6.1 Applet

Applets are small applications that are accessed on an internet server, transported over the internet, automatically installed & run as part of a web document. An applet is typically embedded inside a web-page and runs in the context of the browser. The applet class provides a standard interface between applets & their environment. An applet is nothing but a subclass of java.applet.Applet. Swing provides a special subclass of Applet, called javax.swing.JApplet, which should be used for all applets that use Swing components to construct their GUIs.

Ex:

```
/* <applet code="sapplet" width=200 height=60></applet> */
import java.awt.*;
import java.applet.*;
public class sapplet extends Applet
{
        public void paint(Graphics g)
        {
                g. drawString("hello",20,20);
        }
}
```

The applet begins with import system. The first statement imports abstract window toolkit (awt) classes. Where the awt contains support for a window based graphical interface. The second import statement imports the applet package, which contains the class Applet. Every applet that you create must be subclass of Applet.

The next line declarethe “sapplet” class where this class must be declared as public, because it will be accessed by code that is outside the program.

Inside “sapplet”, paint() method is declared.this method is defined by awt & overridden by applet. This method has one parameter type graphics, which describe graphic environment, in which applet is running.

Inside paint(), drawString function is call, this method outputs a string beginning at the specified x,y location.

void drawString(String msg, int x, int y)

Applet doesn’t have main method. Its execution begins when name of its class is passed to an applet viewer.

To execute an applet in web browser, you need to write a short html text that contains APPLET tag.

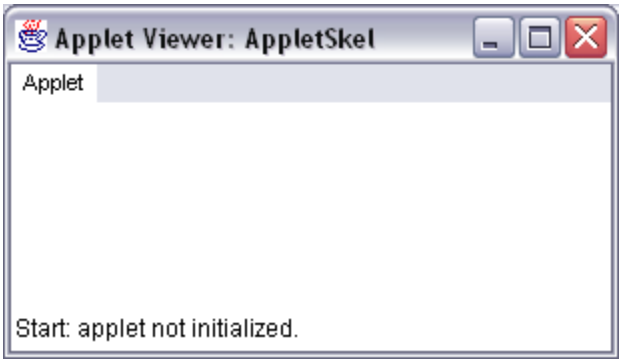
<applet code="sapplet" width=200 height=60></applet>
Where code specifies the name of the file which related to the base URL of the applet code. It is a .class compiled file. Width & height specifies dimension of display area used by an applet. The applet HTML tag embedded in the web page using <applet></applet> tag. "sapplet" compile in the same way that you have been compiling other programs.

Compile: C:\> javac sapplet.java
Execute: C:\>appletviewer sapplet.java

Example of using javax.swing.

```
// An Applet skeleton.
import java.awt.*;
import javax.swing.*;
/*
<applet code="AppletSkel" width=300 height=100>
</applet>
*/

public class AppletSkel extends JApplet {
  // 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
  }
}
```



6.2 Architecture of Applet:

Applet is window based program. As such its architecture is different from normal console based programs.

Applets are event driven program. It waits until an event occurs. The AWT notifies the applet about an event by calling event handler provided by the applet. Once this happens the applet must take appropriate action & then quickly return control to the AWT(runtime system). In those situation where applet need to perform some repetitive task on its own (ex. Displaying scrolling message across its windows) you must start an additional thread of execution.

The user initiates interaction with an applet & not the other way around. In non-window program. When program needs input it will prompt user & then call some input method such as readLine(). But applet takes different approach, Applet handle this with the help of some event/action to which applet must respond. Applet has various event llike mouse-clicked event, key-press event….etc Applet can contain various controls such as push button & check boxes, when user interacts with, one of these control, an event is generated.

Java AWT makes any window based program to response quickly.

6.3 Applet Initialization and Termination(skeleton / Life Cycle):

It is important to understand the order in which the various methods shown in the skeleton are called.

When an applet begins, the AWT calls the following methods, in this sequence:

1. init( ) //defined by applet component class
2. start( ) //defined by applet component class
3. paint( ) //defined by awt component class

When an applet is terminated, the following sequence of method calls takes place:

4. stop( ) //defined by applet component class
5. destroy( ) //defined by applet component class

Let's look more closely at these methods.
init( )
The **init( )** method is the first method to be called. This is where you should initialize variables. This method is called only once during the run time of your applet.

start( )
The **start( )** method is called after **init( )**. It is also called to restart an applet after it has been stopped. Whereas **init( )** is called once—the first time an applet is loaded—**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( )**.

paint( )
The **paint( )** method is called each time your 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 the cause, whenever the applet must redraw its output, **paint( )** 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. This context is used whenever output to the applet is required.

stop( )
The **stop( )** method 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 isprobably running. You should use **stop( )** to suspend threads that don't need to run when the applet is not visible. You can restart them when **start( )** is called if the user returns to the page.

destroy( )
The **destroy( )** method is called when the environment determines that your applet needs to be removed completely from memory. At this point, you should free up any resources the applet may be using (e.g. to kill any threads created in the init ()). The **stop( )** method is always called before **destroy()**.
Example:
Let now write an applet which really does something. Just something, not something useful!

```
/* This Applet sets the foreground and background colors and out puts a string. */
import java.awt.*;
import java.applet.*;
//Also watch command line for messages
/*<applet code="sapplet" width=600 height=50>
  </applet>
*/
public class sapplet extends Applet {
  String msg;
```

```
// set the foreground and background colors.
public void init()
        {
  setBackground(Color.cyan);
  setForeground(Color.red);
  msg = "Initialised --";
        }

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

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

When you compile and run it output of this applet will be something like:

Initialised -- Starting -- Painting.

As you will note the methods stop() and destroy() are not overridden, because they are not needed by this simple applet.

Awt class does have 2 more methods update() and repaint().

6.13. Update and repaint methods of AWT class:

Awt class does have 2 more methods update() and repaint().

6.13.1 update()

It is important in some situations your applet may need to overide the update() method. This method is defined by the AWT and is called when your applet has requested that a portion of its window be redrawn. The problem is that the default version of update() first fills an applet with the default background colour and then calls paint(), this gives rise to a flash of default color (usually gray) each time update is called. To avoid this you define your update() method such that it performs all necessary display activities [NOTE: if you define your own update() method then the default update() is not called - this is called overriding]. The paint() in this casewill simly call update(). Take a look at the following piece of code for better understanding.

```
public void update(Graphic g) {
    //Redisplay your window here.
}

public void paint(Graphics g) {
    update(g) // call to the update()method.
}
```
You need to override update () only when needed.

6.13.2 Requesting Repainting!

An applet writes to its window only when its update() or paint() methods are called by the AWT. You must be wondering how can the applet, itself, cause its window to be updated when its information changes? For example, if an applet is displaying a moving banner, what mechanism does the applet use to update the window each time this banner scrolls? Remember, you cannot create a loop inside paint due the fundamental contraints imposed on an applet - An applet must quickly return control to the AWT run time system. Fortunately the people who designed Java foresaw this predicament and included the repaint() method. Whenever your 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 call to your applet's update() method, which in its default implementation, calls paint(). Again for example if a part of your applet needs to output a string, it can store this string in a variable and then call repaint(). Inside paint(), you can output the string using drawstring().

The repaint method has four forms.

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

Let's look at each one -

void repaint()

This causes the entire window to be repainted

void repaint(int left, int top, int width, int height)

This specifies a region that will be repainted. the integers left, top, width and hieght are in pixels. You save time by specifying a region to repaint instead of the whole window.

void repaint(long maxDelay)

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

Calling repaint() is essentially a request that your applet be repainted sometime soon. However, if your system is slow or busy, update() might not be called immediately. This gives rise to a problem of update() being called sporadically. If your task requires consistent update time, like in animation, then use the above two forms of repaint(). Here, the maxDelay() is the maximum number of milliseconds that can elaspe before update() is called.

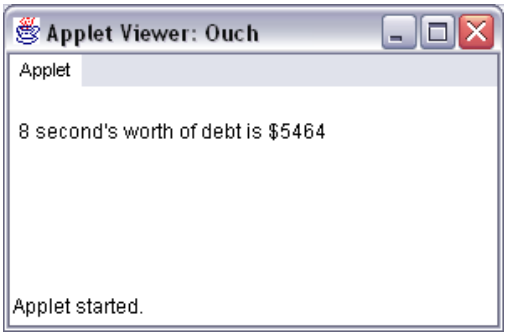
6.13.3 Difference between paint() and repaint()

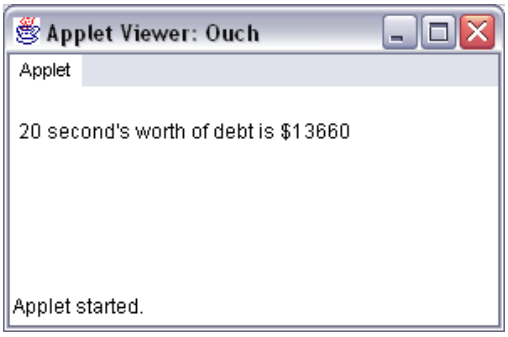
The paint() method contains instructions for painting the specific component. We cannot call this method directly instead we can call **repaint()**.

The repaint() method, which can't be overridden, is more specific: it controlsthe update() to paint() process. You should call this method if you want a componentto repaint itself or to change its look (but not the size)

Example1:

```
import java.awt.*;
import java.util.*;
public class Ex_repaint extends java.applet.Applet
{
    int debt = 683;
    int totalTime = 1;
     public void paint(Graphics g)
         {
         g.drawString(totalTime + " second's worth of debt is $" + (debt * totalTime), 5, 30);
         for (int i = 0; i < 250000000; i++);
         totalTime++;
         repaint();//calls paint() again
         }
 }
```





6.14 The HTML APPLET Tag

The APPLET tag is used to start an applet from both an HTML document and from an applet viewer. An applet viewer will execute each APPLET tag that it finds in a separate window,
While web browsers like Netscape Navigator, Internet Explorer, and HotJava will allow many applets on a single page.

The syntax for the standard 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>

Let's take a look at each part now.
CODEBASE: CODEBASE is an optional attribute that specifies the base URL of the applet code, which is the directory that will be searched for the applet's executable class file (specified by the CODE tag). The HTML document's URL directory is used as the CODEBASE if this attribute is not specified. The CODEBASE does not have to be on the host from which the HTML document was read.

CODE: CODE is a required attribute that gives the name of the file containing your applet's compiled **.class** file. This file is relative to the code base URL of the applet, which is the directory that the HTML file was in or the directory indicated by CODEBASE if set.

ALT: The ALT tag is an optional attribute used to specify a short text message that should be displayed if the browser understands the APPLET tag but can't currently run Java applets. This is distinct from the alternate HTML you provide for browsers that don't support applets.

WIDTH AND HEIGHT: WIDTH and HEIGHT are required attributes that give the size (in pixels) of the applet display area.

ALIGN: ALIGN is an optional attribute that specifies the alignment of the applet. This attribute is treated the same as the HTML IMG tag with these possible values: LEFT, RIGHT, TOP, BOTTOM, MIDDLE, BASELINE, TEXTTOP, ABSMIDDLE, and ABSBOTTOM.

VSPACE AND HSPACE: These attributes are optional. VSPACE specifies the space, in pixels, above and below the applet. HSPACE specifies the space, in pixels, on each side of the applet. They're treated the same as the IMG tag's VSPACE and HSPACE attributes.

PARAM NAME AND VALUE: The PARAM tag allows you to specify appletspecific arguments in an HTML page. Applets access their attributes with the **getParameter( )** method.

6.15 Passing parameters to Applets

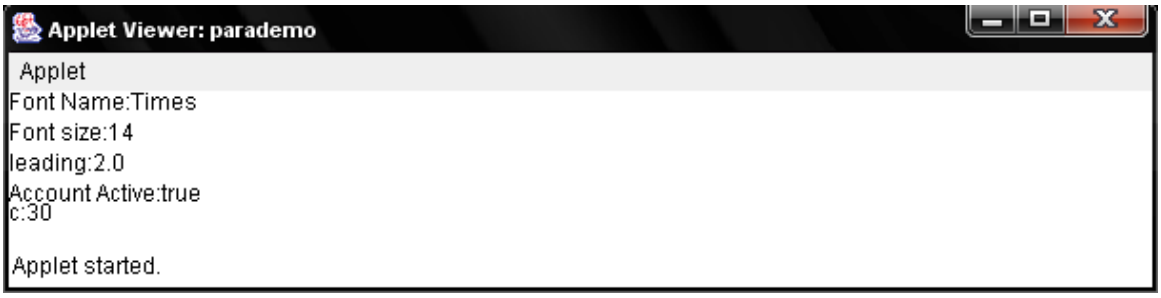
Applet html tag allows you to pass parameters to your applet. To retrieve a parameter, use the getParameter() method. It returns the value of the specified parameter in the form of a String object.

Example:

```
import java.awt.*;
import java.applet.*;
/*<applet code = "parademo" width = 600 height=80>
<param name=fname value=Times New Roman>
<param name=fsize value=14>
<param name=lead value=2>
<param name=accountEnabled value=true>
</applet>*/

public class parademo extends Applet
{
String fname;
int  fsize;
float lead;
boolean active;
int a,b,c;
        public void start()
        {
        String param;

        fname=getParameter("fname");
        if(fname==null)
        fname="not found";

                param = getParameter("fsize");
                try
```

```
		{
			if(param!=null)
			fsize=Integer.parseInt(param);
			else
			fsize=0;
		}
		catch(Exception e)
		{
		fsize= -1;
		}

		param=getParameter("lead");
		try
		{
			if(param != null)
			lead=Float.valueOf(param).floatValue();
			else
			lead=0;
		}
		catch(Exception e)
		{
		lead= -1;
		}
	param=getParameter("accountEnabled");
	if(param!=null)
	active=Boolean.valueOf(param).booleanValue();

	a=10;
	b=20;
	c=a+b;

	}//start

public void paint(Graphics g)
{
	g.drawString("Font Name:"+fname,0,10);
	g.drawString("Font size:"+fsize,0,26);
	g.drawString("leading:"+lead,0,42);
	g.drawString("Account Active:"+active,0,58);
	g.drawString("c:"+c,0,68);
}//paint

}//parademo
```

In given applet program, getParameter method returns values in String object so for numeric, float & boolean values , we convert their string representation into their internal formats.



6.16 Applet Methods

There are different types of methods for the Graphics class of the *java.awt.*;* package has been used to draw the appropriate shape.

1) setBackground
public void setBackground(Color c)
Sets the background color of this component.
The setBackground() method is defined by the Component class. It is used to set the background of an applet's window to a specified color. Here, new color specifies the color. The Color class defines the constants, which can be used to set the color. Some of them are listed below:
Color.black(pink,green,yellow,blue,magenta,orange,red,white,lightGray,darkGray,gray,cyan)
The background color affects each component differently and the parts of the component that are affected by the background color may differ between operating systems.
Parameters:
c - the color to become this component's color; if this parameter is null, then this component will inherit the background color of its parent

2)setForeground
public void setForeground(Color c)
Sets the foreground color of this component.
Parameters:
c - the color to become this component's foreground color; if this parameter is null then this component will inherit the foreground color of its parent

3)getBackground
public Color **getBackground**()
Gets the background color of this component.
Returns: this component's background color; if this component does not have a background color, the background color of its parent is returned

4)getForeground

public Color getForeground()

Gets the foreground color of this component.

Returns: this component's foreground color; if this component does not have a foreground color, the foreground color of its parent is returned

5)setColor

This is the setColor() method which is the Graphics class method imported by the *java.awt.*;* package. This method sets the color for the object by specified color. Here is the syntax of the setColor() method :

g.setColor(**Color.**color_name);

6)drawString

The drawSring() method draws the given string as the parameter on particular position using graphical interface.

void drawString(String string, int X_coordinate, int Y_coordinate);

Text is displayed in an applet window by using the drawString() method of the Graphics class.The drawString() method is similar in function to the System.out.println() method that displays information to the system's standard output device. Before you can use the drawString() method, you must have a Graphics object that represents the applet window.

The paint() method of all applets includes a Graphics object as its only argument. This object represents the applet window, so it can be used to create a Graphics object that also represents the window.

The following three arguments are sent to drawString():

- The text to display, which can be several different strings and variables pasted together with the + operator
- The x position (in an (x,y) coordinate system) where the string should be displayed
- The y position where the string should be displayed

Ex: g.drawString(“hello”,10,10);

7)drawLine

The drawLine() method has been used in the program to draw the line in the applet.

Here is the syntax for the drawLine() method :

drawLine(int X_from_coordinate, int Y_from_coordinate, int X_to_coordinate, int Y_to_coordinate);

Ex: g.drawLine(3,300,200,10);

8)drawOval

The drawOval() method draws the circle. Here is the syntax of the drawOval() method :

g.drawOval(int X_coordinate, int Y_coordinate, int Wdth, int height);

Ex: g.drawOval(10,10,50,50);

9)drawRect

The drawRect() method draws the rectangle. Here is the syntax of thedrawRect() method :

g.drawRect(int X_coordinate, int Y_coordinate, int Wdth, int height)

Ex: g.drawRect(10,10,50,50);

10)fillOval

This is the fillOval() method used to fill the color inside the oval by specified color. Here is the syntax of the fillOval() method :

g.fillOval(int X_coordinate, int Y_coordinate, int Wdth, int height);

Ex: g.drawOval(10,10,50,50);

11)fillRect

This is the fillRect() method used to fill the color inside the rectangle by specified color. Here is the syntax of the fillRect() method :

g.fillRect(int X_coordinate, int Y_coordinate, int Wdth, int height)

Ex: g.fillRect(10,10,50,50);

12)drawRoundRect & fillRoundRect

It draws an outlined round-cornered rectangle. Here are the syntaxies and examples of both the functions.

Syntax: drawRoundRect(int x,int y,int width,int height,int arcWidth,int arcHeight)

Ex: g.drawRoundRect(200,300,100,100,50,50);

Syntax: fillRoundRect(int x,int y,int width,int height,int arcWidth,int arcHeight)

Ex: g.fillRoundRect(200,300,100,100,50,50);

Example 1:

```
import java.applet.*;
import java.awt.*;
public class ShapColor extends Applet{
  int x=300,y=100,r=50;
  public void paint(Graphics g){
  g.setColor(Color.red); //Drawing line color is red
  g.drawLine(3,300,200,10);
  g.setColor(Color.magenta);
  g.drawString("Line",100,100);
  g.drawOval(x-r,y-r,100,100);
  g.setColor(Color.yellow); //Fill the yellow color in circle
  g.fillOval( x-r,y-r, 100, 100 );
  g.setColor(Color.magenta);
  g.drawString("Circle",275,100);
  g.drawRect(400,50,200,100);
  g.setColor(Color.yellow); //Fill the yellow color in rectangel
  g.fillRect( 400, 50, 200, 100 );
  g.setColor(Color.magenta);
```

```
  g.drawString("Rectangel",450,100);
  }
}
```

Example 2:

```
/*<applet code =shapes width=500 height=500></applet>*/
import java.applet.*;
import java.awt.*;
public class shapes extends Applet{
  int x=300,y=100;
  public void paint(Graphics g){
        g.drawLine(3,300,200,10);
        g.drawString("Line",100,100);

        g.drawOval(x,y,100,100);
        g.setColor(Color.cyan);
        g.fillOval(x+1,y+1,99,99);
        g.setColor(Color.black);
        g.drawString("Circle",325,150);

        g.drawRect(400,50,100,100);
        g.setColor(Color.red);
        g.fillRect(401,51,99,99);
        g.setColor(Color.black);
        g.drawString("Rectangel",420,100);

        g.fillRoundRect(200,300,100,100,50,50);
        g.setColor(Color.blue);
        g.drawRoundRect(200,300,100,100,50,50);
        g.drawString("RoundRect",220,350);
  }
}
```

13)Drawing Polygons

Polygons are shapes with many sides. A polygons may be defined as a set of connected lines. The end of first line is the beginning of second line, and so on,

Syntax:

drawPolygon(int[] xPoints, int[] yPoints, int nPoints)

Draws a closed polygon defined by arrays of x and y coordinates.

Example:

```
import java.awt.*;
import java.applet.*;
public class Poly extends Applet
{
```

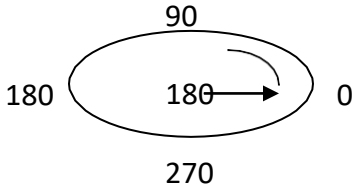
```
int x1[]={20,120,220,20};
int y1[]={20,120,20,20};
int n1=4;
int x2[]={120,220,220,120};
int y2[]={120,20,220,120};
int n2=4;
public void paint(Graphics g)
{
g.drawPolygon(x1,y1,n1);
g.fillPolygon(x2,y2,n2);
}
}
```

14)Drawing Arc

drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)

Draws the outline of a circular or elliptical arc covering the specified rectangle.

Java considers 3 O'clock as 0 degree position and degree increases in **anti-clock** wise direction.



Example:

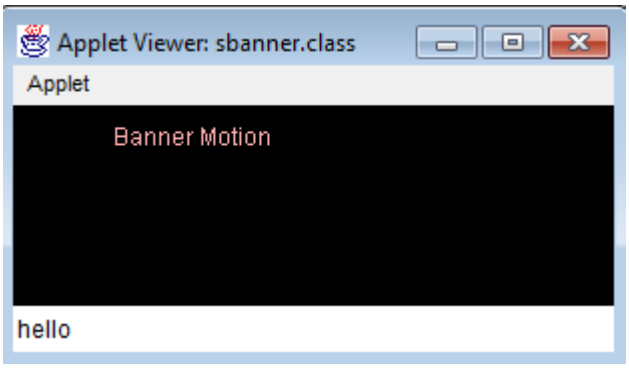
```
import java.awt.*;
import java.applet.*;
public class Face extends Applet
{
public void paint(Graphics g)
{
g.drawOval(40,40,120,150); //Head
g.drawOval(57,75,30,20); //Left eye
g.drawOval(110,75,30,20); //Right eye
g.fillOval(68,81,10,10); //Pupil (left)
g.fillOval(121,81,10,10); //Pupil (right)
g.drawOval(85,100,30,30); //Nose
g.fillArc(60,125,80,40,180,180); //Mouth
g.drawOval(25,92,15,30); //Left ear
g.drawOval(160,92,15,30); //Right ear
}
}
```

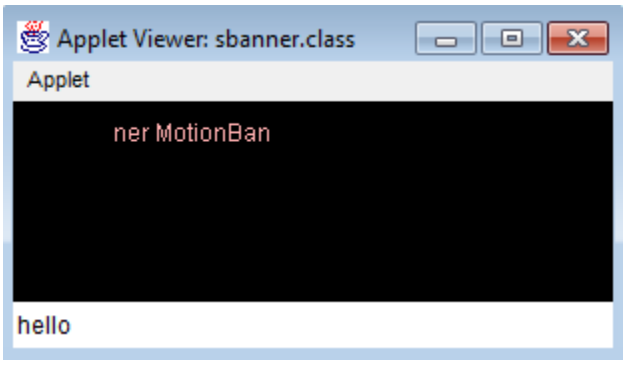
Program: Message movement program at fix location using repaint and thread.

```
import java.awt.*;
import java.io.*;
```

```
import java.applet.*;
import java.lang.Thread;

/* <applet code = "sbanner.class" width = 300 height=100></applet> */

public class sbanner extends Applet implements Runnable{
String msg= "Banner Motion";
Thread t = null;
int state;
boolean stopF;

public void init()
{
setBackground(Color.black);
setForeground(Color.pink);
stopF = false;
}//init

public void start()
{
t=new Thread(this);
t.start();
}//start

public void run()
{
char c;
for(; ;)
{
try
{
repaint();
Thread.sleep(500);
c=msg.charAt(0);
msg=msg.substring(1,msg.length());
msg+=c;
//System.out.println(msg);
if(stopF)
break;
}
catch(InterruptedException e)
{}
}//for
}//run
```

```
public void stop()
{
stopF=true;
t=null;
}//stop

public void paint(Graphics g)
{
g.drawString(msg,50,20);
showStatus("hello");
}//paint
}//sbanner
```



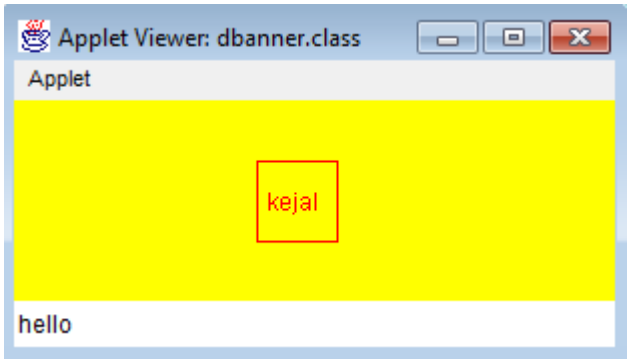


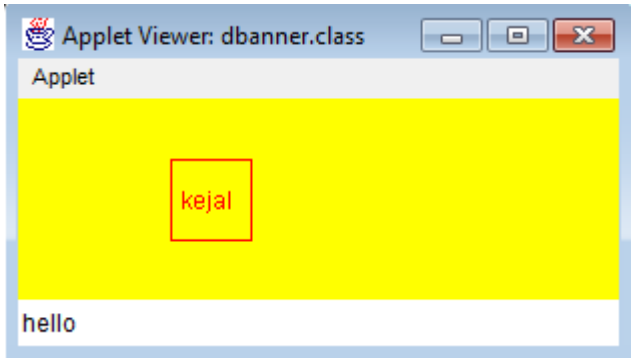
Program: Movement of Object with a Message towards left hand Side.

```
import java.awt.*;
import java.io.*;
import java.applet.*;
import java.lang.Thread;

/* <applet code = " dbanner.class" width = 300 height=100></applet> */

public class dbanner extends Applet implements Runnable{
Thread t = null;
```

```
boolean stopF;
int width, height;
int x, y;
public void init()
{
    width = getSize().width;
    height = getSize().height;
    setBackground( Color.yellow);
    x = width/2 - 20;
    y = height/2 - 20;
    stopF = false;
}//init

public void start()
{
t=new Thread(this);
t.start();
}//start

public void run()
{
for(; ;)
{
try
{
Thread.sleep(250);
repaint();
x=x-1;
if(stopF)
break;
}
catch(InterruptedException e)
{}
}//for
}//run

public void paint(Graphics g)
{
setForeground( Color.red );
g.drawRect( x, y, 40, 40 );
g.drawString("kejal",x+5,y+25);
showStatus("hello");
}//paint
```

```
public void stop()
{
s
t
o
p
F
=
t
r
u
e
;
t
=
n
u
l
l
;
}//stop
}//class
```





Ś