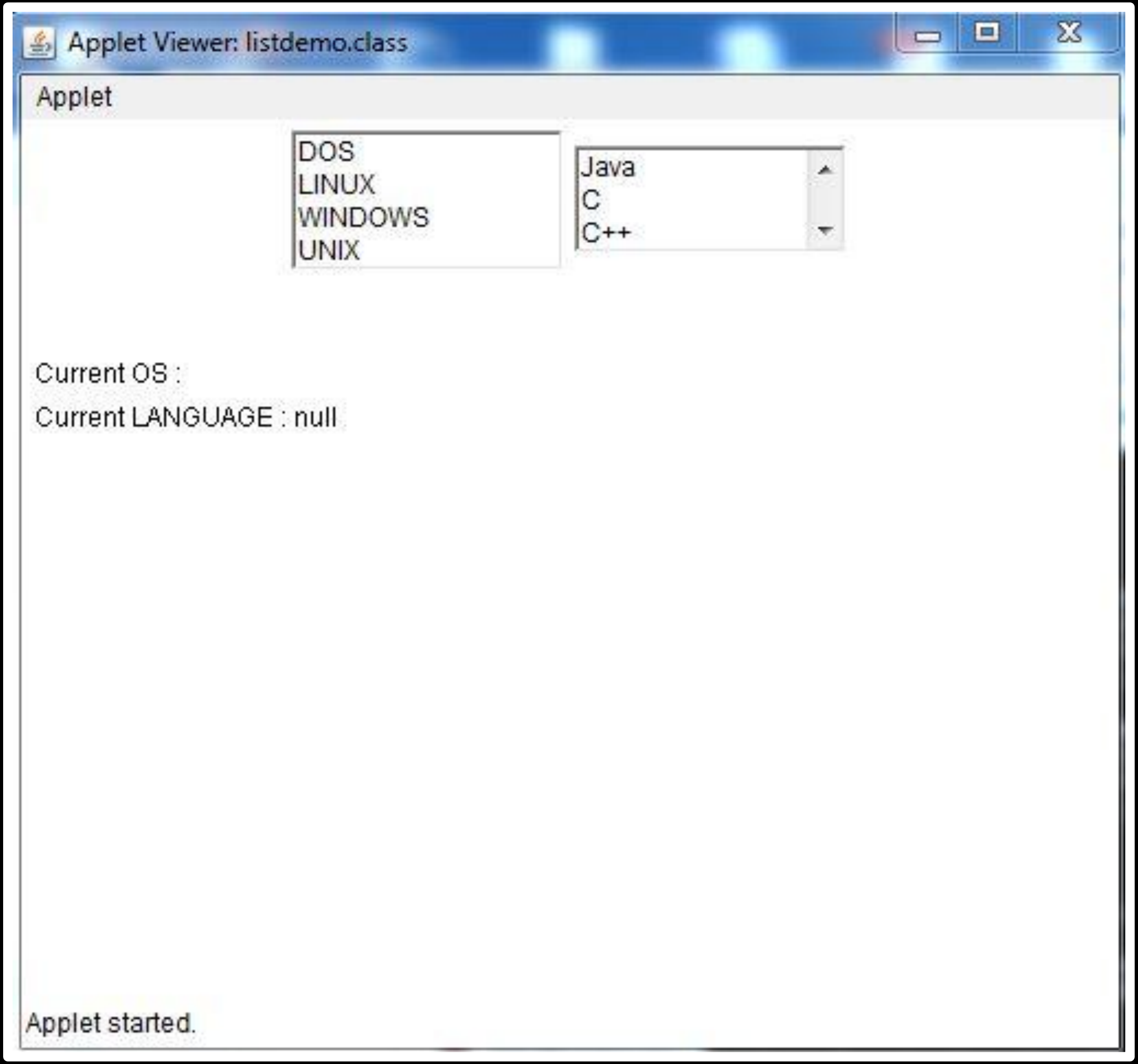
```
public void actionPerformed(ActionEvent ae)
{
   repaint();
 }
```

```
public void paint(Graphics g)
  {
     int index[ ];
     st = "Current OS :";

// all selected items indices are assigned to the array index
 index = os.getSelectedIndexes();
    for(int i=0; i<index.length; i++)
       st = st + os.getItem(index[i]) + "  ";

       g.drawString(st, 6, 120);
       st = "Current LANGUAGE : ";
       st = st + lan.getSelectedItem();
       g.drawString(st, 6, 140);
  }
 }
}
```

list_demo.html

```
<HTML>
   <BODY>
    <H1> TEXT BOX DISPLAY </H1>
     <APPLET CODE=list_demo.class width=500 height=400>
     </APPLET>
   </BODY>
</HTML>
```

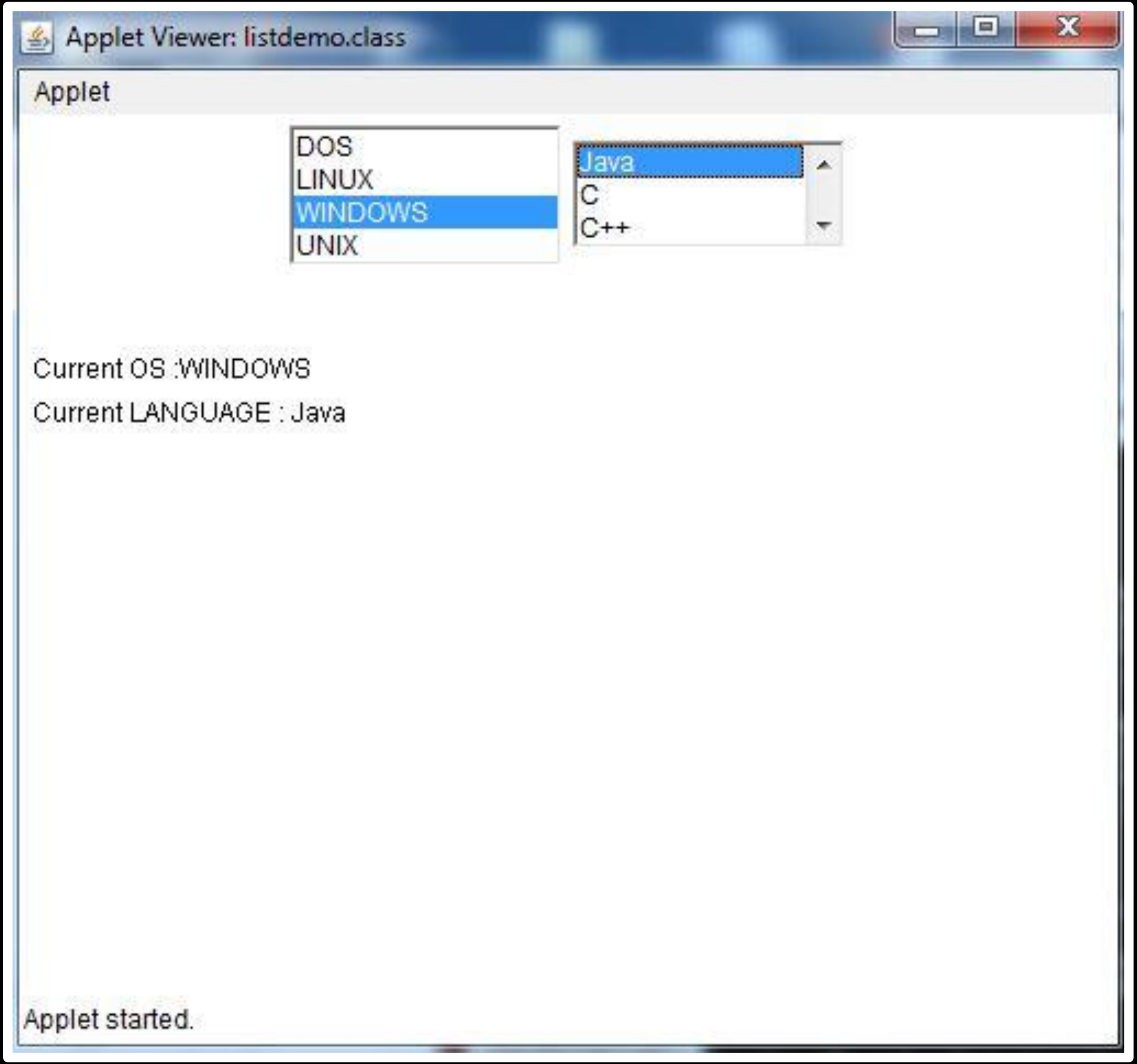
Output



```
Current OS :
Current LANGUAGE : null
```

```
Applet started.
```

Output



```
Current OS :WINDOWS
Current LANGUAGE : Java
```

```
Applet started.
```

(6)Textbox:

The **TextField** class implements a single-lined text entry area, usually called an 'edit control’.

It allows the user to enter text, edit it using arrow keys, cut and paste and selection using mouse.

TextField is the sub-class of **TextComponent**.

It has four constructors. They are:

(i) TextField object = new TextField( );
(ii) TextField object = new TextField (int length);
(iii) TextField object = new TextField(String text);
(iv) TextField object = new TextField(String text , int length);

Methods to **get string from** and **set string to** the TextField:
(a) String getText( ): To get the string currently contained in the text field.

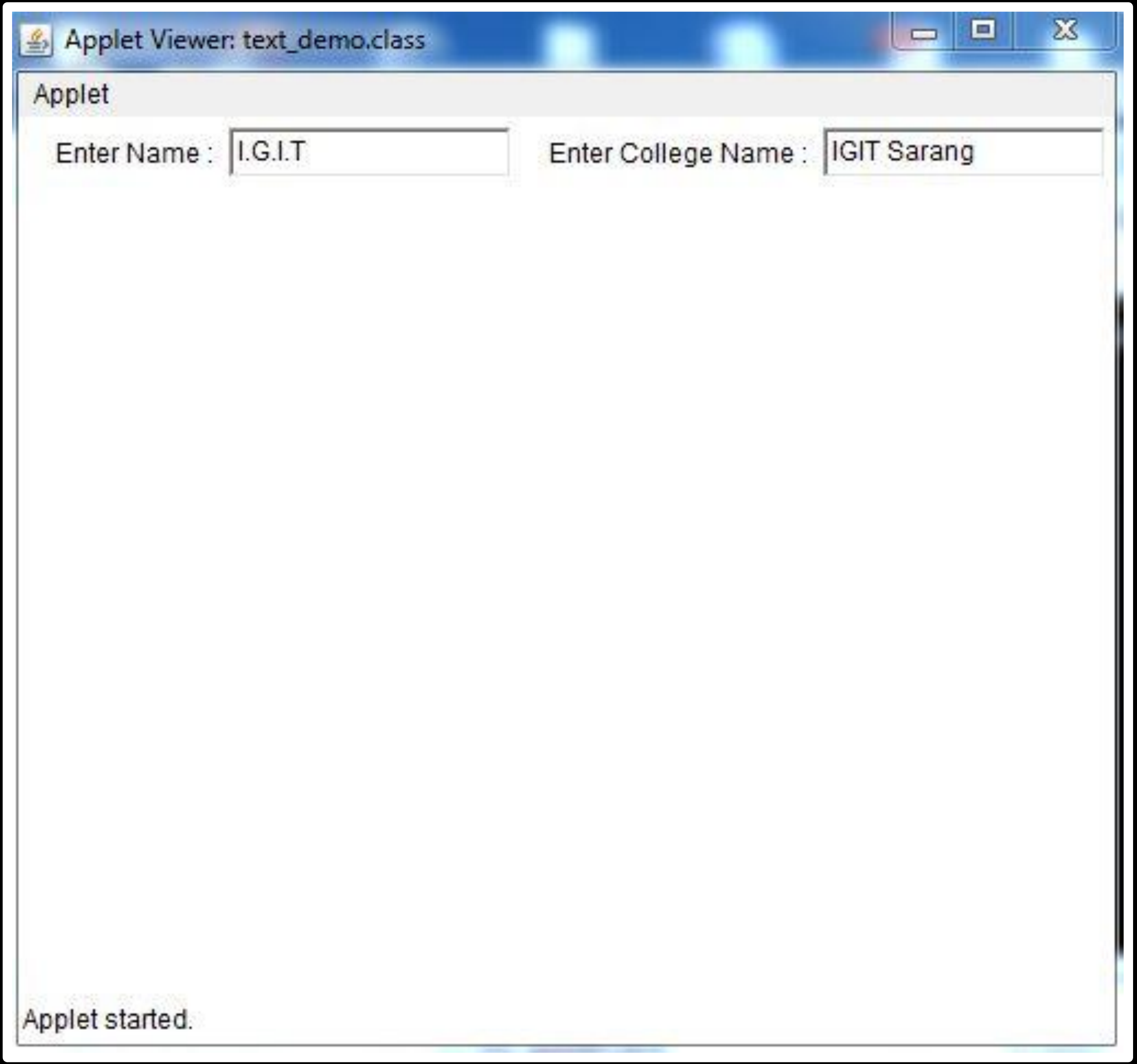
(b) void setText(String): To set the old text with a new text in a text field.

```
// text field with echo character (text_demo.java).
import java.awt.*;
import java.applet.*;
public class text_demo extends Applet
{
    TextField name1, name2;
     public void init( )
     {
            name1 = new TextField("I.G.I.T", 15);
            name2 = new TextField(15);
Label L1 = new Label("Enter Name :", Label.RIGHT);
Label L2= new Label("Enter College Name :", Label.RIGHT);
            add(L1);
            add(name1);
            add(L2);
            add(name2);
    }
 }
```

text_demo.html

```
<HTML>
   <BODY>
    <H1> TEXT BOX DISPLAY </H1>
     <APPLET CODE=text_demo.class width=500 height=400>
     </APPLET>
   </BODY>
</HTML>
```

Output



Enter Name : I.G.I.T

Enter College Name : IGIT Sarang

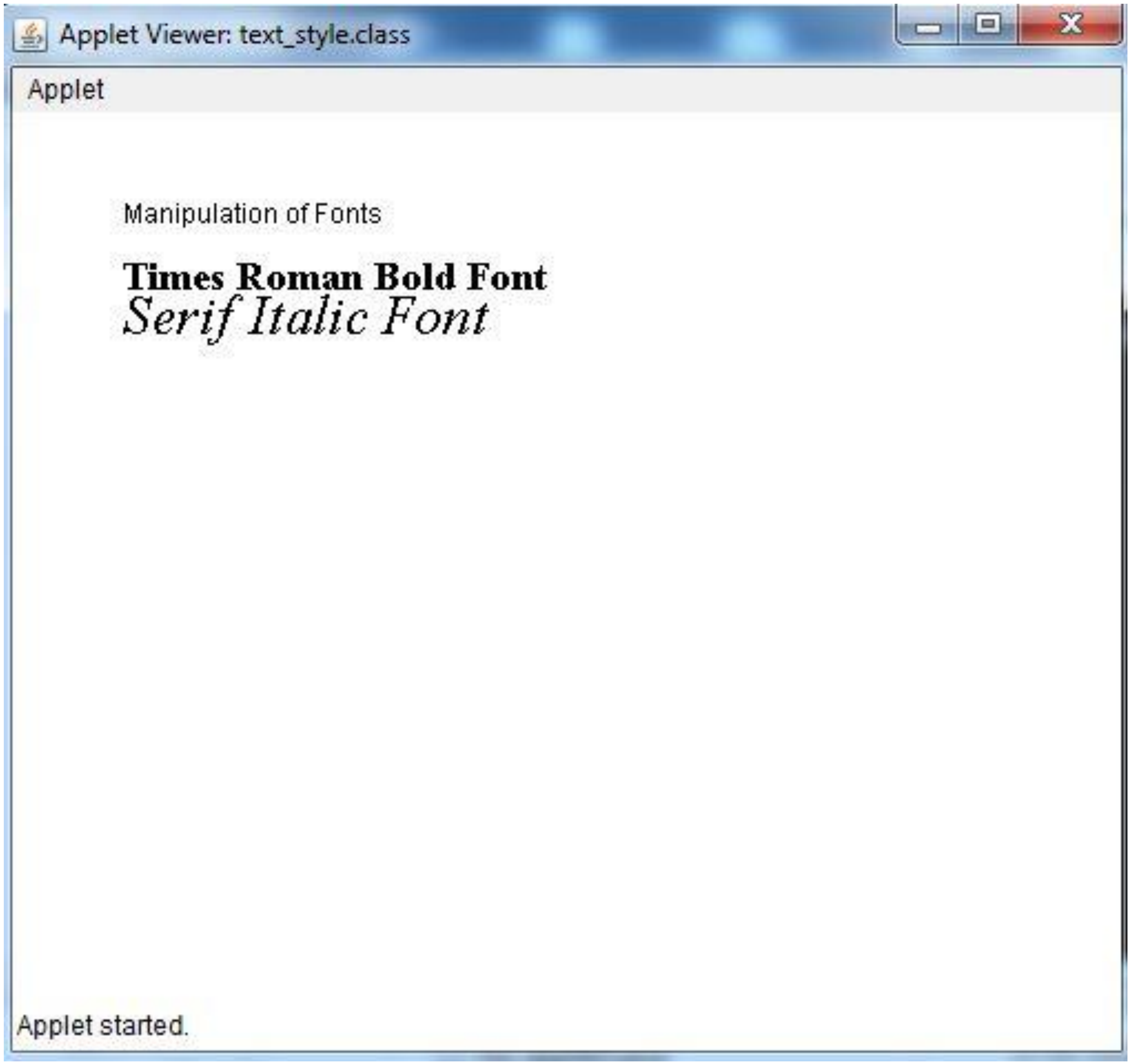
Applet started.

```
 // displaying texts with different style in an applet
//                          text_style.java
   import java.awt.*;
   import java.applet.*;
   public class text_style extends Applet
  {
        Font f1,f2;
      public void init()
     {
        f1 = new Font("TimesRoman", Font.BOLD, 18);
        f2 = new Font("Serif", Font.ITALIC, 25);
      }
public void paint(Graphics g)
 {
        g.drawString("Manipulation of Fonts",50,50);
        g.setFont(f1);
         g.drawString("Times Roman Bold Font",50,80);
         g.setFont(f2);
         g.drawString("Serif Italic Font",50,100);
   }
}
```

text_style.html

```
<HTML>
   <BODY>
     <H1> TEXT STYLE </H1>
     <APPLET CODE="text_style.class" width="500" height="400">
     </APPLET>
   </BODY>
</HTML>
```

Output



```
Manipulation of Fonts
Times Roman Bold Font
Serif Italic Font
```

Applet started.

MODULE-4

Module 4: -

Chapter 1:- Swing (JFC)

Introduction Diff b/w awt and swing, Components Hierarchy, Panes, Individual Swings Components JLabel, JButton, JTextField, JTextArea.

Chapter 2:- JavaFX

Getting started with JavaFX, Graphics, User Interface Components, Effects, Animation, and Media, Application Logic, Interoperability, JavaFX Scene Builder 2, Getting Started with scene Builder. Working with scene Builder.

AWT	SWING(JFC)
When a component is created, it internally calls a native method (i,e., a C function). This means AWT internally depends on C code. This component is called "peer" component. So AWT is also called "peer component based" model.	JFC is the extension of original AWT. The entire JFC is developed in pure Java.
Components are heavy weight. The components take more system resource like more memory and processor time.	Java foundation classes (JFC) components are light weight. It utilizes minimum system resources
It's execution is slow.	It's execution is fast.
The components "look-and-feel" changes depending on the platform (or OS).	The components "look-and-feel" remains same throughout all the platform (or OS).
It does not offer pluggable look-and-feel" feature.	It offers "pluggable look-and-feel" feature, which allows the programmer to change the look and feel as suited for a platform.

JFC

Java Foundation Classes represent a class library developed on pure Java which is an extension of AWT.

Packages of JFC

- **javax.swing**: To develop components like push buttons, radio buttons, menus, etc. The x in javax represents that it is an extended package whose classes are derived from java.awt package.

- **javax.swing.plaf:** plaf stands for pluggable look and feel.

- **javax.awt.dnd:** dnd stands for drag and drop.

Packages of JFC

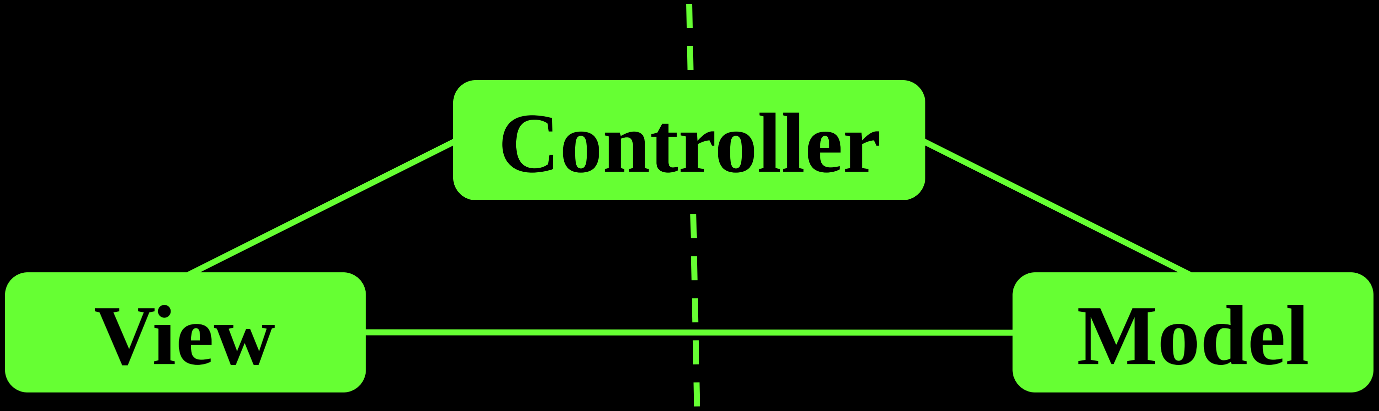
- **javax.accessibility:** It is useful to develop applications in such a way that physically challenged people are able to use the applications. For example, a blind person will be able to convert a printer report into Braille language so that he can read it or a deaf person will be able to enjoy the music which is converted and sent to his ears through his hearing-aid.

- **javax.awt.geom:** geom stands for geometrical shapes. It helps to draw 2D graphics, filling color, rotating the shapes, etc.

javax.swing and **MVC**

- Javax.swing is the most important and most commonly used package among all the other packages of JFC.
- This package provides classes to create components like push buttons, check boxes, radio buttons, menus, etc.
- All the components in swing follow a model-view-controller (MVC) architecture.
- Model represents the data that represents the state of a component. For example, for push button whether it is pressed or not, or whether it is selected or not will be stored in an object called "model".
- View represents the visual appearance of the of the component based on the model data.

MVC Architecture



The goal of the **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**).

Window Panes

- It represents a free area of a window where some text or component can be displayed.

- There are four types of window panes available in javax.swing package. These panes can be imagined like transparent sheetes lying one below the other.

Four types of panes are:

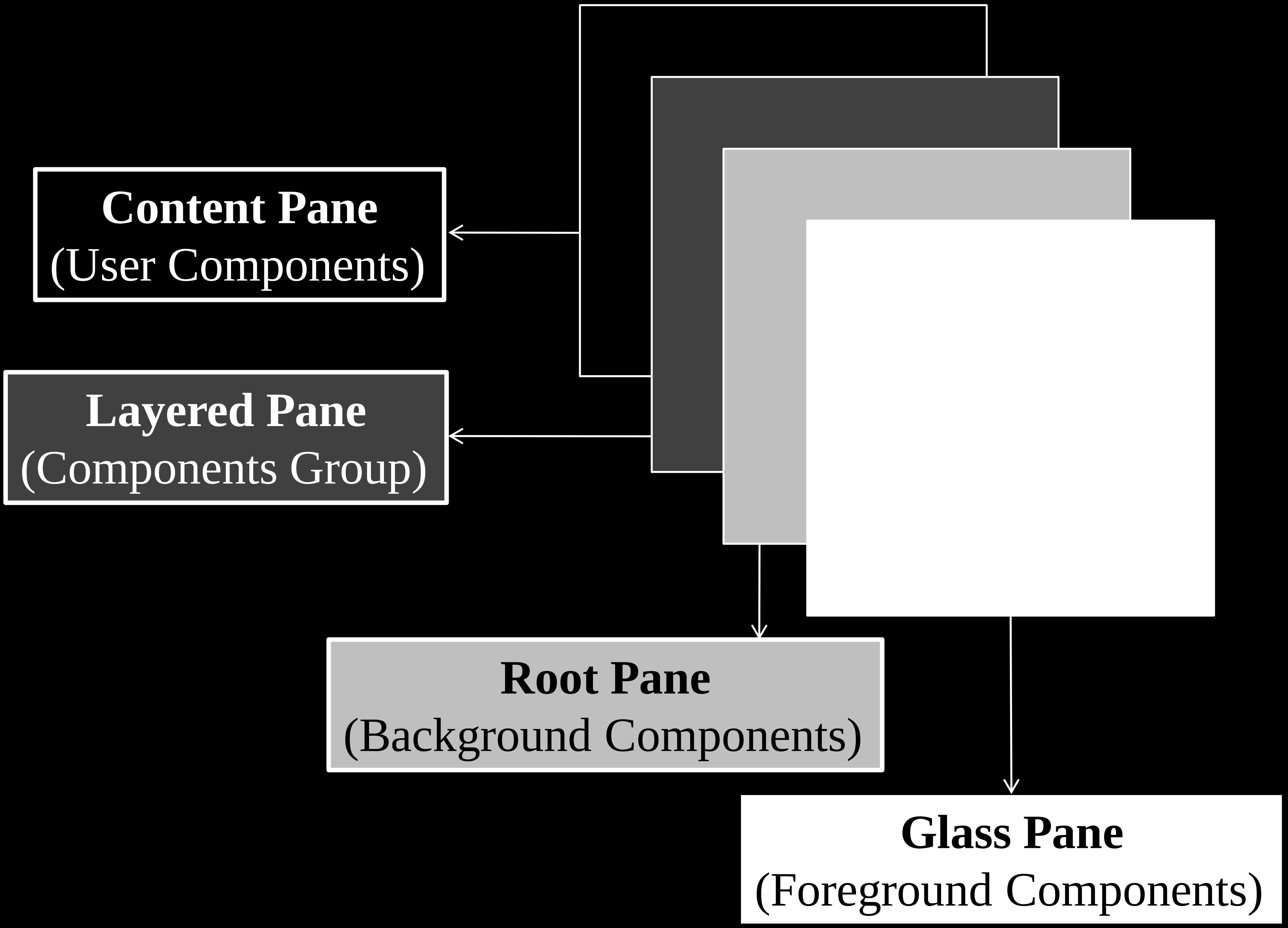
i. Glass pane: It is very close to the monitor’s screen. Any components to be displayed in the foreground are attached to this pane. The method getGlassPane() of JFrame class is used to reach the glass pane. The method returns Component class object.

ii. Root pane: It is below the glass pane. Any component to be displayed in the background are displayed in this pane. For example, to display flying aeroplane in the sky. The aeroplane can be displayed as .gif or .jpg file in the glass pane where as the sky can be displayed in the root plane in the background. The method getRootPane() of JRootPane class is used to reach the glass pane. The method returns JRootPane class object.

iii. Layered pane: It lies below the root pane. To make several components into a group, the group must be attached in the layered pane. The method getLayeredPane() of JFrame class is used to reach the glass pane. The method returns getLayeredPane class object.

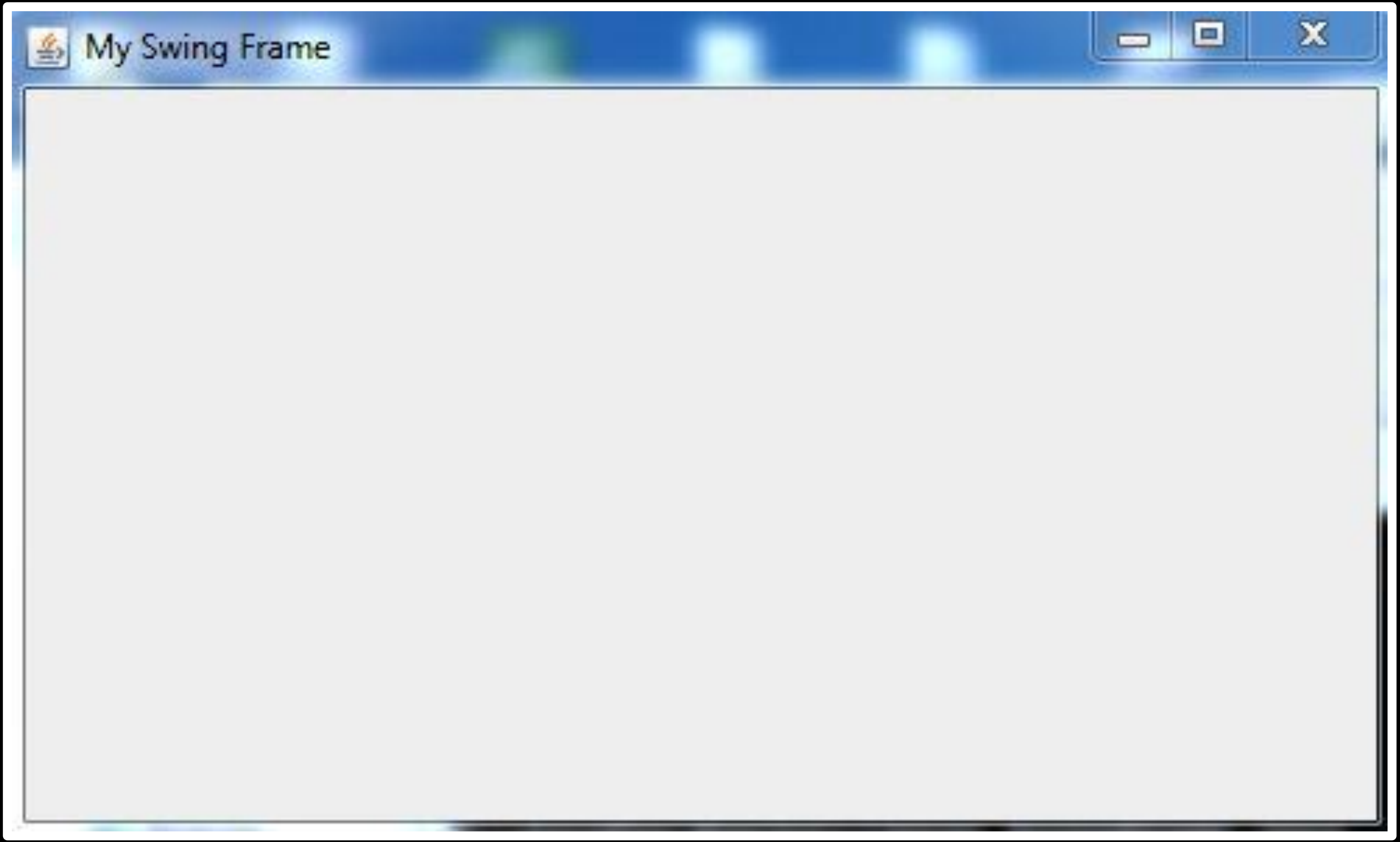
iv. Content pane: It is the bottom most pane of all the four panes. Individual components are attached to this pane. The method getContentPane() of JFrame class is used to reach the glass pane. The method returns Component class object.

Window Panes and their use



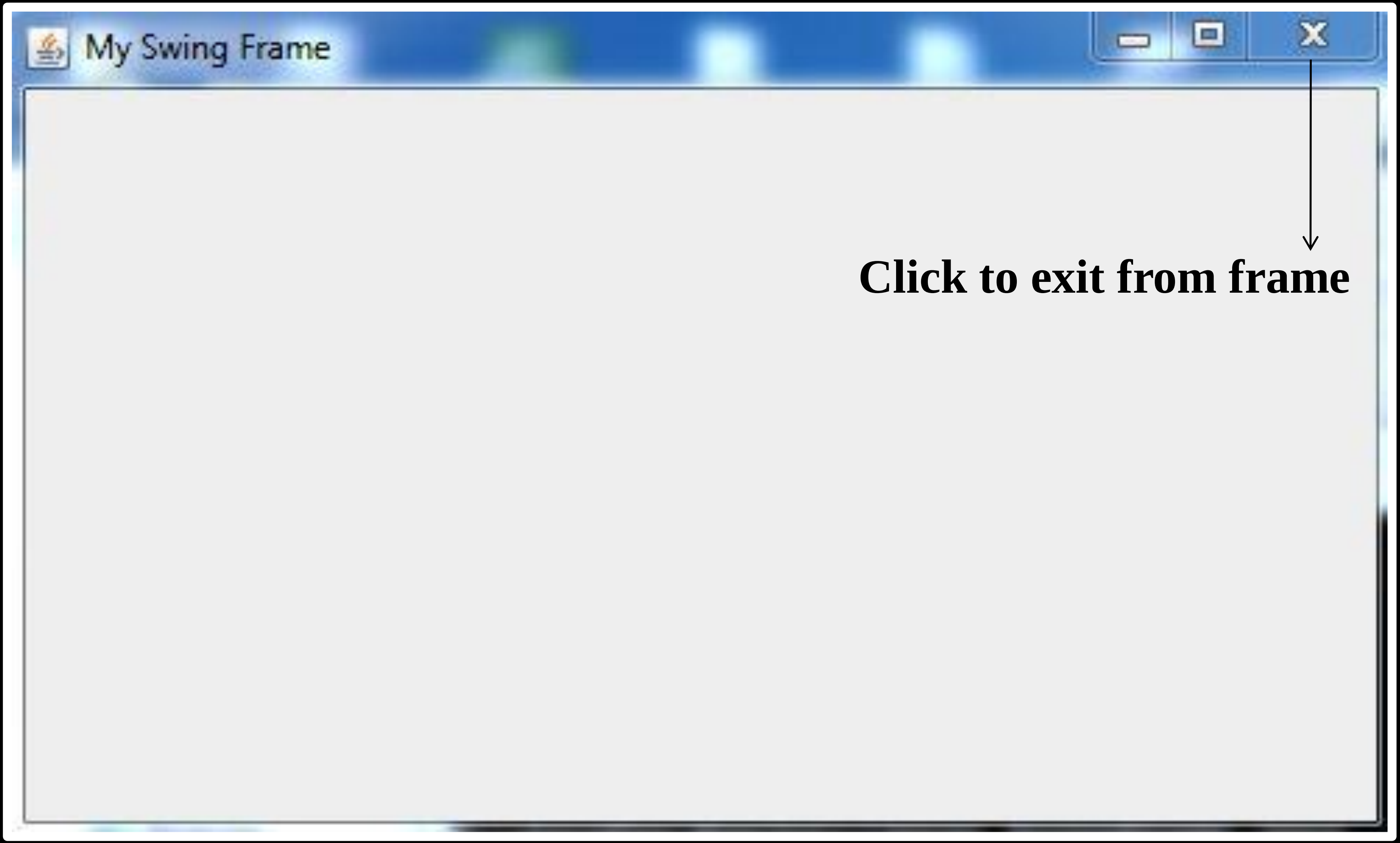
```
// Frame creation using swing
import javax.swing.*;
class SwingFrame
{
public static void main(String args[])
{
    JFrame f = new JFrame("My Swing Frame");
    f.setSize(500, 300);
    f.setVisible(true);
}
}
```

Output



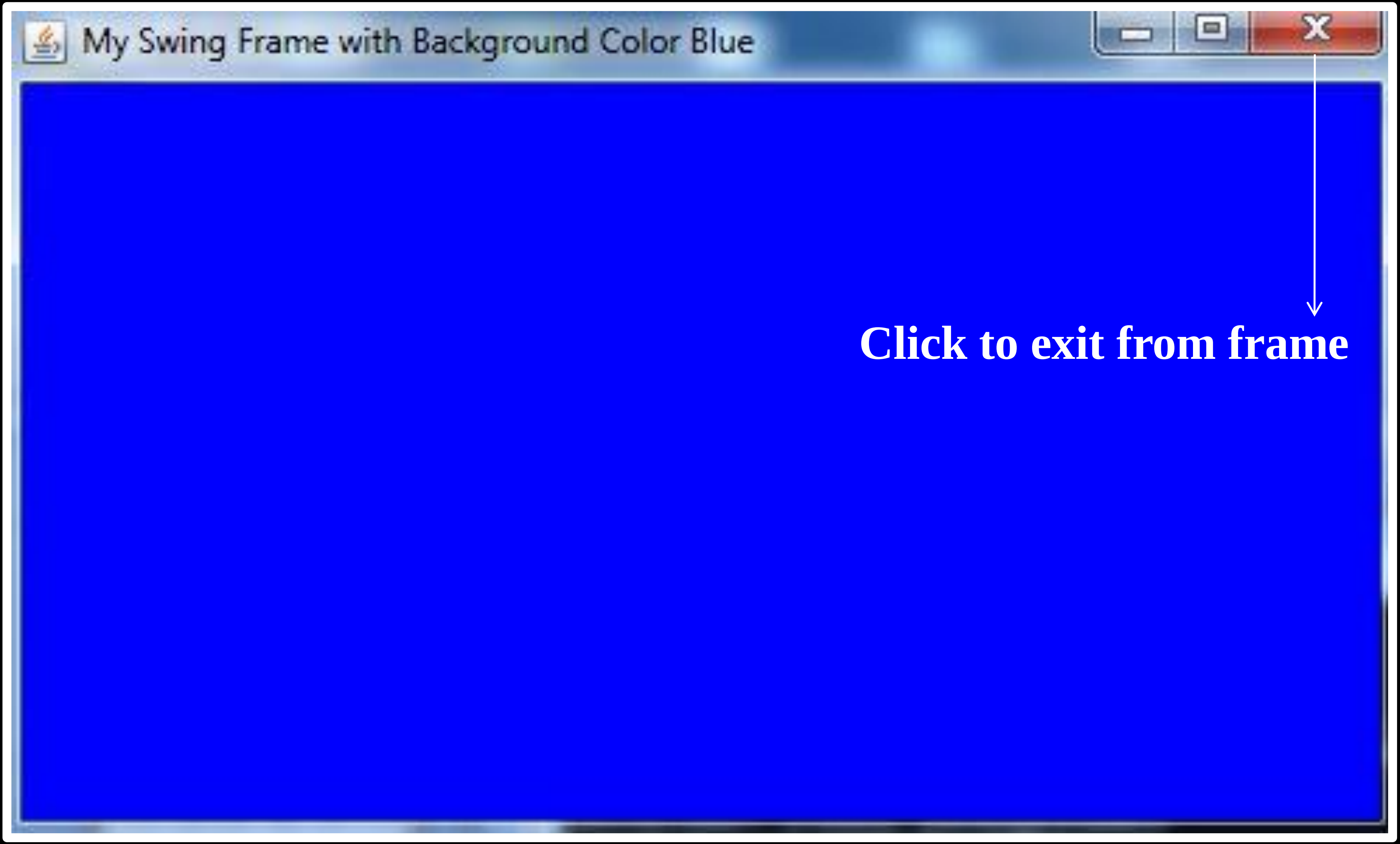
```
// Frame creation using swing and exit from frame by
// clicking  close button
import javax.swing.*;
class SwingFrame1
{
public static void main(String args[])
{
    JFrame f = new JFrame("My Swing Frame");
    f.setSize(500, 300);
    f.setVisible(true);

f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
}
```

Output



```
// Create a Frame and display background colour blue.
import javax.swing.*;
import java.awt.*;   // for Container class
class SwingFrame2 extends JFrame
{
public static void main(String args[])
{
  // create the frame
   SwingFrame2 fobj = new SwingFrame2( );
  // create content pane
   Container cobj = fobj.getContentPane( );
// set green background colour to cobj
    cobj.setBackground(Color.blue);
// set the frame title
   fobj.setTitle("My Swing Frame with Background Color Blue");

    fobj.setSize(500, 300);
    fobj.setVisible(true);

fobj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
;
}
}
```

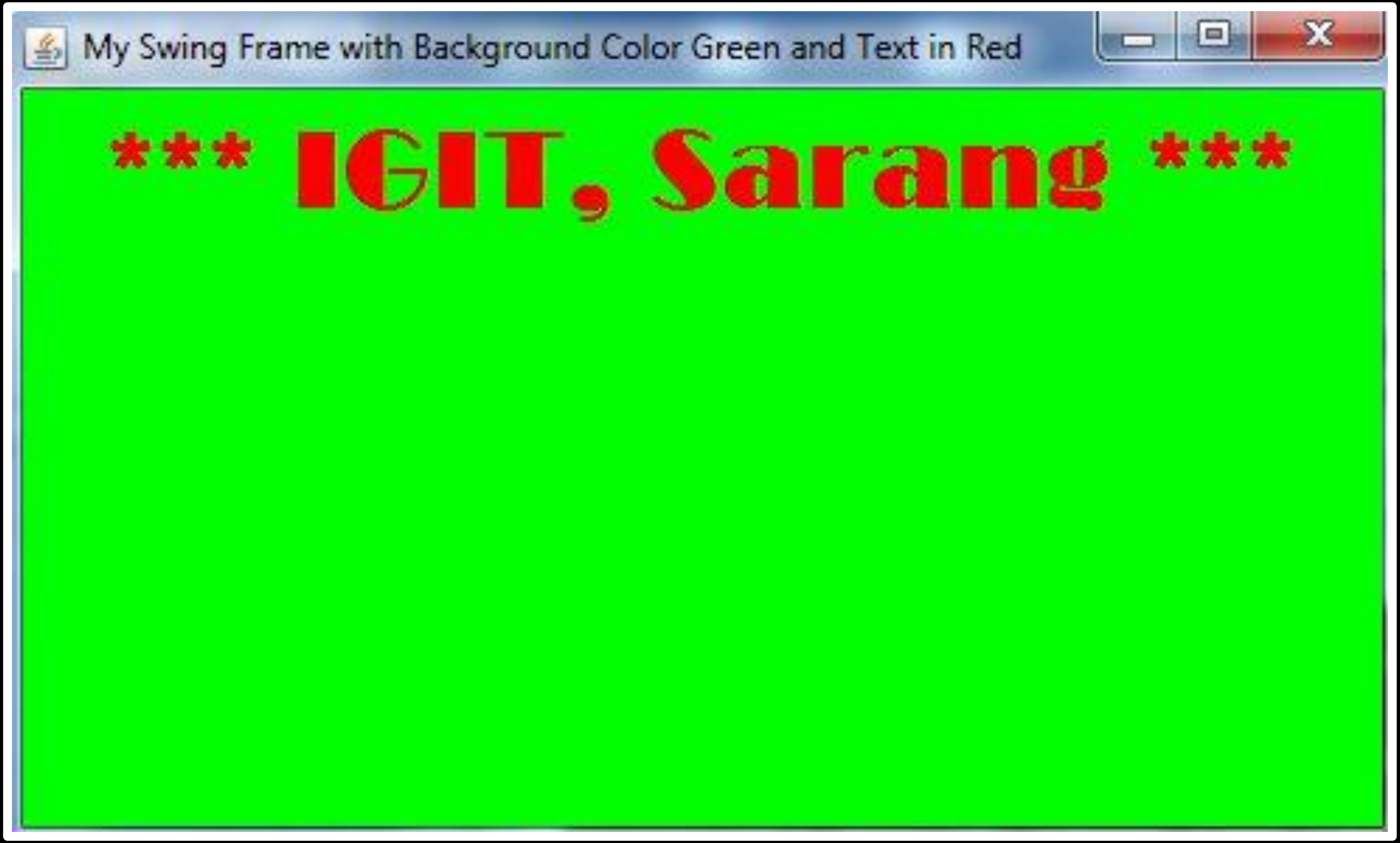
Output


My Swing Frame with Background Color Blue
Click to exit from frame

```
// Create a Frame and display background colour
// green and text red color.
import javax.swing.*;
import java.awt.*;   // for Container class
class SwingFrame3 extends JFrame
{  JLabel jlbl;
   SwingFrame3()
   {
      // create content pane
         Container cobj = this.getContentPane( );
      // set the layout manager
         cobj.setLayout(new FlowLayout( ));
// set green background colour to cobj
   cobj.setBackground(Color.green);

 // create label with text
    jlbl = new JLabel("*** IGIT, Sarang ***");

// font setting to the label
   jlbl.setFont(new Font("Broadway", Font.BOLD, 40));
   jlbl.setForeground(Color.red);
   cobj.add(jlbl);
    }
```

```
public static void main(String args[])
{
// create the frame
    SwingFrame3 fobj = new SwingFrame3();
// set the frame title
fobj.setTitle("My Swing Frame with Background Color Green and
                                                                              Text in Red");

    fobj.setSize(500, 300);
    fobj.setVisible(true);

fobj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
}
```

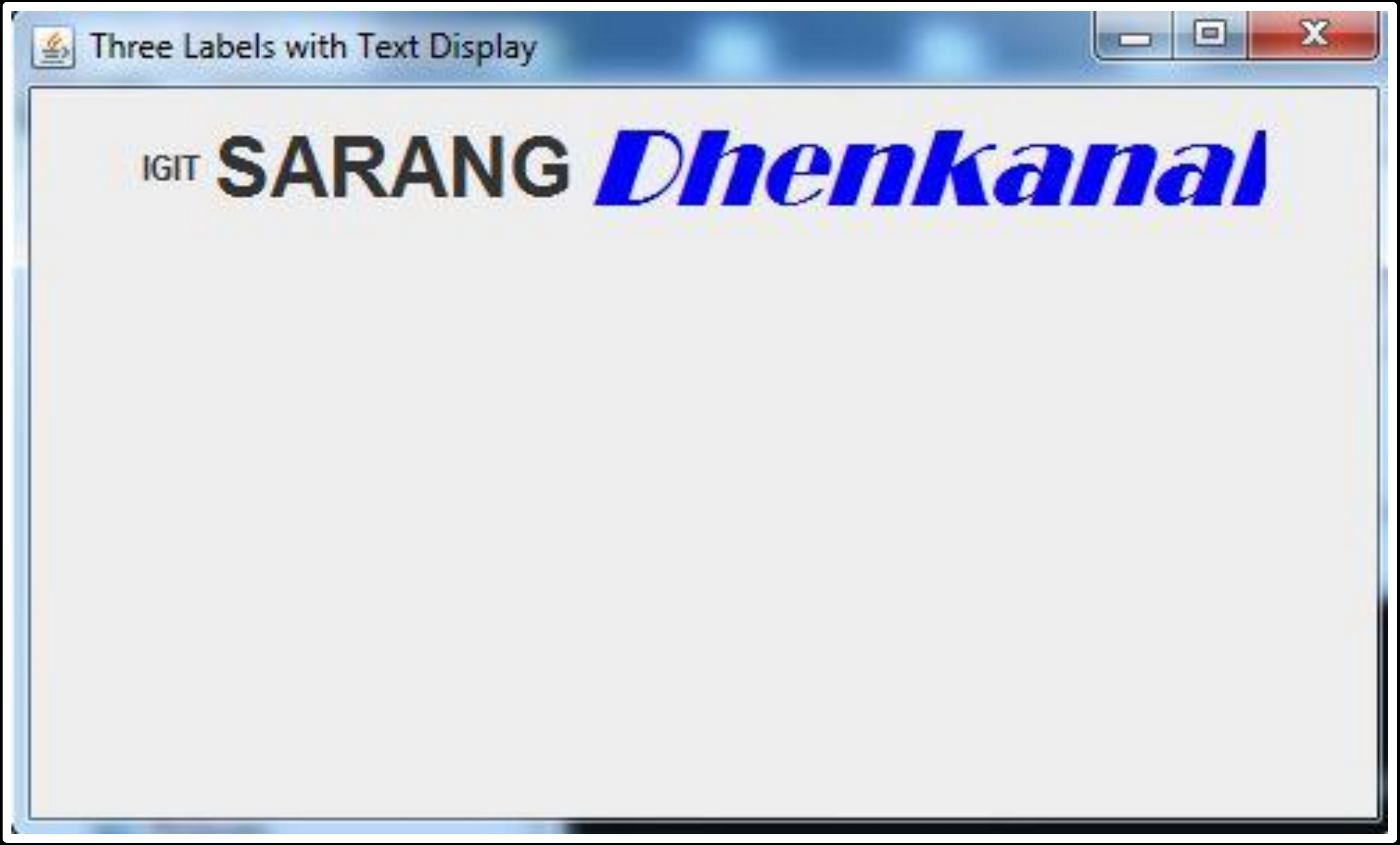
Output



```
My Swing Frame with Background Color Green and Text in Red

*** IGIT, Sarang ***
```

```
// Create three labels with some text.
import javax.swing.*;
import java.awt.*;   // for Container class
class Jlabel extends JFrame
{
    JLabel lab1, lab2, lab3;
    Jlabel()
    {
       Container cobj = this.getContentPane();  // create content pane
       cobj.setLayout(new FlowLayout());  // set the layout manager

       lab1 = new JLabel("IGIT");  // create 1st label with text

       lab2 = new JLabel("SARANG");  // create 2nd label with text

       lab3 = new JLabel("Dhenkanal");  // create 3rd label with text
    // font setting to the label
      lab2.setFont(new Font("Gothic", Font.BOLD, 30));
    // font setting to the label
      lab3.setFont(new Font("Broadway", Font.ITALIC, 40));
    // foreground setting to the label
      lab3.setForeground(Color.blue);
      cobj.add(lab1);        cobj.add(lab2);           cobj.add(lab3);
  }
public static void main(String args[])
{
   Jlabel fobj = new Jlabel();  // create the frame
   fobj.setTitle("Three Labels with Text Display");  // set the frame title
   fobj.setSize(500, 300);
   fobj.setVisible(true);
   fobj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
}
```

Output

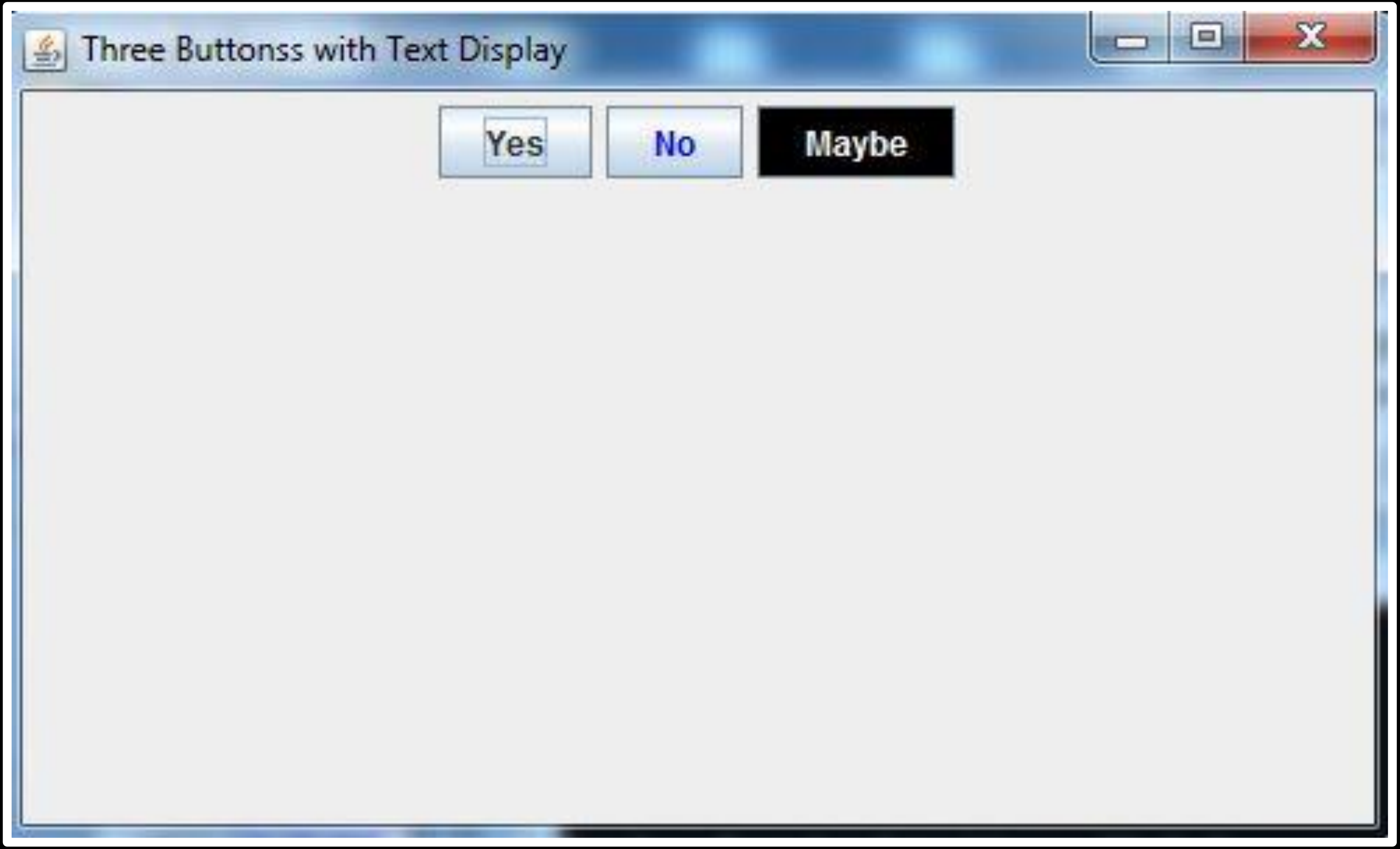

Three Labels with Text Display

IGIT SARANG *Dhenkanal*

```
// Create three buttons with some text.
import javax.swing.*;
import java.awt.*;   // for Container class
class Jbutton extends JFrame
{
    JButton jb1, jb2, jb3;
    Jbutton( )
    {
       Container cobj = this.getContentPane();  // create content pane
       cobj.setLayout(new FlowLayout());  // set the layout manager



       jb1 = new JButton("Yes");  // create 1st button with text Yes
       jb2 = new JButton("No");  // create 2nd button with text No
       jb3 = new JButton("Maybe");  // create 3rd button with Maybe
```

```
      jb2.setForeground(Color.blue);
      jb3.setForeground(Color.white);
      jb3.setBackground(Color.black);
      cobj.add(jb1);
      cobj.add(jb2);
      cobj.add(jb3);
   }
public static void main(String args[])
{
   Jbutton fobj = new Jbutton( );
   fobj.setTitle("Three Buttonss with Text Display");
   fobj.setSize(500, 300);
   fobj.setVisible(true);
   fobj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
}
```

Output

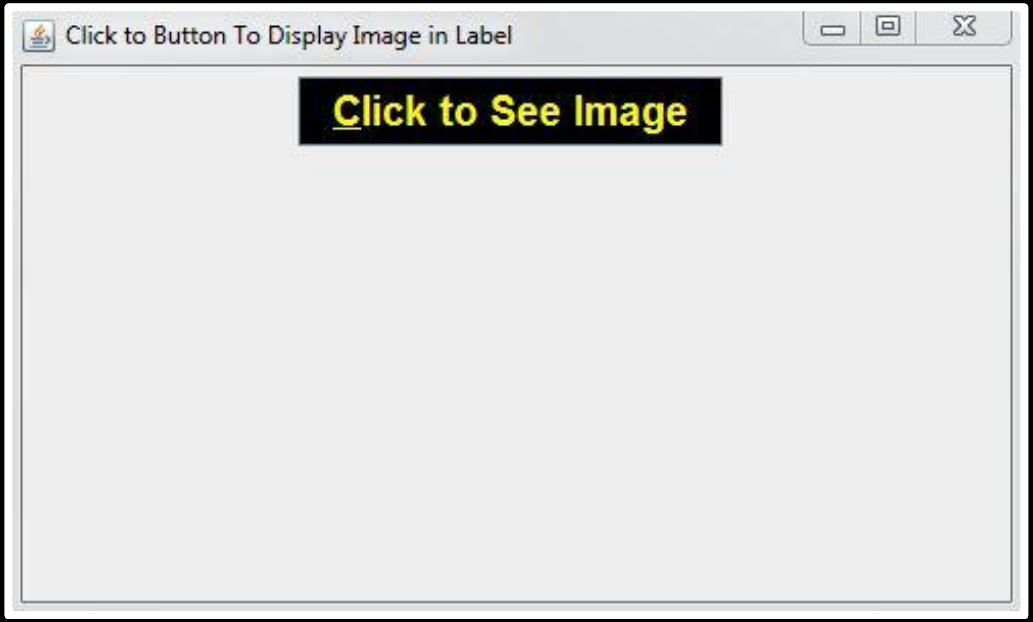


```
// Create a button. Click over it see an image in a label.
import javax.swing.*;
import java.awt.*;          // for Container class
import java.awt.event.*; // for events
class JbuttonClick extends JFrame implements ActionListener
{
   JButton jb;
   JLabel jl;
   JbuttonClick( )
   {
      Container cobj = this.getContentPane( );
      cobj.setLayout(new FlowLayout( ));

      jb = new JButton("Click to See Image");
      jb.setBackground(Color.black);
      jb.setForeground(Color.yellow);
      jb.setFont(new Font("Arial", Font.BOLD, 20));
```

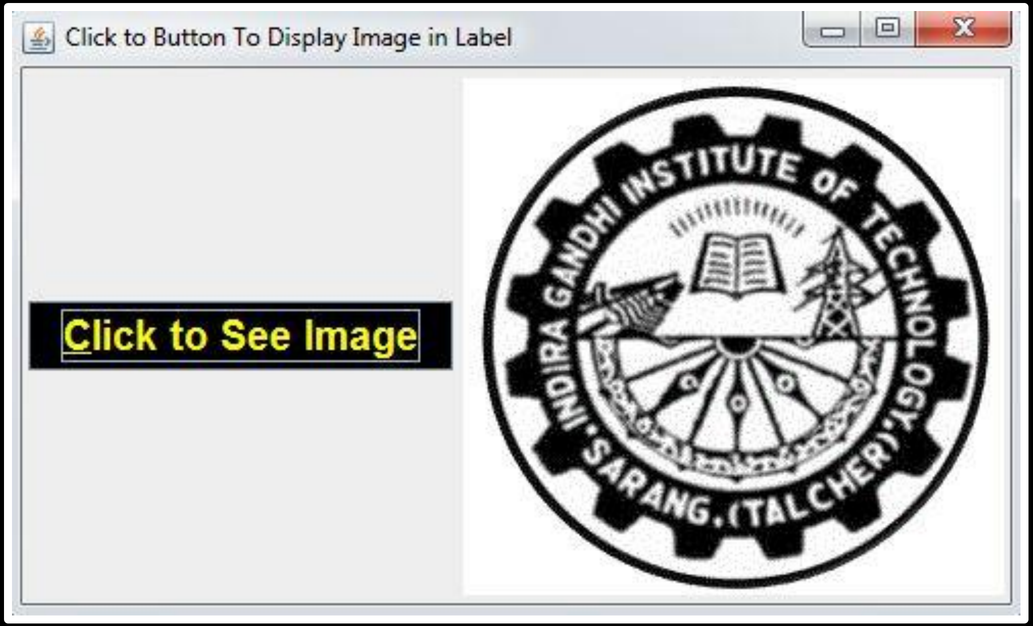
```
        jb.setToolTipText("This is a Button");
        jb.setMnemonic('C');
        cobj.add(jb);
        jb.addActionListener(this);

        jl = new JLabel( );
        cobj.add(jl);
    }
public void actionPerformed(ActionEvent ae)
{
    ImageIcon icon = new ImageIcon("IGIT_LOGO.gif");
    jl.setIcon(icon);
}
```

```
public static void main(String args[])
{
    JbuttonClick fobj = new JbuttonClick( );  // create the frame
    fobj.setTitle("Click to Button To Display Image in Label");
    fobj.setSize(500, 300);
    fobj.setVisible(true);
    fobj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
}
```

Output

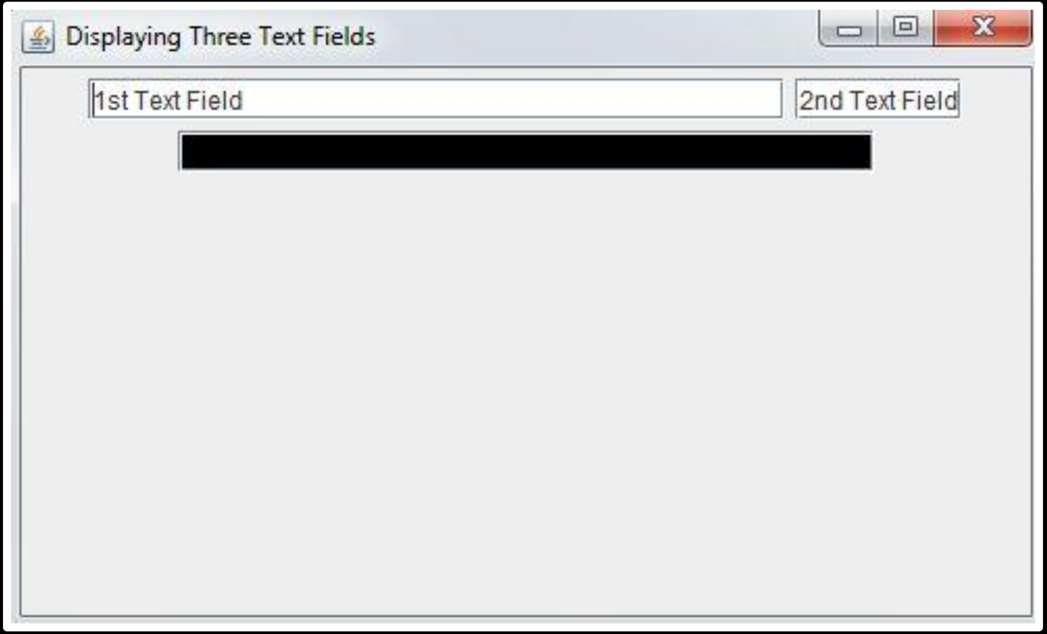


After Click on Button


Click to Button To Display Image in Label
Click to See Image
INDIRA GANDHI INSTITUTE OF TECHNOLOGY, SARANG, (TALCHER)

```
// Create three fields and display it.
import javax.swing.*;
import java.awt.*;   // for Container class
class Jtext extends JFrame
{
   JTextField jt1, jt2, jt3;
   Jtext()
   {
     Container cobj = this.getContentPane();
     cobj.setLayout(new FlowLayout());
     jt1 = new JTextField("1st Text Field", 30);
     jt2 = new JTextField("2nd Text Field");
     jt3 = new JTextField(30);
     jt3.setBackground(Color.black);
     jt3.setForeground(Color.yellow);
     cobj.add(jt1);       cobj.add(jt2);            cobj.add(jt3);
    }
public static void main(String args[])
{
    Jtext fobj = new Jtext();  // create the frame
    fobj.setTitle("Displaying Three Text Fields");  // set the frame title
    fobj.setSize(500, 300);
    fobj.setVisible(true);
    fobj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
}
```

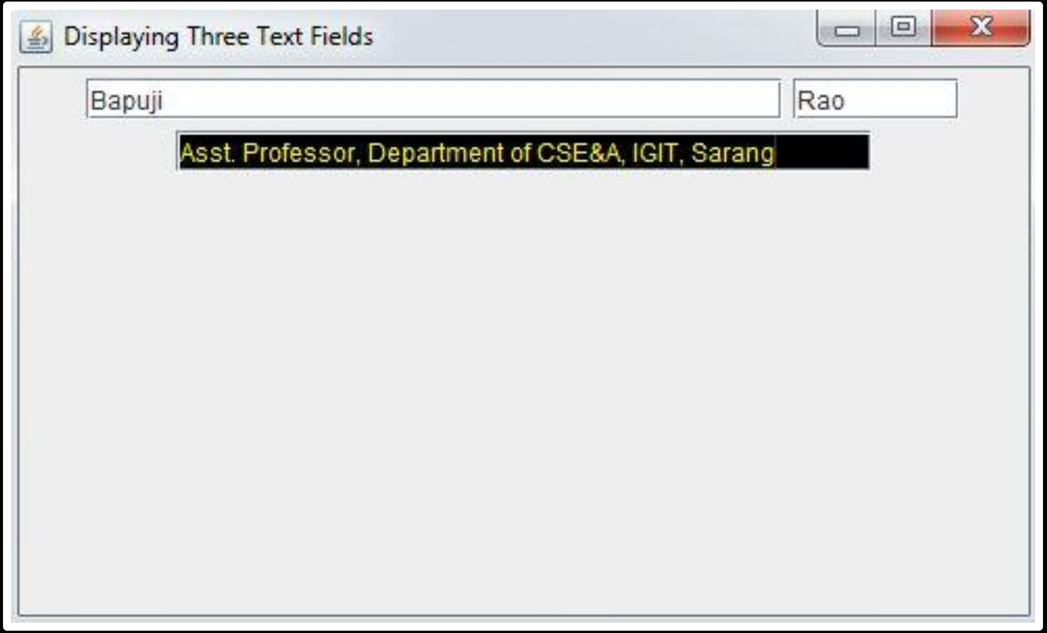
Output


Displaying Three Text Fields

1st Text Field

2nd Text Field

After Input of Text in Text Fields


Displaying Three Text Fields

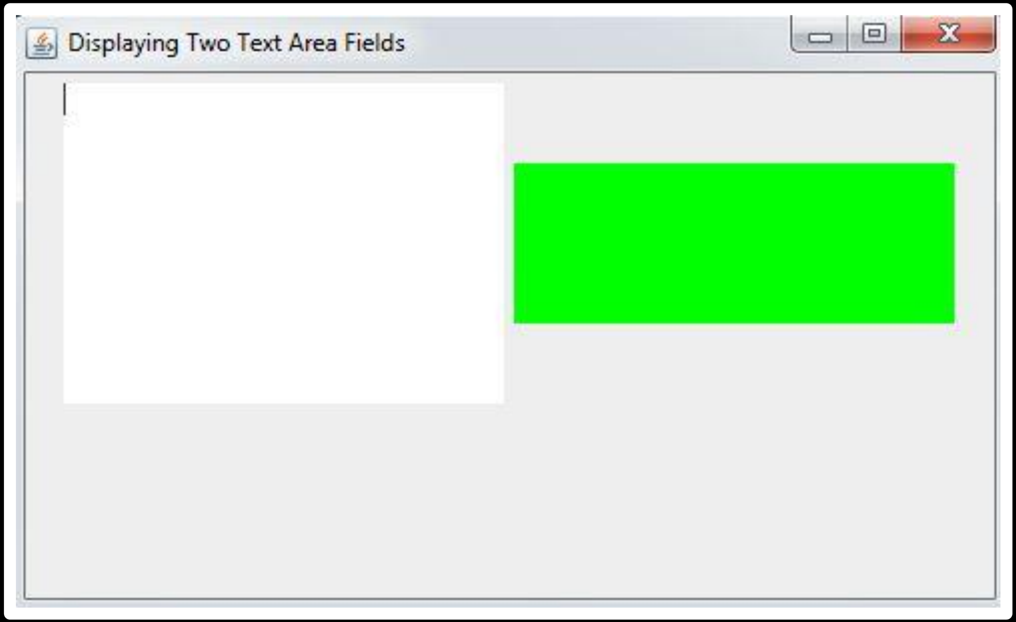
Bapuji

Rao

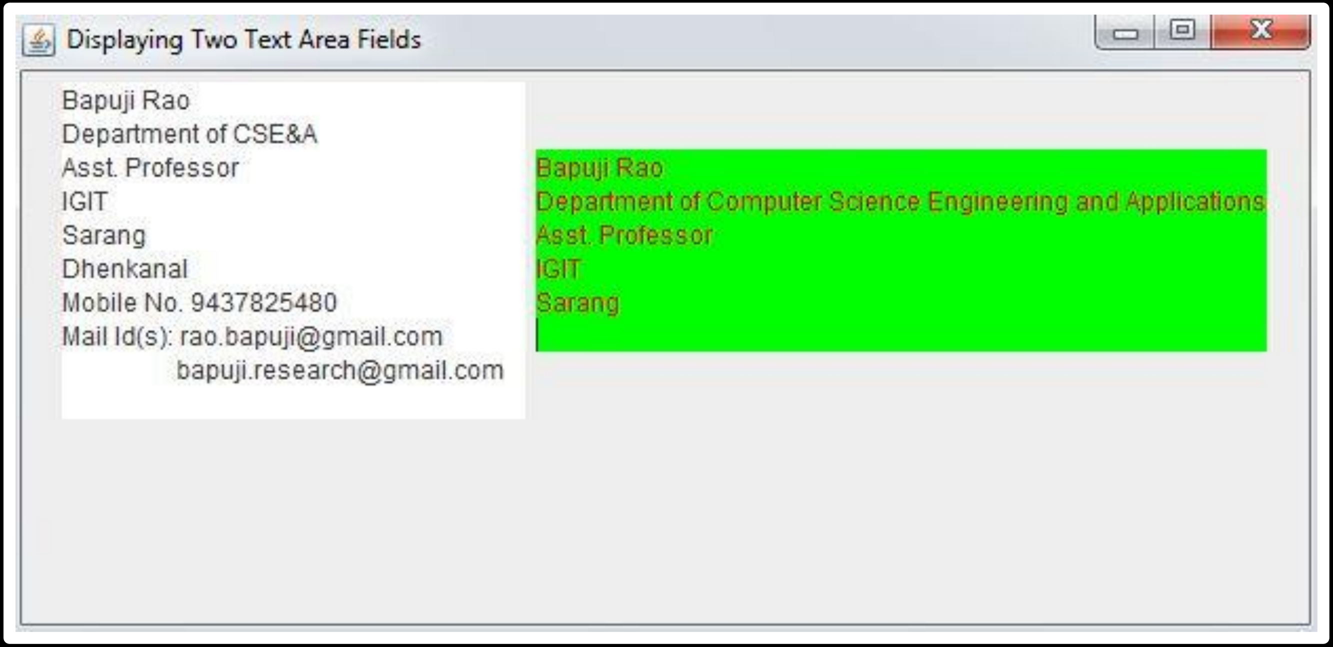
Asst. Professor, Department of CSE&A, IGIT, Sarang

```
// Create two text area fields and display it.
import javax.swing.*;
import java.awt.*;   // for Container class
class Jtextarea extends JFrame
{
    JTextArea jta1, jta2;
    Jtextarea( )
    {
       Container cobj = this.getContentPane( );
       cobj.setLayout(new FlowLayout( ));
       jta1 = new JTextArea(10, 20);
       jta2 = new JTextArea(5, 20);
       jta2.setBackground(Color.green);
       jta2.setForeground(Color.red);
       cobj.add(jta1);
       cobj.add(jta2);
      }
public static void main(String args[])
{
    Jtextarea fobj = new Jtextarea( );  // create the frame
    fobj.setTitle("Displaying Two Text Area Fields");
    fobj.setSize(500, 300);
    fobj.setVisible(true);
    fobj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
}
```

Output


Displaying Two Text Area Fields

After Input of Text in Text Area Fields


Displaying Two Text Area Fields

```
Bapuji Rao
Department of CSE&A
Asst. Professor
IGIT
Sarang
Dhenkanal
Mobile No. 9437825480
Mail Id(s): rao.bapuji@gmail.com
            bapuji.research@gmail.com
```

```
Bapuji Rao
Department of Computer Science Engineering and Applications
Asst. Professor
IGIT
Sarang
```

What is JavaFX?

- JavaFX is a Java library used to build Rich Internet Applications. The applications written using this library can run consistently across multiple platforms.

- The applications developed using JavaFX can run on various devices such as Desktop Computers, Mobile Phones, TVs, Tablets, etc.

Need for JavaFX

- To develop Client Side Applications with rich features, the programmers used to depend on various libraries to add features such as Media, UI controls, Web, 2D and 3D, etc.

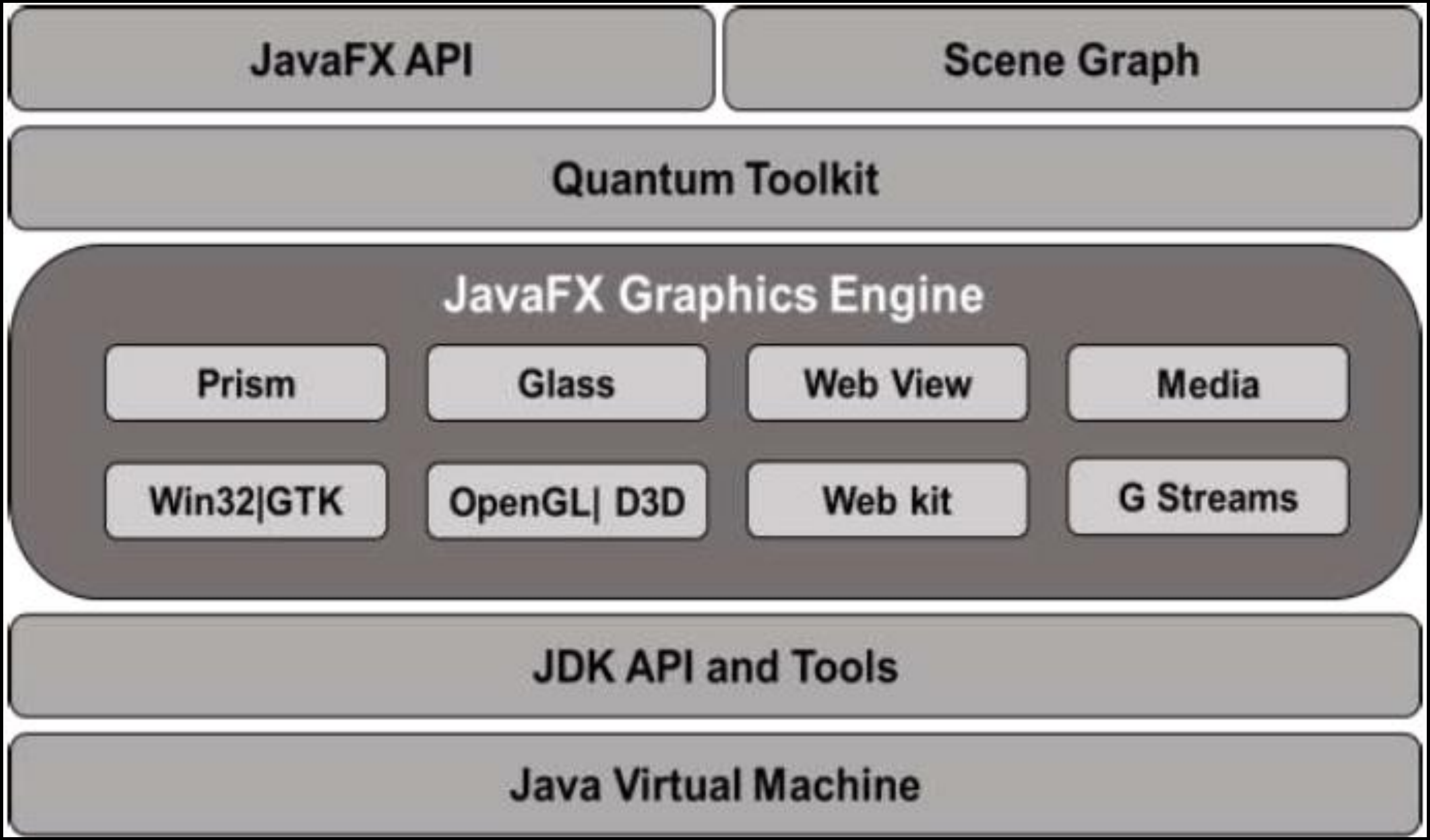
- JavaFX includes all these features in a single library. In addition to these, the developers can also access the existing features of a Java library such as **Swings.**

Features of JavaFX

- Written in Java
- FXML
- Scene Builder
- Swing Interoperability
- Built-in UI controls
- CSS like Styling
- Canvas and Printing API
- Rich set of API’s
- Integrated Graphics library
- Graphics pipeline

JavaFX Environment: From Java8 onwards, the JDK **(Java Development Kit)** includes JavaFX library in it. Therefore, to run JavaFX applications, you simply need to install Java8 or later version in your system. In addition to it, IDE’s like **Eclipse** and **NetBeans** provide support for JavaFX.

Architecture of JavaFX API

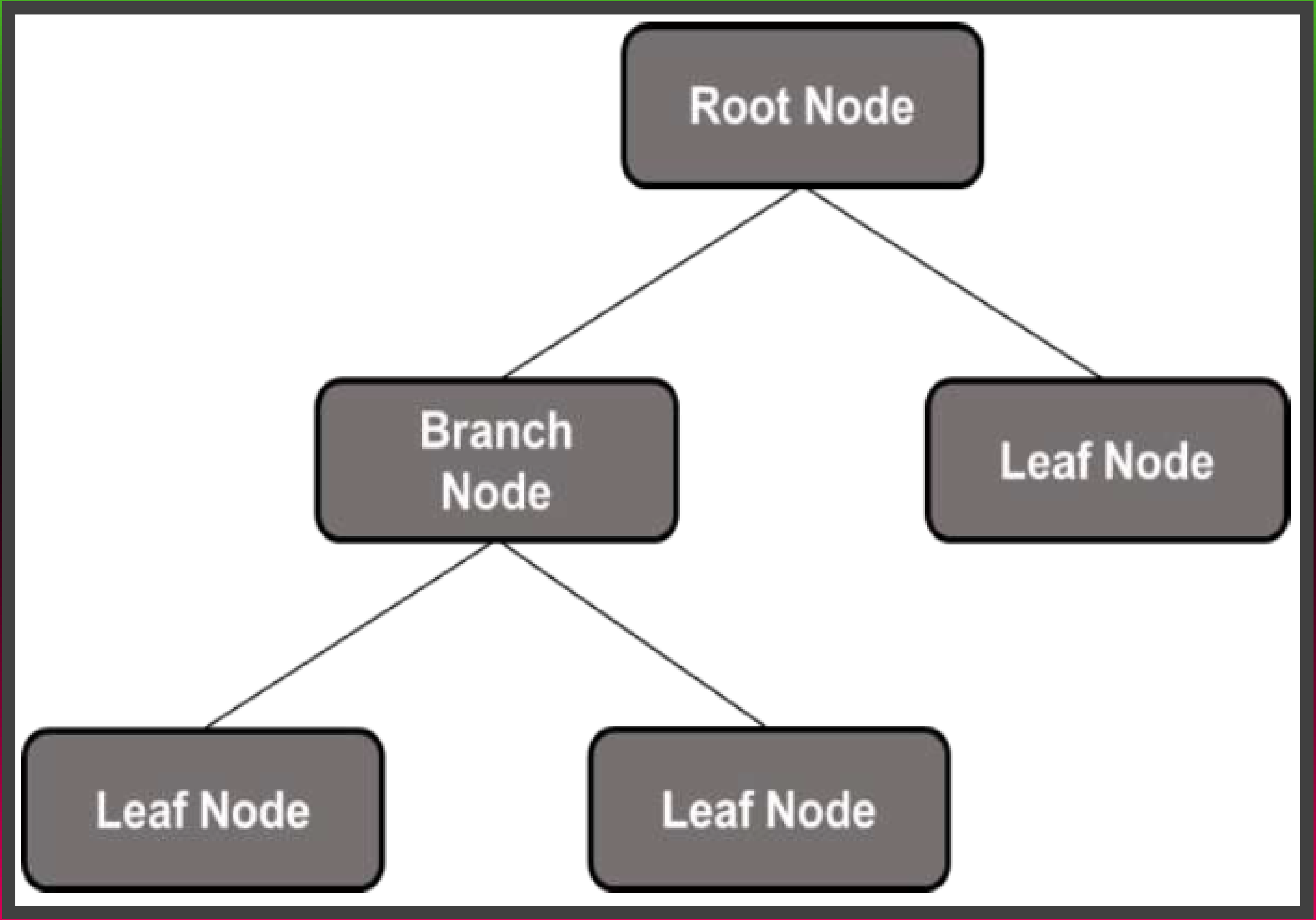


Scene Graph

- In JavaFX, the GUI Applications were coded using a Scene Graph.
- A Scene Graph is the starting point of the construction of the GUI Application.
- It holds the (GUI) application primitives that are termed as nodes.
- A node is a visual/graphical object and it may include:
  - **Geometrical (Graphical) objects:** (2D and 3D) such as circle, rectangle, polygon, etc.
  - **UI controls:** such as Button, Checkbox, Choice box, Text Area, etc.
  - **Containers:** (layout panes) such as Border Pane, Grid Pane, Flow Pane, etc.
  - **Media elements:** such as audio, video and image objects.

- In general, a collection of nodes makes a scene graph. All these nodes are arranged in a hierarchical order as shown below.

Scene Graph



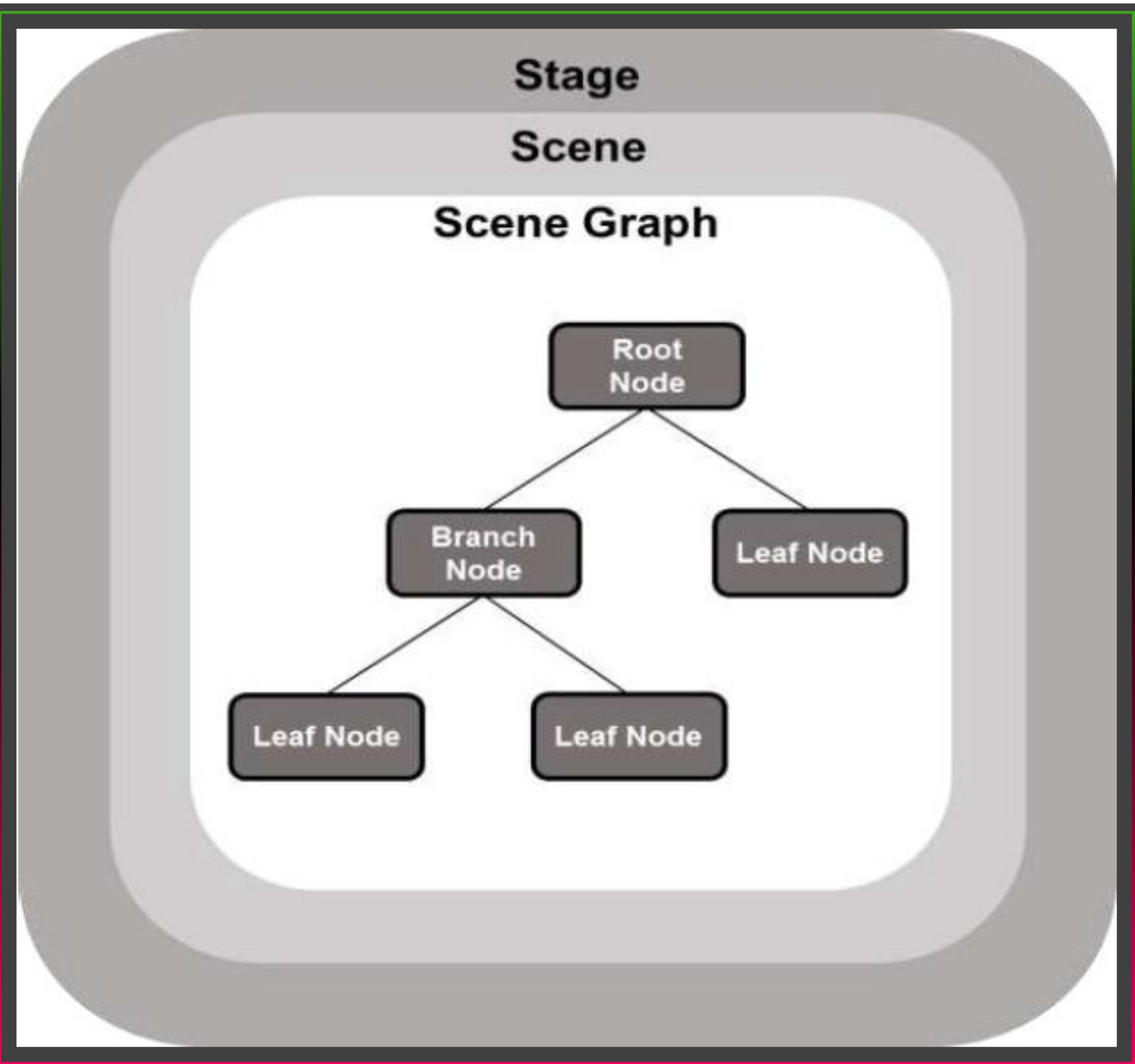
- Each node in the scene graph has a single parent, and the node which does not contain any parents is known as the **root node.**

- In the same way, every node has one or more children, and the node without children is termed as **leaf node; a node with children is termed as a branch node.**

- A node instance can be added to a scene graph only once. The nodes of a scene graph can have Effects, Opacity, Transforms, Event Handlers, Event Handlers, Application Specific States.

JavaFX Application Structure

In general, a JavaFX application will have three major components namely **Stage, Scene** and **Nodes** as shown in the following diagram.



Stage

- A stage (a window) contains all the objects of a JavaFX application. It is represented by **Stage** class of the package **javafx.stage**. The created stage object is passed as an argument to the start() method of the Application class.
- A stage has two parameters determining its position namely **Width and Height. It is** divided as Content Area and Decorations (Title Bar and Borders).
- There are five types of stages available:
    - **Decorated**
    - **Undecorated**
    - **Transparent**
    - **Unified**
    - **Utility**
- You have to call the **show()** method to display the contents of a stage.

Scene

- A scene represents the physical contents of a JavaFX application.
- It contains all the contents of a scene graph.
- The class **Scene** of the package **javafx.scene** represents the scene object.
- At an instance, the scene object is added to only one stage.

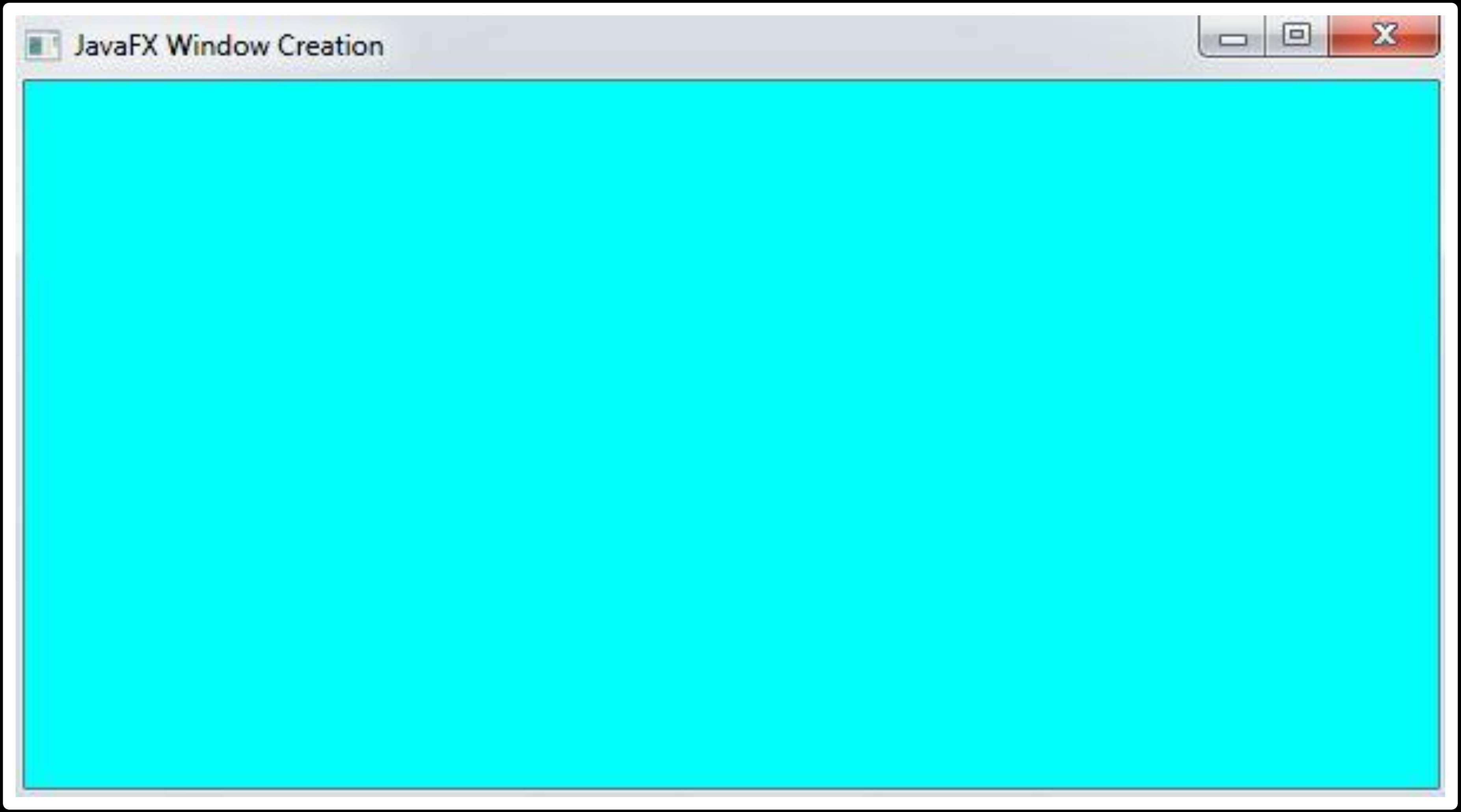
Scene Graph and Nodes

A scene graph is a tree-like data structure (hierarchical) representing the contents of a scene. In contrast, a node is a visual/graphical object of a scene graph.

// Window creation and display using JavaFX

```
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
public class JavafxSample extends Application
{
@Override
public void start(Stage primaryStage) throws Exception
{
//creating a Group object
   Group group = new Group();
//Creating a Scene by passing the group object, height and width
   Scene scene = new Scene(group ,600, 300);
//setting color to the scene
   scene.setFill(Color.CYAN);
//Setting the title to Stage.
   primaryStage.setTitle("JavaFX Window Creation");

//Adding the scene to Stage
   primaryStage.setScene(scene);

//Displaying the contents of the stage
   primaryStage.show();
}
public static void main(String args[])
{
      launch(args);
}
}
```

Output


JavaFX Window Creation