Lecture 25-26

Java Applets

In this lecture and next lecture we will discuss Java Applets. Java applets are special kind of programs (given by Java) which can be embedded into HTML codes. Hope you know about HTML codes deployed on some server machine and downloaded to client machine when request sent (usually) by typing a URL in browser. HTML codes are static codes to bring some dynamicity into these codes applets can be used. Applets are java codes thus translated to class file when compiled. If you have java class file containing applet inside, you may attach it with any HTML code by writing applet tag as shown below:

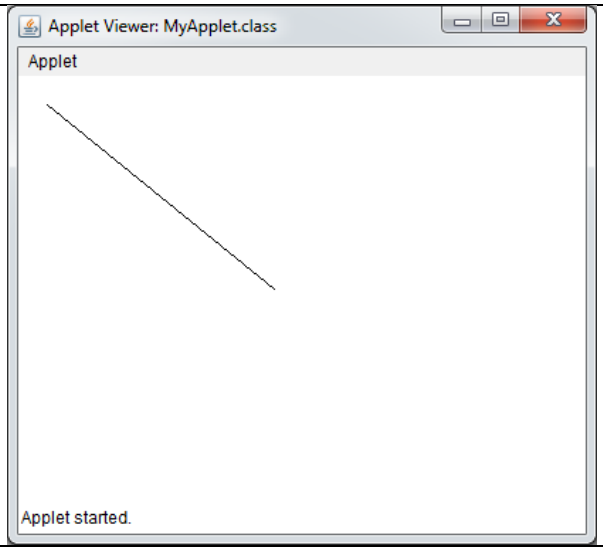
```
<APPLET CODE=SimpleApplet.class WIDTH=200 HEIGHT=100>
```

Here SimpleApplet is the name of java applet class; you may have any other name likely. A big condition for running applet code on client machine is availability of JVM (Java Virtual Machine). If JVM not installed on client side an empty box will appear instead of actual applet. Width & Height parameters mention the window size where your applet will appear in your HTML page. Typically we do create simulations using applets that required graphics. Graphics are also interesting therefor this is a good idea to introduce applets for the practice of coding like loop, array etc. Before discussing coding of applets first see a sample applet:

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

public class MyApplet extends JApplet{
  public void paint(Graphics g){
    g.drawLine(20,20,180,150);
  }
}
```

Output shown on right hand side



Here you will save code with file name ***MyApplet*** as usual but if you are using ***textpad*** then to run instead off **Ctrl+2** press **Ctrl+3**. If you are using ***Eclipse*** (Preferred to use) if you write click on code it will give option Run Java Applet. To create an Applet class you have to write ***extends JApplet***. For JApplet you have to write " import javax.swing.*". Secondly like main method applets have paint method executed automatically. There are other methods as well some of which we may discuss later.

Paint Method

As normal programs starts execution from main method applet starts execution from paint method (usually). Method paint has parameter of type Graphics. Using this object we can call many methods to draw graphics. Like, we have used **drawLine** method in above code. Other methods are:

- `drawOval`
- `drawString`
- `setColor`
- `drawArc`
- `drawPolygon`
- `fillOval`

Init Method

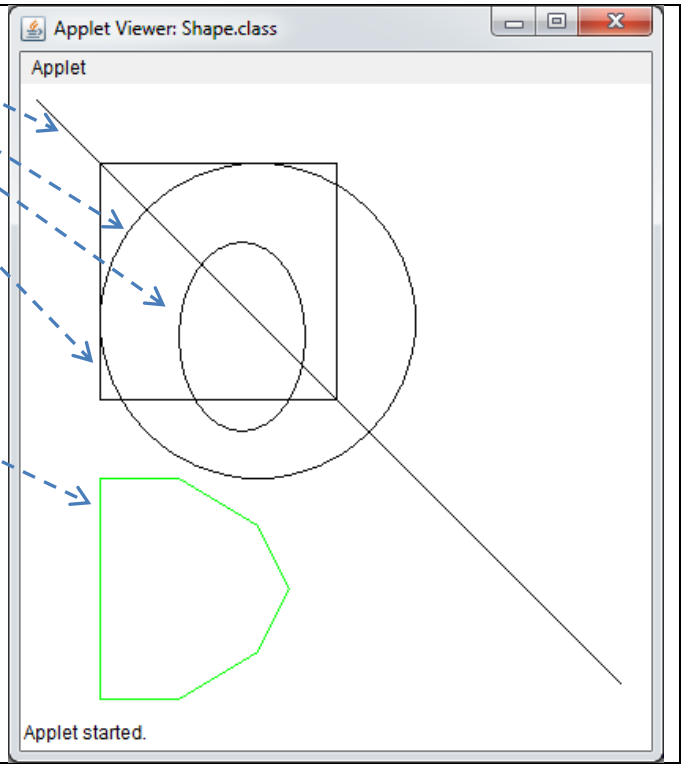
Init method is also useful it is executed before paint method. ***init*** stands for initialization. Usually we call ***setSize*** method to set size of applet.

Examples

We start with a very simple applet having shapes of different types:

```
public void paint(Graphics g){
  g.drawLine(10, 10, 380, 380);
  g.drawOval(50, 50, 200, 200);
  g.drawOval(100, 100, 80, 120);
  g.drawRect(50, 50, 150, 150);
  g.setColor(Color.GREEN);
  int x[]={50,100,150,170,150,100,50};
  int y[]={250,250,280,320,360,390,390};
  g.drawPolygon(x, y, 7);
}
```

Output shown on right hand side



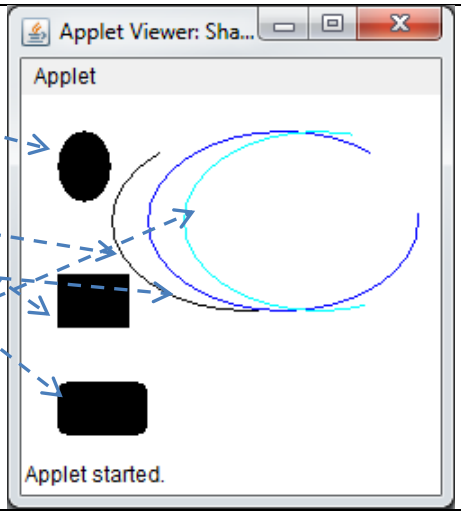
In above example ***drawLine*** method has four parameters namely x1, y1, x2 & y2. ***drawOval*** method has four parameters namely x1, y1 [upper left corner], radius x [width] & radius y [height]. If we keep radius x & radius y same we get circle, you can observe first call is generating a circle whereas; second call having different parameters for radius x & radius y generating oval. ***drawRect*** method has four parameters namely x1, y1 [upper left corner], width & height. Before last shape we have changed the color so you can see polygon with green color and of different shade if you are reading from a gray scale print out. Polygon is defined as a closed shape having 3 or more points; last point is automatically connected to first point. ***drawPolygon*** method has 3 parameters. First two points are arrays of int type where corresponding element of each array defines a point like x[0] & y[0] defines first point, x[1] &y[1] defines second point and so on. The third parameter of this method is number of points which shows you may send a large array but you can specify that how many points to be considered to draw polygon. Here student must note that our polygon has 7 points and the same we have mentioned in 3rd argument. Similarly there are 7 lines drawn to complete the shape whereas two points define 1 line thus 7 points defines 6 lines where each point comes twice, first as ending point than as starting point. Automatically function draws another line where last point act as starting point & first point act as ending points and as a result we get complete closed figure. If you will specify lesser points [not less than 3] you will still get a closed figure but may not be required one. However, if you send number of points greater than size of array a run-time exception will occur.

See another example where we use more functions of Graphics class. Fill methods have figures which are filled in compare to draw methods having hollow figures. ***drawArc*** method has 6 parameters. First 2 parameters are upper left corner, 3rd and 4th parameters are width & height of

arc. Fifth parameter is the starting angle of the arc and last 6th parameter is the further curving angle after starting angle:

```
public void paint(Graphics g){
  g.fillOval(20, 20, 30, 40);
  g.fillRect(20, 100, 40, 30);
  g.fillRoundRect(20, 160, 50, 30, 10, 10);
  g.drawArc(50, 20, 150, 100, 130, 145);
  g.setColor(Color.BLUE);
  g.drawArc(70, 20, 150, 100, 50, 315);
  g.setColor(Color.CYAN);
  g.drawArc(90, 20, 150, 100, 75, 215);
}
```

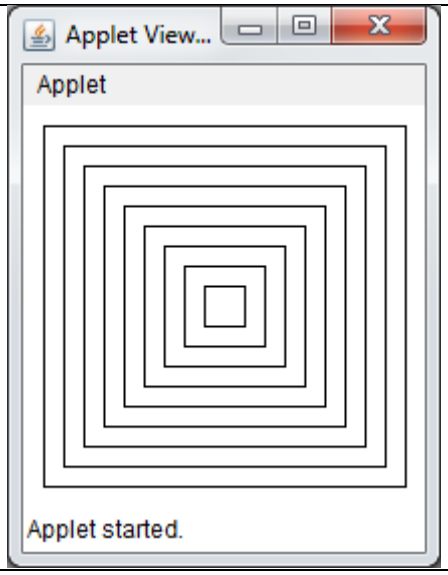
Output shown on right hand side



Applets have graphics where graphics are interesting comparing to alphabets and numbers. Applets are good for practicing loops and arrays. See an example of a drawing having set of nested squares:

```
public void paint(Graphics g){
  int x=10, y=10;
  int w=180, h=180;
  for ( ;x<=90;x=x+10, y=y+10){
    g.drawRect(x, y, w, h);
    w=w-20;
    h=h-20;
  }
}
```

Output shown on right hand side



Similarly we can make an animation showing things are moving by changing color to foreground and background. When an object is drawn in foreground color than in background color human feel object is moving. See the code where we are drawing rectangle in foreground color and after small pause in background color so if you run this code you can see triangle shrinking slowly:

```
public void paint(Graphics g){
  int x=10, y=10;
  int w=180, h=180;
  Color color;
  for ( ;x<=90;x=x+5, y=y+5){
    color=Color.BLACK;
    g.setColor(color);
    g.drawRect(x, y, w, h);
    try{
      Thread.sleep(100);
    }catch(Exception e){}
    color=Color.WHITE;
    g.setColor(color);
    g.drawRect(x, y, w, h);
    w=w-10;
    h=h-10;
  }
}
```

100 is time in milliseconds

Redrawing rectangle at same position with same width & height after setting color to background to give effect that triangle is no more there.

Last example is just an extension of previous example where we are animating rectangle from large to small and small to large. We have an infinite loop over that so that it may go on and on:

```
public void paint(Graphics g){
  int x=10, y=10;
  int w=180, h=180;
  Color color;
  while (true){
    for ( ;x<=90;x=x+5, y=y+5){
      color=Color.BLACK;
      g.setColor(color);
      g.drawRect(x, y, w, h);
      try{
        Thread.sleep(200);
      }catch(Exception e){}
      color=Color.WHITE;
      g.setColor(color);
      g.drawRect(x, y, w, h);
      w=w-10;
      h=h-10;
    }
```

Moving rectangle from larger to smaller

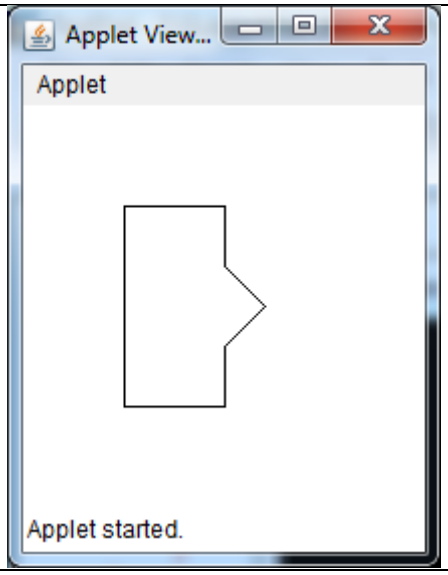
```
    for ( ;x>=10;x=x-5, y=y-5){
      color=Color.BLACK;
      g.setColor(color);
      g.drawRect(x, y, w, h);
      try{
        Thread.sleep(200);
      }catch(Exception e){}
      color=Color.WHITE;
      g.setColor(color);
      g.drawRect(x, y, w, h);
      w=w+10;
      h=h+10;
    }
  }
}
```

Moving rectangle from smaller to larger, just addition & subtraction reversed from first loop

There are some methods requiring more discussion like ***drawPolygon***. Before discussing this method see a code for drawing a polygon with the help of lines:

```
public void paint(Graphics g){
  g.drawLine(50,50,100,50);
  g.drawLine(100,50,100,80);
  g.drawLine(100,80,120,100);
  g.drawLine(120,100,100,120);
  g.drawLine(100,120,100,150);
  g.drawLine(100,150,50,150);
  g.drawLine(50,150,50,50);
}
```

Output shown on right hand side



Apparently this is simple code but think if we have to change location or size we have to modify arguments very carefully because each point is coming twice. A relatively easier way is to create two arrays and call ***drawPolygon*** method. It has more advantages we will discuss shortly:

```
public void paint(Graphics g){
  int x[]={50,100,100,120,100,100,50};
  int y[]={50,50,80,100,120,150,150};
  g.drawPolygon(x,y,7);
}
```

This method draws lines between points where each point comprises of corresponding elements from both arrays. Automatically last line is drawn between last & first point. One of the advantage

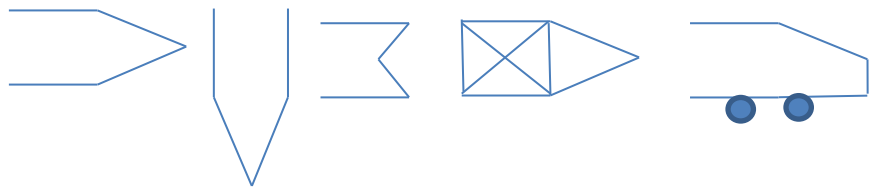
of using this method is that it is easier to change array values instead of changing arguments inside draw line function. Also it is easier to apply different operations like move, resize etc. As we will shortly going to add move method in previous code and do an animation where same object will move:

```
  public void move(int x[], int y[], int dx, int dy){
    for (int i=0;i<x.length;i++){//Here I have written int i to save space, not preferred
      x[i]=x[i]+dx;
      y[i]=y[i]+dy;
    }
  }
  public void paint(Graphics g){
    int x[]={50,100,100,120,100,100,50};
    int y[]={50,50,80,100,120,150,150};
    g.drawPolygon(x,y,7);
    int d=10;
    for ( ;d<=100;d++){
      g.setColor(Color.BLUE);
      g.fillPolygon(x,y,7);
      try{      Thread.sleep(500);   }catch(Exception e){}
      g.setColor(Color.WHITE);
      g.fillPolygon(x,y,7);
      move(x,y,d,d);
    }
}
```

This method is adding dx & dy to all array elements. If array is representing some object this method can move object to a distance passed in both directions.

After pause of 500 milliseconds redrawing object in background color to erase previous object. Than calling move method to move all array elements with some distance. Within loop repeating the process gives effect of animation.

We may not have all objects of type polygon we may have different objects in our drawing like shapes given below:



We may write methods to draw them. The advantage will be to call them again and again with different inputs to draw same object at different locations or similar objects with different dimensions like variation in width or height or direction etc. Such methods require graphics parameter besides other parameter depending upon the shape because we have to use object of graphics class for drawing. Such methods may have signatures like:

```
public void drawHorizontalBullet(Graphics g, int x, int y, int w, int h)
public void drawVerticalBullet(Graphics g, int x, int y, int w, int h)
public void drawVehicle(Graphics g, int x, int y, int w, int h)
public void drawTrain(int x, int y, int n, Graphics g)
```

All methods should have Graphics object as parameter. Above is code to draw train:

```
public void drawTrain(int x, int y, int t, Graphics g){
  int i;
  for (i=1;i<t;i++){
    g.drawRect(x,y,30,20);
```

```
      g.drawLine(x+30,y+10,x+45,y+10);
      g.drawOval(x+5,y+20,5,5);
      g.drawOval(x+20,y+20,5,5);
      x=x+45;
    }
    g.drawRect(x,y,30,20);
    g.drawOval(x+5,y+20,5,5);
    g.drawOval(x+20,y+20,5,5);
}
```

Here x, y is starting position. t is no of bogies. Idea is to draw rectangles and two circles (representing wheels) to show bogies. Line is connection between bogies. Last bogie is drawn outside loop because lines will be one lesser than bogies. Below is the code to call this method and draw train:

```
public void paint(Graphics g){
  drawTrain(20,60,5,g);
}
```

Output shown on right hand side