Chapter - 17

Class Color

Class Color

The Color class is predefined in java.awt package. This class contain public static final variables that represents colors. The following table shows the variables in Color class.

Color.black	Color.blue	Color.cyan	Color.darkGray	Color.gray
Color.green	Color.lightgray	Color.megenta	Color.orange	Color.pink
Color.red	Color.white	Color.yellow		

Note: These variables are also available in capital letters Ex: Color.BLACK

The Color class can also be used to create user-defined RGB colors using RGB values. The RGB values should range between 0-255.

The following table shows the constructors of **Color** class

Constructors	Meaning
Color(ColorSpace cspace, float[] components, float alpha)	Creates a color in the specified ColorSpace with the color components specified in the float array and the specified alpha.
Color(float r, float g, float b) Creates an opaque	sRGB color with the specified red, green, and blue values in the range (0.0 - 1.0).
Color(float r, float g, float b, float a)	Creates an sRGB color with the specified red, green, blue, and alpha values in the range (0.0 - 1.0).
Color(int rgb)	Creates an opaque sRGB color with the specified combined RGB value consisting of the red component in bits 16-23, the green component in bits 8-15, and the blue component in bits 0-7
Color(int rgba, boolean hasalpha)	Creates an sRGB color with the specified combined RGBA value consisting of the alpha component in bits 24-31, the red component in bits 16-23, the green component in bits 8-15, and the blue component in bits 0-7.
Color(int r, int g, int b)	Creates an opaque sRGB color with the specified red, green, and blue values in the range (0 - 255).
Color(int r, int g, int b, int a)	Creates an sRGB color with the specified red, green, blue, and alpha values in the range (0 - 255).

The following table shows the methods of Color class

Methods	Meaning
Color **brighter**()	Creates a new Color that is a brighter version of this Color.
Color **darker**()	Creates a new Color that is a darker version of this Color.
int **getBlue**()	Returns the blue component in the range 0-255 in the default sRGB space.
int **getGreen**()	Returns the green component in the range 0-255 in the default sRGB space.
static Color **getHSBColor**(float h, float s, float b)	Creates a Color object based on the specified values for the HSB color model.
int **getRed**()	Returns the red component in the range 0-255 in the default sRGB space.
int **getRGB**()	Returns the RGB value representing the color in the default sRGB ColorModel.
int **hashCode**()	Computes the hash code for this Color.
static int **HSBtoRGB**(float hue, float saturation, float brightness)	Converts the components of a color, as specified by the HSB model, to an equivalent set of values for the default RGB model.
String **toString**()	Returns a string representation of this Color.

Bin>edit color1.java

```
import java.awt.*;
import java.applet.*;
public class color2 extends Applet
{
   public void init()
   {
        Color c1=new Color(100,50,150);
        setBackground(c1);
        Color c2=new Color(50,200,50);
        setForeground(c2);
   }
```

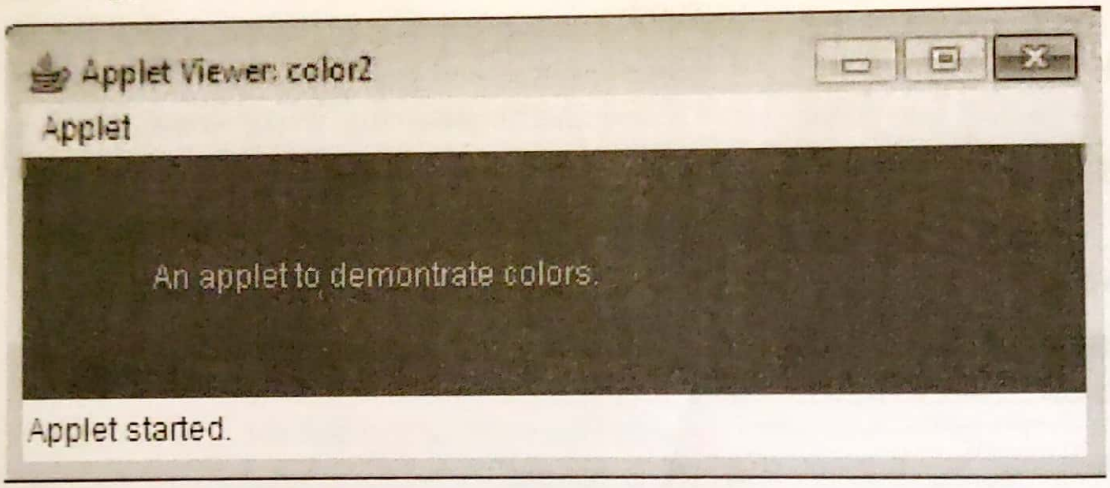
```
    public void paint(Graphics g)
    {
        g.drawString("An applet to demontrate colors.",50,50);
    }
}
```

Bin>edit color2.java

Bin>edit color2.html

```
<applet code="color2" width=400 height=400>
</applet>
```

Bin>appletviewer color2.html



Class Font

The Font class is a predefined class in java.awt package. The Font class represents fonts, which are used to render text in a visible way. The Font class can be used to display the text in different styles(**bold**, **italic**, **plain**), sizes and fonts.

The font can be applied to component using the **setFont()** method of Component class. The signature of this method is

public void setFont (Font fobj)

The following table shows variables of **Font** class.

Fields	Meaning
static int **BOLD**	The bold style constant.
static int **ITALIC**	The italicized style constant.
static int **PLAIN**	The plain style constant.

The following table shows constructors of **Font** class.

Constructors	Meaning
protected **Font**(Font font)	Creates a new Font from the specified font.
Font(String name, int style, int size)	Creates a new Font from the specified name, style and point size.

The following table shows methods of **Font** class.

Methods	Meaning
static Font **createFont**(int fontFormat, File fontFile)	Returns a new Font using the specified font type and the specified font file.
String **getFamily**()	Returns the family name of this Font.
String **getFamily**(Locale l)	Returns the family name of this Font, localized for the specified locale.
static Font **getFont**(String nm)	Returns a Font object from the system properties list.
static Font **getFont**(String nm, Font font)	Gets the specified Font from the system properties list.
String **getFontName**()	Returns the font face name of this Font.
float **getItalicAngle**()	Returns the italic angle of this Font.
String **getName**()	Returns the logical name of this Font.
int **getSize**()	Returns the point size of this Font, rounded to an integer.
int **getStyle**()	Returns the style of this Font.
boolean **isBold**()	Indicates whether or not this Font object's style is BOLD.
boolean **isItalic**()	Indicates whether or not this Font object's style is ITALIC.
boolean **isPlain**()	Indicates whether or not this Font object's style is PLAIN.
String **toString**()	Converts this Font object to a String representation.

Bin>edit font1.java

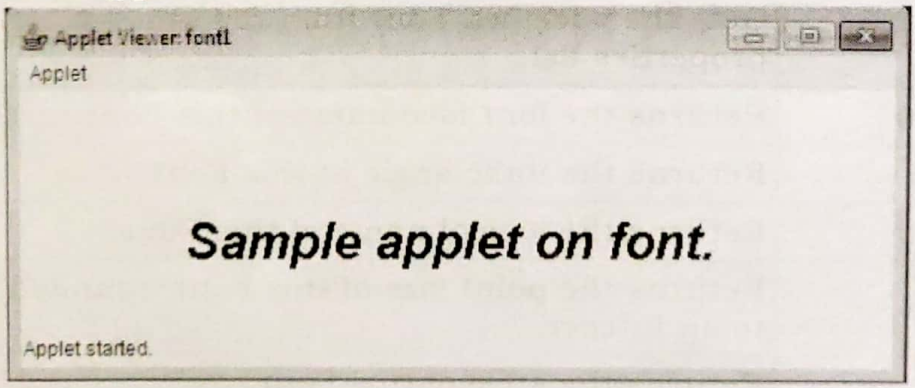
```
import java.applet.*;
import java.awt.*;
public class font1 extends Applet
{
   Font f;
   public void init()
   {
       f=new Font("Arial",Font.BOLD | Font.ITALIC ,30);
       setFont(f);
   }
   public void paint(Graphics g)
   {
       g.drawString("Sample applet on font.",100,100);
   }
}
```

Bin>edit font1.java

Bin>edit font1.html

```
<applet code="font1" width=400 height=400>
</applet>
```

Bin>appletviewer font1.html



Chapter - 18

Graphics

Graphics

In Java all drawing takes place via a **Graphics** object. This is an instance of the class **java.awt.Graphics**.

Initially the Graphics object we use will be the one passed as an argument to an applet's **paint()** method. We can also obtain the graphics object using **getGraphics()** method of **Component** class. Everything related to drawing in an applet transfers directly to drawing in other objects like **Panels**, **Frames**, **Buttons**, **Canvas** and more.

The following table shows methods of **Graphics** class

Methods	Meaning
abstract Graphics **create**()	Creates a new Graphics object that is a copy of this Graphics object.
abstract void **dispose**()	Disposes of this graphics context and releases any system resources that it is using.
void **draw3DRect**(int x, int y, int width, int height, boolean raised)	Draws a 3-D highlighted outline of the specified rectangle.
abstract void **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.
abstract boolean **drawImage**(Image img, int x, int y, Color bgcolor, ImageObserver observer)	Draws as much of the specified image as is currently available.
abstract void **drawLine**(int x1, int y1, int x2, int y2)	Draws a line, using the current color, between the points (x1, y1) and (x2, y2) in this graphics context's coordinate system.
abstract void **drawOval**(int x, int y, int width, int height)	Draws the outline of an oval.
abstract void **drawPolygon**(int[] xPoints, int[] yPoints, int nPoints)	Draws a closed polygon defined by arrays of *x* and *y* coordinates.
void **drawPolygon**(Polygon p)	Draws the outline of a polygon defined by the specified Polygon object.

void **drawRect**(int x, int y, int width, int height)	Draws the outline of the specified rectangle.
abstract void **drawRoundRect**(int x, int y, int width, int height, int arcWidth, int arcHeight)	Draws an outlined round-cornered rectangle using this graphics context's current color.
abstract void **drawString**(String str, int x, int y)	Draws the text given by the specified string, using this graphics context's current font and color.
void **fill3DRect**(int x, int y, int width, int height, boolean raised)	Paints a 3-D highlighted rectangle filled with the current color.
abstract void **fillArc**(int x, int y, int width, int height, int startAngle, int arcAngle)	Fills a circular or elliptical arc covering the specified rectangle.
abstract void **fillOval**(int x, int y, int width, int height)	Fills an oval bounded by the specified rectangle with the current color.
abstract void **fillPolygon**(int[] xPoints, int[] yPoints, int nPoints)	Fills a closed polygon defined by arrays of *x* and *y* coordinates.
void **fillPolygon**(Polygon p)	Fills the polygon defined by the specified Polygon object with the graphics context's current color.
abstract void **fillRect**(int x, int y, int width, int height)	Fills the specified rectangle.
abstract void **fillRoundRect**(int x, int y, int width, int height, int arcWidth, int arcHeight)	Fills the specified rounded corner rectangle with the current color.
abstract Color **getColor**()	Gets this graphics context's current color.
abstract Font **getFont**()	Gets the current font.
FontMetrics **getFontMetrics**()	Gets the font metrics of the current font.
abstract FontMetrics **getFontMetrics**(Font f)	Gets the font metrics for the specified font.
abstract void **setColor**(Color c)	Sets this graphics context's current color to the specified color.

abstract void **setFont**(Font font)	Sets this graphics context's font to the specified font.
abstract void **setPaintMode**()	Sets the paint mode of this graphics context to overwrite the destination with this graphics context's current color.
abstract void **setXORMode**(Color c1)	Sets the paint mode of this graphics context to alternate between this graphics context's current color and the new specified color.
String **toString**()	Returns a String object representing this Graphics object's value.

Drawing Lines

Drawing straight lines with Java is easy. The drawLine() method of Graphics class can be used to draw straight. The signature of drawLine() method is

public abstract void drawLine(int x1, int y1, int x2, int y2)

Draws a line, using the current color, between the points (x1, y1) and (x2, y2) in the current graphics context's coordinate system.

Example:

g.drawLine(x1, y1, x2, y2)

Where (x1, y1) and (x2, y2) are the endpoints of your lines and g is the Graphics object using which we are drawing with.

The following program draws lines on applet.

Bin>edit line1.java

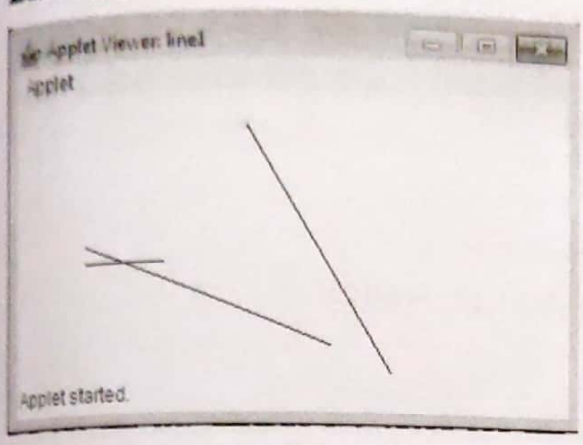
```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class line1 extends Applet
{
      public void paint(Graphics g)
      {
            g.drawLine(100,100,50,100);
            g.drawLine(50,90,200,150);
            g.drawLine(150,20,250,190);
      }
}
```

Bin>javac line1.java

Bin>edit line1.html

```
<applet code="line1" width=400 height=400>
</applet>
```

Bin>appletviewer line1.html



A program to draw lines using mouse events.

The following program draw lines using mouse pressed and mouse released events. When mouse is pressed we obtain **x1**, **y1** of mouse pointer and when the mouse is released again we obtain **x2**, **y2** of mouse pointer on the applet. Using these **x1**, **y1** and **x2**,**y2** a line is drawn.

Bin>edit mouseline1.java

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class mouseline1 extends Applet
{
    int x1=0,y1=0,x2=0,y2=0;
    public void init()
    {
          addMouseListener(new mymousehandler());
    }
    class mymousehandler extends MouseAdapter
    {
          public void mousePressed(MouseEvent m)
          {
                x1=m.getX();
                y1=m.getY();
          }
```

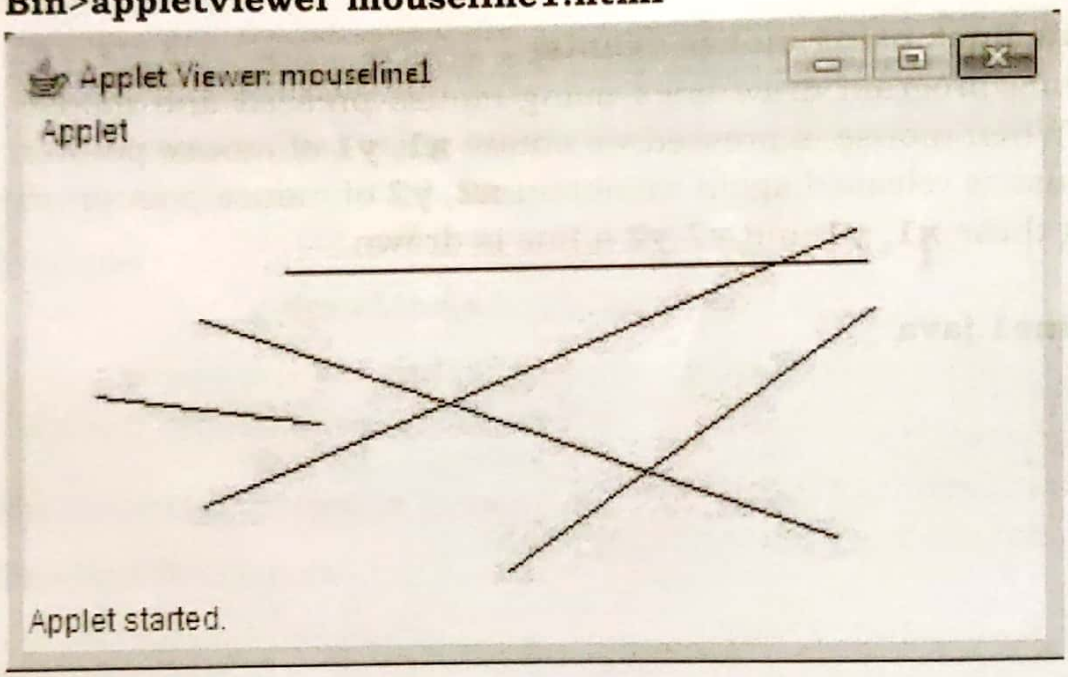
```
        public void mouseReleased(MouseEvent m)
        {
                x2=m.getX();
                y2=m.getY();
                Graphics g=getGraphics();
                g.drawLine(x1,y1,x2,y2);
        }
    }
}
```

Bin>javac mouseline1.java

Bin>edit mouseline1.html

```
<applet code="mouseline1" width=400 height=400>
</applet>
```

Bin>appletviewer mouseline1.html



A program to draw pixels using drawLine() method

The following program prints pixel at the point where the mouse is clicked.

Bin>edit pixel1.java

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class pixel1 extends Applet
{
    int x1=0,y1=0;
```

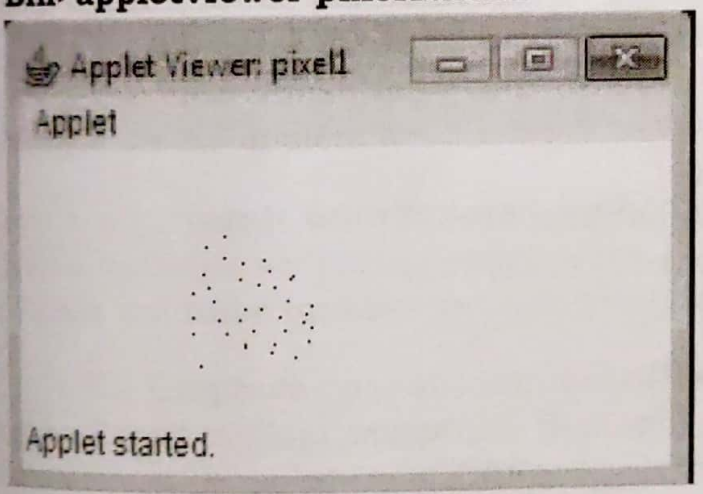
```
    public void init()
    {
          addMouseListener(new mymousehandler());
    }
    class mymousehandler extends MouseAdapter
    {
          public void mouseClicked(MouseEvent m)
          {
                x1=m.getX();
                y1=m.getY();
                Graphics g=getGraphics();
                g.drawLine(x1,y1,x1,y1);
          }
    }
}
```

Bin>javac pixel1.java

Bin>edit pixel1.html

```
<applet code="pixel1" width=400 height=400>
</applet>
```

Bin>appletviewer pixel1.html



A program on free handle using mouse drag event.

In this program, we create a pencil to draw on the applet using mouse drag event. The direction in which the mouse is dragged, the line is drawn. In this program we handle two events i.e. mouse pressed and mouse dragged. Mouse pressed event is taken as the starting point of pencil and drawing is done by the dragged event.

Bin>edit freehandle.java

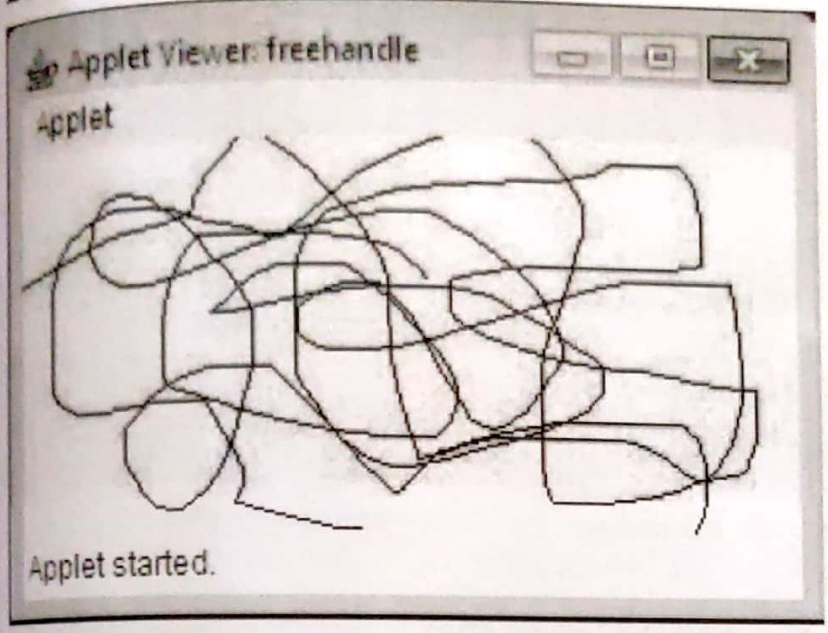
```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class freehandle extends Applet
{
    int x1=0,y1=0,x2=0,y2=0;
    public void init()
    {
        addMouseListener(new mymousehandler());
        addMouseMotionListener(newmymousemotionhandler());
    }
    class mymousehandler extends MouseAdapter
    {
        public void mousePressed(MouseEvent m)
        {
            x1=m.getX();
            y1=m.getY();
        }
    }
    class mymousemotionhandler extends MouseMotionAdapter
    {
        public void mouseDragged(MouseEvent m)
        {
            x2=m.getX();
            y2=m.getY();
            Graphics g=getGraphics();
            g.drawLine(x1,y1,x2,y2);
            x1=x2;
            y1=y2;
        }
    }
}
```

Bin>javac freehandle.java

Bin>edit freehandle.html

```
<applet code="freehandle" width=400 height=400>
</applet>
```

Bin>appletviewer freehandle.html



Drawing Rectangles

The drawRect() method of **Graphics** class can be used to draw rectangles. The signature of the **drawRect()** is

public void drawRect(int x, int y, int width, int height)

The first **int x** is the left hand side of the rectangle, the second **int y** is the top of the rectangle, the third is the **width** and the fourth is the **height** of rectangle. With these four co-ordinates the **drawRect()** method draws rectangle on the applet.

There is no separate **drawSquare()** method in the **Graphics** classs. A square is just a rectangle with equal length sides, so to draw a square call **drawRect()** and pass the same number for both the height and width arguments.

The **Graphics** class also contain **fillRect()** method which works similar to **drawRect()** method except the **fillRect()** method fills the rectangle object with color. The signature of **fillRect()** method is

public void fillRect(int x, int y, int width, int height)s

A program to draw rectangles using drawRect() and fillRect() methods.

Bin>edit rect1.java

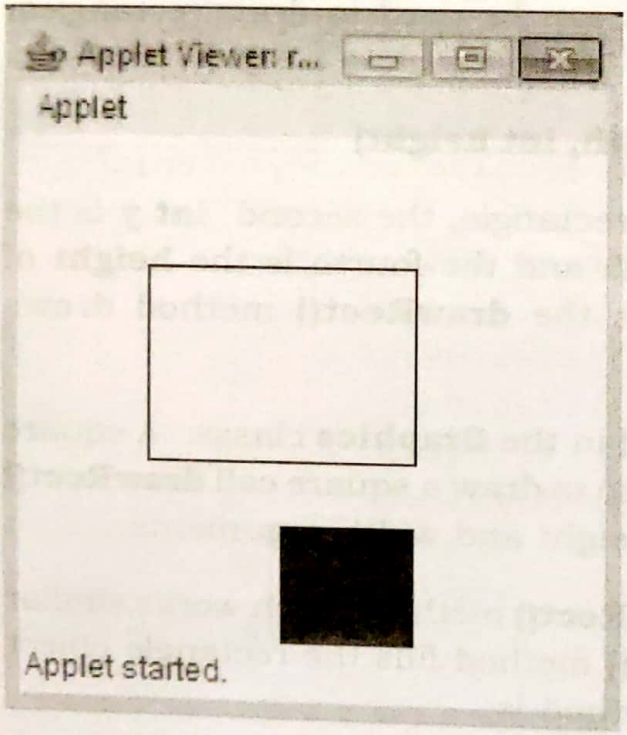
```
import java.applet.*;
import java.awt.*;
public class rect1 extends Applet
{
    public void paint(Graphics g)
    {
        g.drawRect(50,50,100,75);
        g.fillRect(100,150,50,50);

    }
}
```

Bin>javac rect1.java

Bin>edit rect1.html

```
<applet code="rect1" width=400 height=400>
</applet>
```

Bin>appletviewer rect1.html



A program to draw rectangles using mouse.

In this program, we can draw the rectangles using mouse. When mouse is pressed we obtain **x1**, **y1** points which are the top, left points of the rectangle. When mouse is dragged and released another two points of mouse **x2**, **y2** is obtained. With these four co-ordinates we can find the width and height of rectangle ex: **width=x2-x1** , **height=y2-y1**. These co-ordinates are passed to **drawRect()** method to draw rectangle.

Bin>edit mouserect1.java

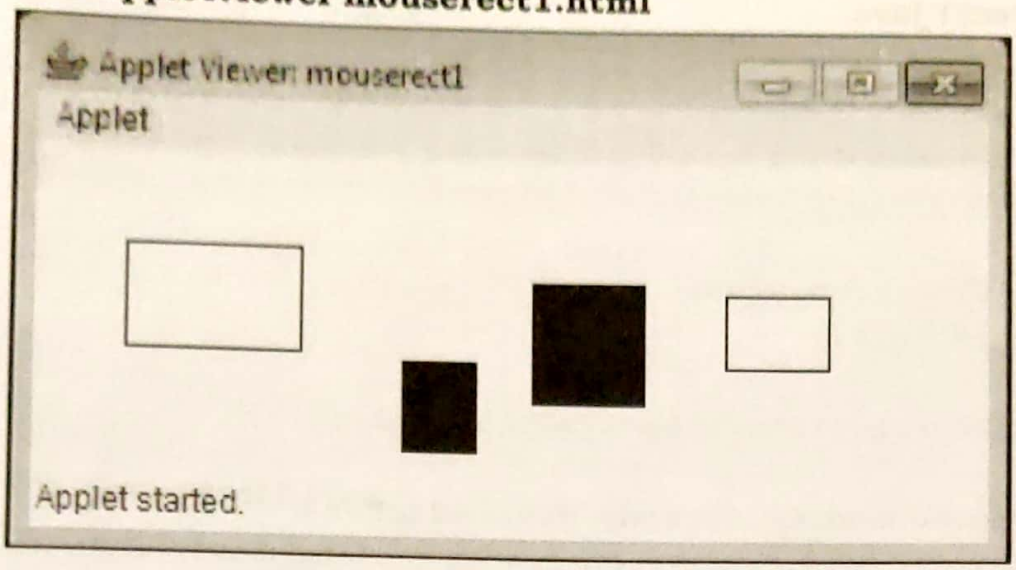
```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class mouserect1 extends Applet
{
    int x1=0,y1=0,x2=0,y2=0;
    public void init()
    {
          addMouseListener(new mymousehandler());
    }
    class mymousehandler extends MouseAdapter
    {
          public void mousePressed(MouseEvent m)
          {
                x1=m.getX();
                y1=m.getY();
          }
          public void mouseReleased(MouseEvent m)
          {
                int w,h;
                x2=m.getX();
                y2=m.getY();
                Graphics g=getGraphics();
                w=Math.abs(x2-x1);
                h=Math.abs(y2-y1);
                if(x1<x2)
                    g.drawRect(x1,y1,w,h);
                  else if(x1>x2)
                    g.fillRect(x2,y1,w,h);
          }
    }
}
```

Bin>javac mouserect1.java

Bin>edit mouserect1.html

```
<applet code="mouserect1" width=400 height=400>
</applet>
```

Bin>appletviewer mouserect1.html



Drawing Round Rectangles

The **drawRoundRect()** and **fillRoundRect()** methods of **Graphics** class can be used to draw rounded rectangles. These methods have extra two arguments then the **drawRect()** and **fillRect()** methods. The signature of these methods are:

public void drawRoundRect(int x, int y, int width, int height, int x-diameter, int y-diameter)

public void drawRoundRect(int x, int y, int width, int height, int x-diameter, int y-diameter)

A program to draw rounded rectangles using drawRoundRect() and fillRoundRect() methods.

Bin>edit roundrect1.java

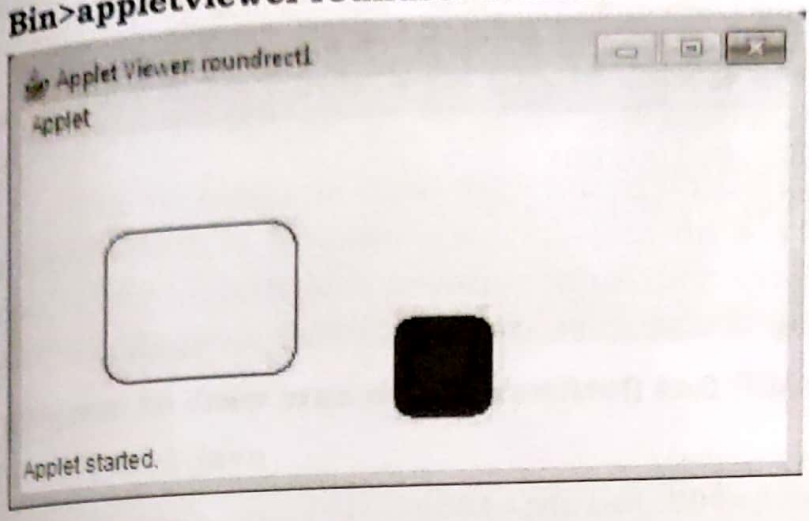
```
import java.applet.*;
import java.awt.*;
public class roundrect1 extends Applet
{
    public void paint(Graphics g)
    {
          g.drawRoundRect(50,50,100,75,15,15);
          g.fillRoundRect(100,150,50,50,10,10);
    }
}
```

Bin>javac roundrect1.java

Bin>edit roundrect1.html

```
<applet code="roundrect1" width=400 height=400>
</applet>
```

Bin>appletviewer roundrect1.html

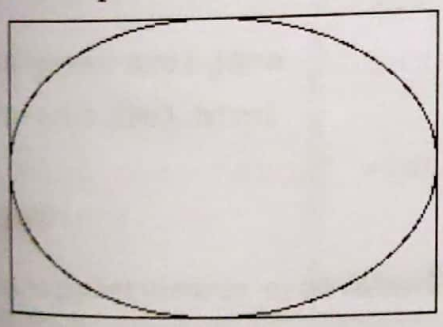


Ovals and Circles

Ovals can be drawn using **drawOval()** method of Graphics class. This method draws outlined oval. The signature of the **drawOval()** method is

public void drawOval(int left, int top, int width, int height)

Instead of the dimensions of the oval itself, the dimensions of the smallest rectangle which can enclose the oval are specified. The oval is drawn as large as it can be to touch the rectangle's edges at their centers as shown in below picture.



There is no special method to draw a circle. Just draw an oval inside a square.

To draw filled ovals we can use **fillOval()** method whose signature is similar to **drawOval()** i.e.

public void fillOval(int left, int top, int width, int height)

A program to draw outlined and filled ovals

Bin>edit oval1.java

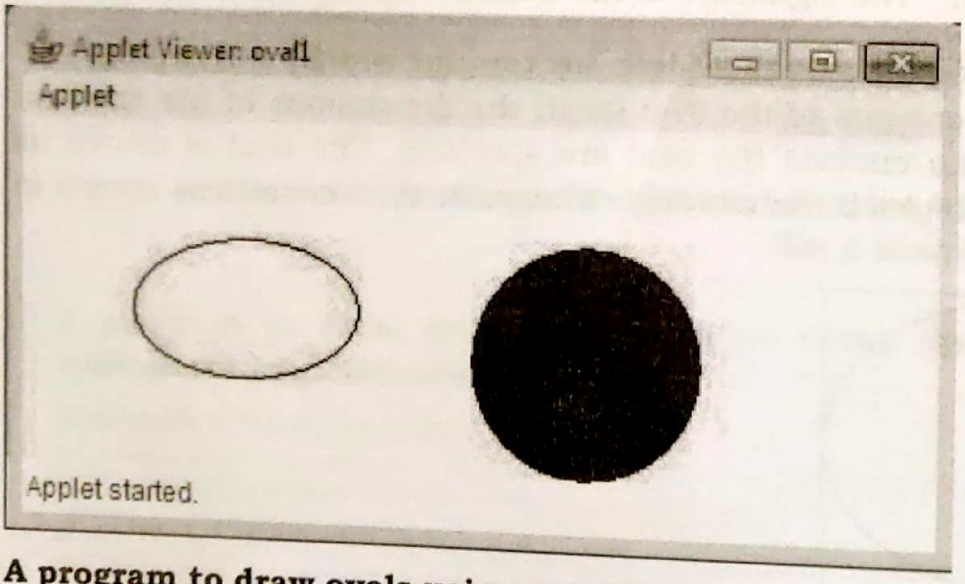
```
import java.awt.*;
import java.applet.*;
public class oval1 extends Applet
{
    public void paint(Graphics g)
    {
        g.drawOval(50,50,100,60);
        g.fillOval(200,50,100,100);
    }
}
```

Bin>javac oval1.java

Bin>edit oval1.html

```
<applet code="oval1" width=400 height=400>
</applet>
```

Bin>appletviewer oval1.html



A program to draw ovals using mouse events.

The program is same as drawing rectangles using mouse. Execute that program replacing **drawRect** method name with **drawOval** and **fillRect** with **fillOval**.

Drawing Arcs

Java also has methods to draw outlined and filled arcs. They're similar to **drawOval()** and **fillOval()** but we must also specify a starting and ending angle for the arc. Angles are given in degrees. The signatures are:

public void drawArc(int left, int top, int width, int height,
int startAngle, int stopAngle)

public void fillArc(int left, int top, int width, int height,
int startAngle, int stopAngle)

The rectangle is filled with an arc of the largest circle that could be enclosed within it. The arc is drawn from the startAngle to stopAngle of the circle. If the stopAngle is negative value then the arc is drawn in clockwise, and if stopAngle is positive the arc is drawn in anti-clockwise.

A program to draw arcs using drawArc() and fillArc() mehods

Bin>edit arc1.java

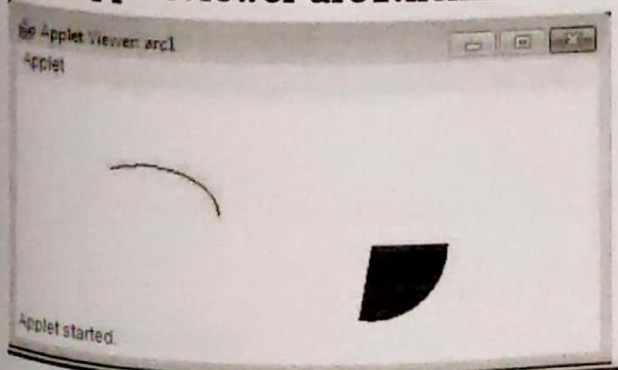
```
import java.awt.*;
import java.applet.*;
public class arc1 extends Applet
{
    public void paint(Graphics g)
    {
          g.drawArc(50,50,100,60,0,120);
          g.fillArc(200,50,100,100,0,-100);
    }
}
```

Bin>javac arc1.java

Bin>edit arc1.html

```
<applet code="arc1" width=400 height=400>
</applet>
```

Bin>appletviewer arc1.html



Drawing Polygons

It is possible to draw arbitrarily shaped figures using **drawPolygon()** and **fillPolygon()** metods. Polygons are defined by their corners. The signature of these methods are:

public void drawPolygon(int xpoints[], int ypoints[], int npoints)

public void fillPolygon(int xpoints[], int ypoints[], int npoints)

xpoints is an array that contains the **x** coordinates of the polygon. **ypoints** is an array that contains the **y** coordinates. Both should have the length **npoints**.

A program to draw polygon.

Bin>edit polygon1.java

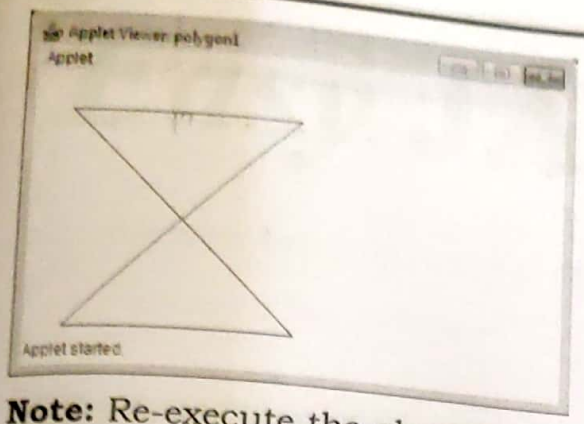
```
import java.awt.*;
import java.applet.*;
public class polygon1 extends Applet
{
    public void paint(Graphics g)
    {
          int xpoints[]={30,200,30,200,30};
          int ypoints[]={30,30,200,200,30};
          int npoints=5;
          g.fillPolygon(xpoints,ypoints,npoints);

    }
}
```

Bin>javac polygon1.java

Bin>edit polygon1.html

```
<applet code="polygon1" width=400 height=400>
</applet>
```

Bin>appletviewer polygon1.html



Note: Re-execute the above program replacing **darwPolygon()** method name with

fillPolygon().

A program to draw shapes using colors.

Bin>edit drawcolor.java

```
import java.awt.*;
import java.applet.*;
public class drawcolor extends Applet
{
    public void paint(Graphics g)
    {
          g.setColor(Color.red);
          g.drawLine(100,100,200,150);
          g.setColor(Color.blue);
          g.drawRect(200,200,50,50);
    }
}
```

Bin>javac drawcolor.java

Bin>edit drawcolor.html

```
<applet code="drawcolor" width=400 height=400>
</applet>
```

Bin>appletviewer drawcolor.html