




APPLET and GRAPHICS PROGRAMMING

Dr. S.V Shirbhate
Asst.professor
DCPE, HVPM, Amravati

Content's

- 
- Introduction to Applet
 - The Applet Class
 - Applet Life Cycle
 - Applet tag
 - Passing Parameters to Applets
 - Displaying Text in Applet Window

Working with Graphics:

- Drawing Lines, Rectangles, Polygon, Ellipse & Circles, Arcs
- Working with Color

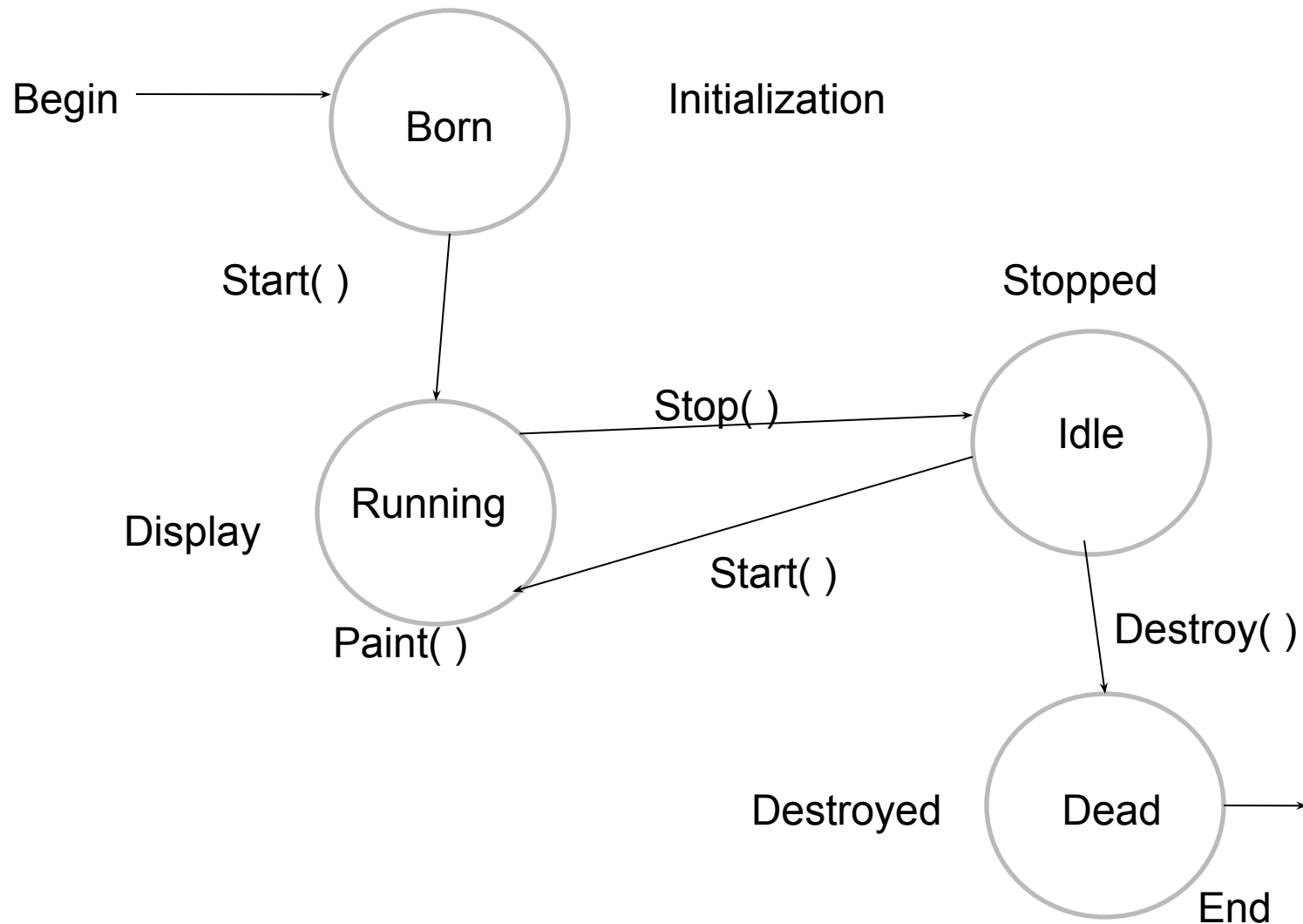
Introduction



Applet are small Java program that are primarily used in Internet computing and runs under Java enabled web browser. Applet can perform arithmetic operations, display graphics, play sounds, create animation and play interactive games.

An Applet when run can produce graphics, sounds and moving images. Thus, Java Applet have begun to make a significant impact on World Wide Web.

Applet Life Cycle



Applet Life Cycle

- **'Born state'** is used for Initialization of Applet. For that it uses `init()` method of Applet class.
- **'Running state'** is used for start or run the Applet. For that it uses `start()` method of Applet class.
- **'Idle or Stopped state'** is used to stop the Applet from the Running state. For that it uses `stop()` method of Applet class.
- An Applet is said to be **'Dead'** when it is removed from the memory. This occurs automatically by invoking the `destroy()` method.
- For performing output operation on the screen Applet require **'Display state'** which uses `paint()` method for this purpose.

Applet Tag

Like HTML Applet also includes a pair of `<Applet>` and `</Applet>` in its body section. The `<Applet>` tag supplies the name of applet to be loaded and tells the browser how much space the applet requires. The `<Applet>` tag indicates that it contains certain attributes that must be specified.

The `<Applet>` tag given below specifies the minimum requirements to place the HelloJava applet on web.

```
<APPLET
```

```
    CODE= HelloJava.class
```

```
    WIDTH= 400
```

```
    HEIGHT= 200 >
```

```
</APPLET>
```

Applet Tag

This HTML code tells the browser to load the compiled Java applet HelloJava.class, which is in the same directory with this HTML file. And also specifies the display area for the applet output as 400 pixel width and 200 pixel height.

Note that, <APPLET> tag discussed above specifies three things.

1. Name of the Applet
2. Width of the Applet(in pixels)
3. Height of the Applet(in pixels)

Passing Parameters To Applet

We can pass user-defined parameters to an applet using `<PARAM..>` tag. `<PARAM..>` tag has an attribute such as **color**, and a **value**. Inside the applet code, the applet can refer to that parameter by name to find its value.

e.g, We can change the color of the text displayed to a red by a `<PARAM..>` tag as follows:

```
<APPLET ....>
```

```
<PARAM = color VALUE= "red">
```

```
</APPLET>
```


Passing Parameters To Applet

Passing parameters to an applet code using `<PARAM>` tag is something similar to passing parameters to the `main()` method using command line arguments. To set up and handle parameters, we need to do two things.

1. Include appropriate `<PARAM>` tag.
2. Provide Code in the applet to use these parameters.

How to run an Applet?



There are two ways to run an applet

- By html file.
- By appletViewer tool (for testing purpose).

By html file.



```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet
{

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

}
```

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

Simple example of Applet by appletviewer tool:

To execute the applet by appletviewer tool,
create an applet that contains applet tag in comment and
compile it.

After that run it by: `appletviewer First.java`.

Now Html file is not required but it is for testing purpose only.

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet
{
    public void paint(Graphics g){
g.drawString("welcome to applet",150,150);
    }
}
/*
<applet code="First.class" width="300" height="300">
</applet>
*/
```

Java AWT



- **Java AWT** (Abstract Window Toolkit) is *an API to develop GUI or window-based applications* in java.
- Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.
- The java.awt package provides classes for AWT api such as TextField, Label, Text Area, Checkbox, Choice, List , RadioButton, etc.

Working with Graphics



Java's Graphics class includes methods for drawing many different types of shapes, from simple lines to polygons and text in a variety of fonts.

To draw a shape on the screen, we may call one of the methods available in the Graphics class. Following are some commonly used drawing methods in Graphics class.

Methods

Task Performed

drawRect()

Draws a hollow rectangle.

drawString()

Displays a Text string.

drawLine()

Draws a straight line.

drawPolygon()

Draws a hollow Polygon.

drawOval()

Draws a hollow Oval.



Working with Graphics

Methods

Task Performed

`drawArc()`

Draws a hollow arc.

`drawRoundRect()`

Draws a hollow rectangle with rounded corners.

`fillRect()`

Draws a filled rectangle.

`fillArc()`

Draws a filled arc.

`fillPolygon()`

Draws a filled Polygon.

`fillRoundRect()`

Draws a filled rectangle with rounded corners.

`fillOval()`

Draws a filled Oval.

Working with Graphics



Methods

clearRect()

copyArea()

getColor()

getFont()

setColor()

setFont()

Task Performed

Erases the rectangular area.

Copies the rectangular area to another area.

Retrieves the current drawing color.

Retrieves the currently used font.

Sets the drawing color.

Sets the font.



Lines And Rectangle

For drawing a Line in graphics we use **drawLine()** method from the Graphics class. The **drawLine()** method takes two pair of coordinates (x1,y1) and (x2,y2) as arguments and draws line between them.

e.g. **g.drawLine(10,10,50,50);**

Similarly for drawing a Rectangle in graphics we use **drawRect()** method from the Graphics class. The **drawRect()** method takes four arguments. The first two represent the x and y coordinates of the top left corner of the rectangle, and the remaining two represent the width and height of the rectangle.

e.g. **g.drawRect(10,60,40,30);**

We can also use **fillRect()**, **drawRoundRect()**, **fillRoundRect()** methods of Graphics class to the rectangle.

Note that, **drawRoundRect()**, **fillRoundRect()** takes six arguments as these two methods are used for drawing & filling round rectangle.

Circles And Ellipses

The Graphics class does not have any method for circles or ellipse. However, the **drawOval()** method are used to draw circle or an ellipse from the Graphics class. The **drawOval()** method takes four arguments. The first two represent the top left corner of the imaginary rectangle and the other two represent the width and height of the oval.

Note that, if the width and height are same, the oval become a circle.

e.g. `g.drawOval(20,20,250,150);`

Drawing Arcs



An arc is a part of an Oval. In fact, an Oval is series of arcs that are connected together in an orderly manner. The `drawArc()` method is used to draw arcs takes six arguments. The first four are the same as the arguments for `drawOval()` method and the last two represent the starting angle of the arc and degree around the arc.

e.g. `g.drawArc(20,20,100,150,180,180);`

We can use `fillArc()` method to fill the arc. It also take six arguments like `drawArc()` method.

e.g. `g.fillArc(60,120,80,40,160,160);`

Drawing Polygon



Polygons are shapes with many sides. A Polygon may be considered a set of lines connected together. The end of first line is the beginning of the second line, the end of the second is the beginning of the third, and so on.

We can draw polygons more conveniently by using **drawPolygon()** method. This method takes three arguments.

1. **An array of Integers containing X- coordinates.**
2. **An array of Integers containing Y- coordinates.**
3. **An Integers for the total number of points.**

We can also draw filled polygon by using **fillPolygon()** method.

Pgm for drawing Lines & Rectangles



```
import java.awt.*;
import java.applet.*;
public class LineRect extends Applet
{
    public void paint(Graphics g)
    {
        g.drawLine(10,10,50,50);
        g.drawRect(10,60,40,30);
        g.fillRect(60,40,30,80);
    }
}
```

```
<APPLET
CODE= LineRect.class
WIDTH= 250
HEIGHT= 200>
</APPLET>
```

Working with Colours



Colors can be controlled by accessing the Color class. The following table shows the available colors for Color class.

Available Colors

Red

Yellow

Blue

Orange

Pink

Cyan

Magenta

Black

White

Working with Colours



Images are drawn in the current color until the color is changed. Changing the color does not affect the color of previously drawn image.

e.g. `g.setColor(Color.blue);`

`g.drawString(" Welcome to Java Programming",100,120);`

This segment of code will print **Welcome to Java Programming** in blue color.

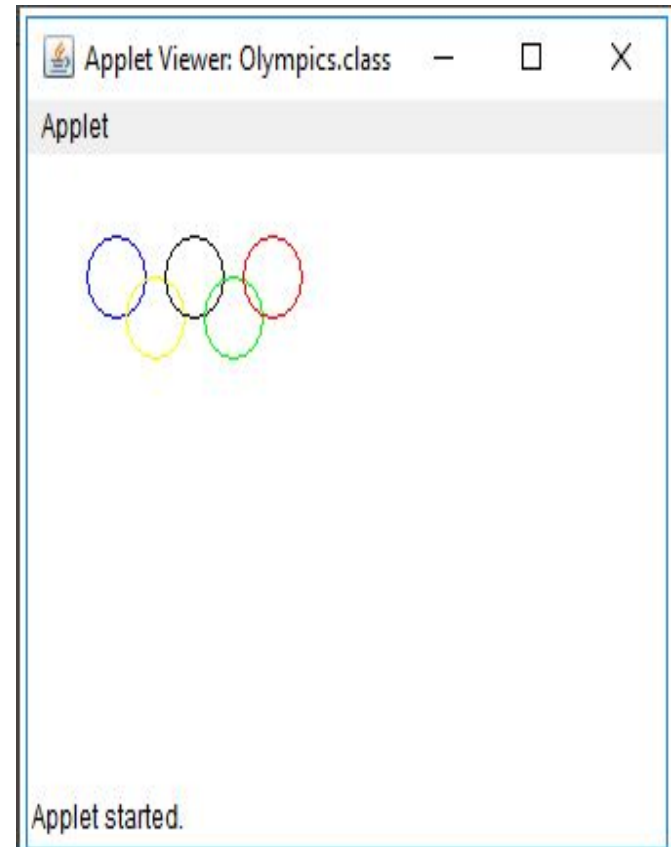
We can get the colors by controlling the RGB(red/green/blue) density values. Each of these 3 colors has 256 shades. It is possible to mix $256 \times 256 \times 256 = 2^{24}$ possible colors .

Difference between Application and Applet:

Java Application	Java Applet
Applications are just like a Java programs that can be execute independently without using the web browser.	Applets are small Java programs that are designed to be included with the HTML web document. They require a Java-enabled web browser for execution.
Application program requires a main function for its execution.	Applet does not require a main function for its execution.
Java application programs have the full access to the local file system and network.	Applets don't have local disk and network access.
Applications can access all kinds of resources available on the system.	Applets can only access the browser specific services. They don't have access to the local system.
Applications can executes the programs from the local system.	Applets cannot execute programs from the local machine.
An application program is needed to perform some task directly for the user.	An applet program is needed to perform small tasks or the part of it.

```
// Java program to Draw an Olympic
// Symbol using Java Applet
import java.awt.*;
import java.applet.*;

public class Olympics extends Applet
{
    public void paint(Graphics g)
    {
        g.setColor(Color.BLUE);
        g.drawOval(30, 30, 30, 30);
        g.setColor(Color.YELLOW);
        g.drawOval(50, 45, 30, 30);
        g.setColor(Color.BLACK);
        g.drawOval(70, 30, 30, 30);
        g.setColor(Color.GREEN);
        g.drawOval(90, 45, 30, 30);
        g.setColor(Color.RED);
        g.drawOval(110, 30, 30, 30);
    }
}
/*
<applet code="Olympics.class" width="300" height="300"
>
</applet>
*/
```



To run the applet in command line use the following commands

```
javac Olympics.java
> appletviewer Olympics.java
```

```
// Java program to Draw a
// Smiley using Java Applet
import java.applet.*;
import java.awt.*;

public class Smiley extends Applet
{
    public void paint(Graphics g)
    {
        // Oval for face outline
        g.drawOval(80, 70, 150, 150);
        // Ovals for eyes
        // with black color filled
        g.setColor(Color.BLACK);
        g.fillOval(120, 120, 15, 15);
        g.fillOval(170, 120, 15, 15);
        // Arc for the smile
        g.drawArc(130, 180, 50, 20, 180, 180);
    }
}
/*
<applet code="Olympics.class" width="300"
    height="300">
</applet>
*/
```



Note: To run the applet in command line use the following commands

```
□ javac Smiley.java
□ appletviewer Smiley.java
```

Java AWT



- **Java AWT** (Abstract Window Toolkit) is *an API to develop GUI or window-based applications in java.*
- Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.
- The java.awt package provides classes for AWT api such as TextField, Label, Text Area, Checkbox, Choice, List , RadioButton, etc.

Java AWT



- **Java AWT** (Abstract Window Toolkit) is *an API to develop GUI or window-based applications* in java.
- Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.
- The java.awt package provides classes for AWT api such as TextField, Label, Text Area, Checkbox, Choice, List , RadioButton, etc.



Java Event Handling

- Any program that uses GUI (graphical user interface) such as Java application written for windows, is event driven. Event describes the change in state of any object.

For Example : Pressing a button, Entering a character in Textbox, Clicking or Dragging a mouse, etc.

Components of Event Handling

Event handling has three main components,

- **Events** : An event is a change in state of an object.
- **Events Source** : Event source is an object that generates an event.
- **Listeners** : A listener is an object that listens to the event. A listener gets notified when an event occurs.

How Events are handled?

- A source generates an Event and send it to one or more listeners registered with the source. Once event is received by the listener, they process the event and then return. Events are supported by a number of Java packages, like **java.util**, **java.awt** and **java.awt.event**.

Important Event Classes and Interface

Event Classes	Description	Listener Interface
ActionEvent	generated when button is pressed, menu-item is selected, list-item is double clicked	ActionListener
MouseEvent	generated when mouse is dragged, moved, clicked, pressed or released and also when it enters or exit a component	MouseListener
KeyEvent	generated when input is received from keyboard	KeyListener
ItemEvent	generated when check-box or list item is clicked	ItemListener

Important Event Classes and Interface

TextEvent	generated when value of textarea or textfield is changed	TextListener
MouseWheelEvent	generated when mouse wheel is moved	MouseWheelListener
WindowEvent	generated when window is activated, deactivated, deiconified, iconified, opened or closed	WindowListener
ComponentEvent	generated when component is hidden, moved, resized or set visible	ComponentEventListener
ContainerEvent	generated when component is added or removed from container	ContainerListener
AdjustmentEvent	generated when scroll bar is manipulated	AdjustmentListener
FocusEvent	generated when component gains or loses keyboard focus	FocusListener

- **Steps to handle events:**
- Implement appropriate interface in the class.
- Register the component with the listener.

Registration Methods

- **TextArea**
 - `public void addTextListener(TextListener a){}`
- **Checkbox**
 - `public void addItemListener(ItemListener a){}`
- **Choice**
 - `public void addItemListener(ItemListener a){}`
- **List**
 - `public void addActionListener(ActionListener a){}`
 - `public void addItemListener(ItemListener a){}`

Java MouseListener Interface

- The Java MouseListener is notified whenever you change the state of mouse.
- It is notified against MouseEvent.
- The MouseListener interface is found in java.awt.event package. It has five methods.

```
public abstract void mouseClicked(MouseEvent e);  
public abstract void mouseEntered(MouseEvent e);  
public abstract void mouseExited(MouseEvent e);  
public abstract void mousePressed(MouseEvent e);  
public abstract void mouseReleased(MouseEvent e);
```

Example of Java MouseListener

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
//<applet code="circle.class" width=200 height=300></applet>
public class circle extends Applet
{
    int x,y;
    public void init()
    {
        add a=new add();
        addMouseListener(a);
    }
    public void paint(Graphics g)
    {
        g.setColor(Color.BLUE);
        g.fillOval(x,y,30,30);
    }
    class add implements MouseListener
    {
        public void mouseClicked(MouseEvent e)
        {
        }
        public void mouseEntered(MouseEvent e)
        {
            x=e.getX();
            y=e.getY();
            repaint();
        }
        public void mouseReleased(MouseEvent e){}
        public void mousePressed(MouseEvent e){}
        public void mouseExited(MouseEvent e){
        }
    }
}
```

Java MouseMotionListener Interface

The Java MouseMotionListener is notified whenever you move or drag mouse. It is notified against MouseEvent. The MouseMotionListener interface is found in java.awt.event package. It has two methods.

- **Methods of MouseMotionListener interface**

There are two methods found in MouseMotionListener interface are given below:

- **public abstract void** mouseDragged(MouseEvent e);
- **public abstract void** mouseMoved(MouseEvent e);

```
import java.awt.*;
import java.awt.event.*;
public class MouseMotionListenerExample extends Frame implements MouseMotionListener{
    MouseMotionListenerExample(){
        addMouseMotionListener(this);
        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }
    public void mouseDragged(MouseEvent e) {
        Graphics g=getGraphics();
        g.setColor(Color.BLUE);
        g.fillOval(e.getX(),e.getY(),20,20);
    }
    public void mouseMoved(MouseEvent e) {}

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

Java KeyListener Interface

- The Java KeyListener is notified whenever you change the state of key.
- It is notified against KeyEvent. T
- The KeyListener interface is found in java.awt.event package.
- It has three methods.

public abstract void keyPressed(KeyEvent e);

public abstract void keyReleased(KeyEvent e);

public abstract void keyTyped(KeyEvent e);

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.applet.*;
import java.awt.event.*;
import java.awt.*;
<applet code="Test" width=300, height=100> </applet>
public class Test extends Applet implements KeyListener
{
String msg=""; public void init()
{ addKeyListener(this);
}
public void keyPressed(KeyEvent k)
{
showStatus("KeyPressed");
}
public void keyReleased(KeyEvent k)
{
showStatus("KeyRealesed");
}
public void keyTyped(KeyEvent k)
{
msg = msg+k.getKeyChar();
repaint();
}
public void paint(Graphics g)
{
g.drawString(msg, 20, 40);
```




THANK YOU...