Introduction to Java Applets

In this module you will be learning how to create Java applets. The differences between a Java application and a Java applet centre around how they are executed, and the restrictions that are placed on them.

A **Java application** is run on a workstation, and can do anything that any other programming language can do. A Java application is saved as a *.java file, and is compiled to create a *.class file.

A **Java applet**, on the other hand, is run from within a browser (like Internet Explorer, or Netscape Navigator), and is limited in some of its functionality - specifically file access. For security reasons, applets cannot access (read or write) files on the computer running them. A Java applet is also saved as a *.java file, and is compiled to create a *.class file. A web page (HTML file) can then be created which will display the applet. The power of creating a Java applet is that you can now add power to a web page that you just can't do with regular HTML (the regular markup language used to create web pages).

While you are programming using **Ready to Program with Java**, you can create applets, and run them all from within the Ready programming environment. After the first exercise, details on how to use your applets in a web page will be given.

Creating a simple applet - text output, graphics output

Let's start small with a simple applet, one that displays text and graphics on the screen.

1. Start off with a new applet by choosing FILE > NEW > APPLET TEMPLATE
2. For the class name, enter **FirstApplet**
3. You will have a shell of a program that looks like this:

```
//    The "FirstApplet" class.
import java.applet.*;
import java.awt.*;

public class FirstApplet extends Applet
{
    // Place instance variables here

    public void init ()
    {
      // Place the body of the initialization method here
    } // init method

    public void paint (Graphics g)
    {
        // Place the body of the drawing method here
    } // paint method

} // FirstApplet class
```

Notes:

- You will notice immediately some differences between this applet and the Java applications you have created previously. There is no **main** method in an applet. If you were to add a main method, it would not act the same way that it would in a Java application.
- There is a method called **init** which is used generally to initialize variables and setup GUI components. This method is called automatically when the applet is run.
- There is a method called **paint** which handles the drawing (or redrawing) routines for the applet.The paint method is called whenever the browser window is re-sized, or when it is first opened, or changed in appearance.Text and graphics are drawn on the applet window using the predefined **Graphics g** object in the paint method.

- There are a few other methods that have special functions in a Java applet, and we will encounter them throughout the module:

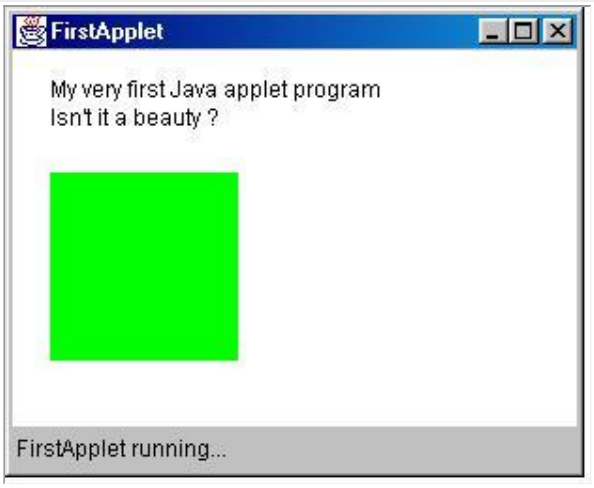
1. **start** method - for more complex initialization features
2. **action** method - to decipher mouse and keyboard input
3. **stop** method - called when the execution of the applet is finished
4. **destroy** method - cleans up after the applet is removed

Making that FirstApplet interesting !

Modify the FirstApplet by replacing the comment line in the **paint** method (// Place the body of the drawing method here) with the following lines:

```
g.drawString("My very first Java applet program", 20, 25);
g.drawString("Isn't it a beauty?", 20, 40);
g.setColor(Color.green);
g.fillRect(20,65,100,100);
```

Run the applet by pressing F1 or clicking on the RUN button. It should appear like the Applet shown here (to the right)



NOTE:

When you run a Java applet from inside the **Ready to Program with Java** environment, you will be asked for the size of the applet. (for now, just accept the default suggestions). The reason for this is that a Java applet, when incorporated into a web page, must be given an area, width and height, so that the internet browser can correctly space out the various elements of the web page - the java applet may only be one portion of the web page. In fact, it is possible to create a web page that has a number of java applets on it, and each one must have a predefined width and height.

The **drawString** commands place text on the output window according to the (x,y) pixel co-ordinates given after the text string. Since the x co-ordinate given in both drawString commands in this example were the same, the output appears to line up evenly on the screen along the left margin. The y-coordinates are 15 pixels apart, which is the standard height for the default browser screen font. It is possible to output in different fonts, the default doesn't have to be used all the time (more on that later).

The **fillRect** command is just one example of a graphics command that works with a Java applet. In fact, almost all graphics drawing commands that were used in the previous module will work with Java applets.

In the previous Java applications, a ***Console*** object was created which accepted all of the text input/output and drawing commands. In a Java applet, the **paint** method has a predefined **Graphics** object associated with it which will handle the test input/output and drawing commands.

Let's Build a House (Using the methods in a Java Applet)

Earlier we wrote Java Applications that used **methods** to execute simple tasks, like drawing an object. We can do the same in a Java Applet.

Start a new file using the Applet Boilerplate, and call the class **MySceneApplet**

In a Java application program, a Console object is created at the beginning of the program, and all graphics commands are attached to it. In a Java applet, the graphics object is not created outright, it is passed, as a parameter, to the **paint** method, which in turn must pass it onto any methods that IT calls if

drawing on the screen is desired. *(That is why in the paint method, you use the statement **drawHouse(g);** instead of just **drawHouse( );**)*

In the actual **drawHouse** method itself, you will have to add the Graphic object to its ***signature line*** (the first line in the method's definition). It should become:

```
public static void drawHouse (Graphics g)
```

So, create a new method called drawHouse, with the signature shown above.

Replace the comment line in the **paint** method (// Place the body of the drawing method here) with the following lines:

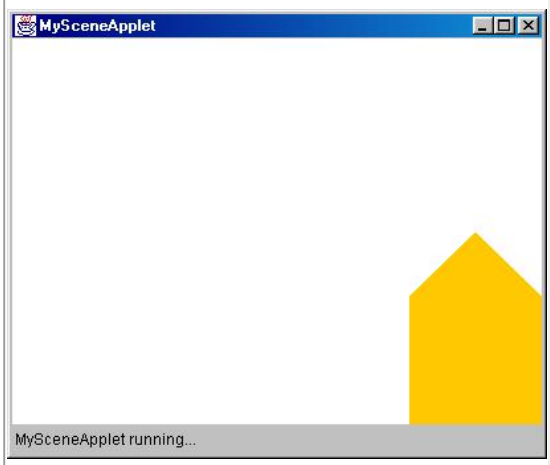
```
drawHouse(g);
```

In the **drawHouse** method, you may use methods from the Graphics class to draw your house. See the Java API for a list of methods in the Graphics class.These commands will work in a similar way to the draw commands in the Ready Console class, but each command will start with a **`g.`** instead of a `c.` (since the drawing commands will be output to the Graphics object called **g**, not the Console object c)

You may also need to change your spelling of certain commands to the Americanized version, like **setColor** for example. Since Java has established standards, and they were set by an American company, certain commands use that spelling convention. *(The Console object is an H.S.A. defined object, a Canadian company, and the Canadian, British and American spellings of most commands are accepted)*

Your java applet should look like this:

```
// The "MySceneApplet" class.
import java.applet.*;
import java.awt.*;

public class MySceneApplet extends Applet

{
    // Place instance variables here

    public void init ()
    {
      // Place the body of the initialization method here
    } // init method

    public void paint (Graphics g)

    {
        drawHouse (g);
    } // paint method

    public static void drawHouse (Graphics g)
    {
        // draw solid base of house
        g.setColor (Color.orange);
        g.fillRect (300, 200, 100, 100);

        // add roof
        Polygon roof;
        roof = new Polygon (); // define a polygon object
        // add points in (x,y) combinations
        roof.addPoint (300, 200);
        roof.addPoint (350, 150);
        roof.addPoint (400, 200);
        g.fillPolygon (roof.xpoints, roof.ypoints,3);
    }

} // MySceneApplet class
```



Run the applet to see if it works.

NOTE: since your house design extends beyond the x=300 pixel position, you should run your applet with a width of 400 and a height of 300, or larger. *(not the default of 300 x 200)*

Note on using different colors in your Applets:

In a java Applet, if you want to draw in a particular color, you use the setColor method of the Graphics class as here:
g.setColor (Color.orange);

But there are a limited number of colours you can call by name. If you would like greater control over the exact shade you are using, you can set a new colour using the RGB (Red-Green-Blue) values. Each of the primary colours (RGB) must have a value between 0 and 255. With three primary colours, each having 256 different possible levels, you have access to a total of 16,777,216 (that's 256x256x256) different possible colour combinations.

To set the colour like this, use the setColor method like this:
g.setColor (new Color (67,115, 200));
The three numbers represent the intensity of each of the three primary colours. Want to find a specific shade? Try the colour picker applet below, and just use the three RGB values you see at the bottom left of the window. Use the decimal number values 0-255, *not the Hex values*.

More information about colours can be found in the Java Class Libraries (look for the Color class).

You can even make some of the colours you use parially transparent by setting their **alpha value**. By adding a fourth number, you can vary the alpha value from 0 (invisible) to 255 (completely solid). take a look at this applet and its code:

```
// The "AlphaColors" class.
import java.applet.*;
import java.awt.*;

public class AlphaColors extends Applet
{
// Place instance variables here

public void init ()
{
// Place the body of the initialization method
here
} // init method

public void paint (Graphics g)
{
// these rectangles are "solid" colours
g.setColor (new Color (0, 0, 0));
g.fillRect (150, 0, 300, 400);
g.setColor (new Color (0, 255, 255));
g.fillRect (60, 0, 25, 400);
g.setColor (new Color (255, 255, 0));
g.fillRect (215, 0, 25, 400);

// this circle is "solid" red
g.setColor (new Color (255, 0, 0));
```

```
g.fillOval (50, 0, 200, 200);

// this circle is "transparent" green
g.setColor (new Color (0, 255, 0, 150));
g.fillOval (50, 100, 200, 200);

// this circle is "transparent" blue
g.setColor (new Color (0, 0, 255, 175));
g.fillOval (50, 200, 200, 200);

// Place the body of the drawing method here
} // paint method
} // AlphaColors class
```

Exercise Applet 1
Complete the Java Applet called **MySceneApplet** by following the instructions here:

Create methods drawHouse, drawSun, drawFlower and drawTree that draw the appropriate objects. Create at least two other methods of your own invention to draw other objects.

Create a **paint** method which will make calls to the drawHouse method, and the other methods you have created, so the entire scene is drawn on screen.

Remember that

1. Graphics commands start with g , not c
2. The paint method must have the parameter **"Graphics g"**
3. Change spelling from setColour to setColor (Java uses American spelling!)

When running applet - change default width and height to 500 x 400 (or bigger) so all of scene is visible