CS101
Input, I/O via LabPkg Viewframe, Intro to Graphics

Input using Native Java

So far we have "hard coded" all values we want directly in our code. For example, in the race pace calculation, we put the pace and distance directly as variables in our program. It would be nice to be able to run our program, have it ask the user to enter what pace and distance we want, let the user enter those values, and then calculate the time. This makes the program much more general and functional with easily-changed values instead of having to change the values in the program, recompile it, and then re-run the program.

Unfortunately, input in Java is a bit unwieldy (this has been one of its criticisms). All data that is input from the keyboard must be read as a String. Recall that a String is a sequence of ASCII characters. If we want to read a number, such as an integer, then we must convert the String to an integer. For example, this means converting from ASCII to two's complement.

Here is an example of a program that shows how to read a string in Java:

```
import java.io.*;
class InputTest {
      public static void main(String[] argv) throws Exception {
                String sUserInput;
                BufferedReader inFromUser = new BufferedReader(
                              new InputStreamReader(System.in));

                System.out.println("Enter something: ");
                sUserInput = inFromUser.readLine();
                System.out.println("You entered: " + sUserInput);
      }
}
```

In the first line, we are importing the java.io package. This package includes code routines we will need that read data from the keyboard.

In the definition of main, we added "throws Exception" before the body of the method. An **exception** occurs when something unexpected happens, typically an error. For example, if data could not be read, then an exception may occur. Another situation that would cause an exception is behavior like trying to divide a number by zero. This is undefined (or infinity in some cases). By adding this statement, Java will notify us if an exception occurs.

The variable sUserInput is a string variable that will hold the contents of what the user types at the keyboard.

InFromUser is a variable of type BufferedReader. Don't worry about what this line does, it essentially defines an object that is capable of reading input from the keyboard (System.in). However, you might be wondering why sometimes we use the keyword

new and sometimes we don't. The answer is that whenever we want to create an Object or Class, we must use *new* to create an instance of that class. By invoking *new*, we are allocating memory and initializing the object. However, whenever we are declaring a simple variable (type int, float, char, double, etc.) we don't need to worry about initializing or allocating any extra memory and therefore don't need to use *new*.

The line : sUserInput = inFromUser.readLine(); waits for the user to type data at the keyboard. The data entered by the user is stored in ASCII format in the variable sUserInput.

The last line simply prints out the contents of this variable.

Reading Values Other Than Strings

The above code shows how to read data into strings. What if we want to read data into other types of variables, such as integers or doubles or floats? The answer is that we must read the data as a string, and then convert it to the respective type of interest. This is not as simple as typecasting the string, because we need much more logic to translate a string of characters to a 2's complement or IEEE 754 value (float or double).

For example, Java will complain if you try the following:

```
import java.io.*;
class InputTest {
      public static void main(String[] argv) throws Exception {
                String sUserInput;
                int i;
                BufferedReader inFromUser = new BufferedReader(
                              new InputStreamReader(System.in));

                System.out.println("Enter an integer: ");
                sUserInput = inFromUser.readLine();
                i = (int) sUserInput;
                System.out.println("You entered: " + i);
      }
}
```

This results in an error message while compiling, complaining that i requires an integer be assigned to it.

Fortunately, Java has already defined code that does this conversion for us. Examples are shown below for ints, floats, and doubles:

```
import java.io.*;

class InputTest {
      public static void main(String[] argv) throws Exception {
                String sUserInput;
```

```
            int i;
            float f;
            double d;
            BufferedReader inFromUser = new BufferedReader(
                        new InputStreamReader(System.in));

            System.out.println("Enter an integer. ");
            sUserInput = inFromUser.readLine();
            i = Integer.valueOf(sUserInput ).intValue();
            System.out.println("You entered: " + i);

            System.out.println("Enter a double. ");
            sUserInput = inFromUser.readLine();
            d = Double.valueOf(sUserInput).doubleValue();
            System.out.println("You entered: " + d);

            System.out.println("Enter a float. ");
            sUserInput = inFromUser.readLine();
            f = Float.valueOf(sUserInput).floatValue();
            System.out.println("You entered: " + f);

            // Now sum up all three things entered as a double
            d = d + (double) f + (double) i;
            System.out.println("The sum of all three is: " + d);
      }
}
```

Things to note: The line

Integer.valueOf(sUserInput).intValue();

is case-sensitive! In particular, notice the capital I on Integer. This is necessary, because Integer is different from int. Integer refers to an object defined in Java that deals with integers. One of the things this object can do is convert from String to the Integer object. The function valueOf() does this. However, we then have to convert from the Integer object to the actual int value (i.e., the 2's complement value). The function intValue() does this conversion.

There is a similar behavior for doubles and floats. Double refers to the Java object that deals with doubles, while Float refers to the Java object that deals with floats. We could do the same thing with char's and long's by creating a Char object or a Long object and then reading charValue() or longValue().

Here is the output for a sample run:

```
Enter an integer.
5
You entered: 5
Enter a double.
2.412
You entered: 2.412
Enter a float.
3.14
You entered: 3.14                    // Note roundoff below!
The sum of all three is: 10.552000104904174
```

Using LabPkg for Input and Output

Performing input and output with what Java provides directly can be a bit cumbersome as seen above. Fortunately, the authors of the "Basic Java Programming" textbook have provided a package you can use that simplifies this process. They have written code to make it easy to display graphical windows and input data. This code is contained in the LabPkg file distributed with the textbook.

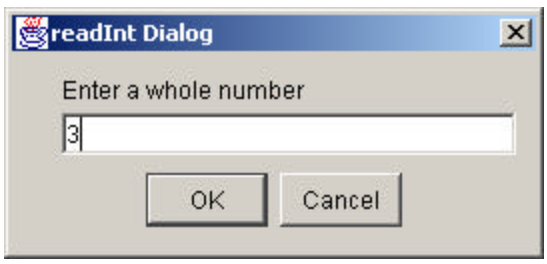
The objects we will interact with using the LabPkg for now involve the **ViewFrame** object. Listed below is an example illustrating how it works. **Although we will use LabPkg in this class, please note that this is not a part of the standard Java distribution**. That means these functions won't be available on any system that doesn't have the LabPkg installed (i.e. most systems).

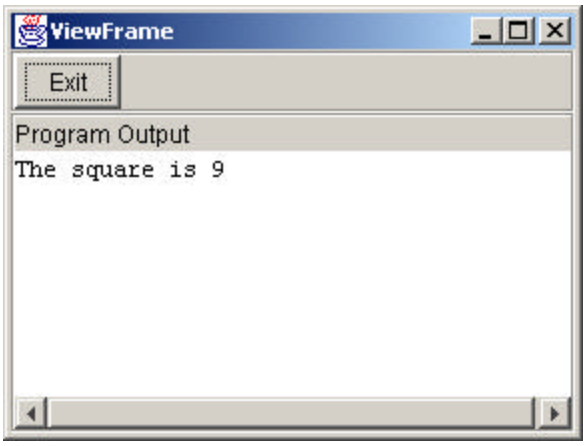
```
import LabPkg.*;
public class Square {
      public static void main(String[] args) {
            ViewFrame vf = new ViewFrame();
            int n;

            vf.setVisible(true);
            n = vf.readInt("Enter a whole number");
            vf.println("The square is " + n*n);
      }
}
```

The import statement tells Java that we want to use objects defined in LabPkg. The object we use is the ViewFrame class. In this case, we create a new ViewFrame in the variable *vf*. The ViewFrame class creates a graphical window that we can use to input and output data. Before we can do that though, we must make the window visible. This is done by invoking the method setVisible and giving it the value "true" to mean the window is now visible.

To input an integer, we invoke the method readInt. This method displays a dialog box in which the user can enter an integer. ReadInt takes a single parameter, which is the message we will display in the dialog box.

Finally, to print the integer, we use the println method. This works just like System.out.println, except instead of System.out we use the viewframe we want to output our text to instead. Here is sample output from this program:





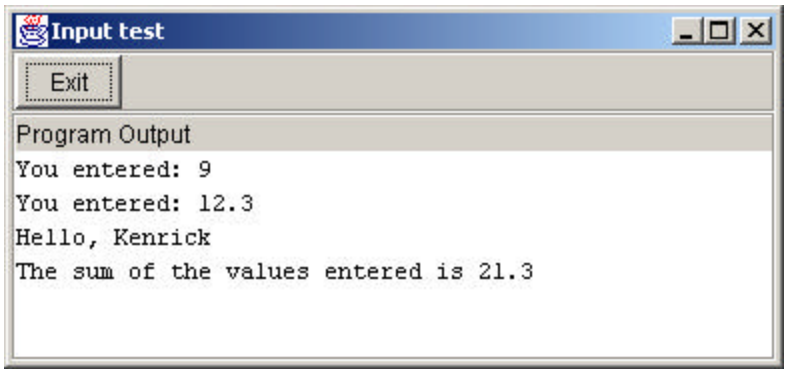
Here is another version that builds on this simple example. Can you tell what it will do?

```
import LabPkg.*;
public class Square {
      public static void main(String[] args) {
            ViewFrame vf1 = new ViewFrame(“square”);  // Assigns title
            ViewFrame vf2 = new ViewFrame(“cube”);
            int n;

            vf1.setVisible(true);
            vf2.setVisible(true);
            n = vf1.readInt("Enter a whole number");
            vf1.println("The square is " + n*n);
            vf2.println("The cube is " + n*n*n);
      }
}
```

What if we want to read values other than integers? The ViewFrame object allows us to also directly input Strings and Doubles (but not Floats or any other simple data type). An example is given below:

```
import LabPkg.*;
public class LabPkgTest {
      public static void main(String[] args) {
            ViewFrame vf1 = new ViewFrame("Input test");
            int i;
            double d;
            String s;

            vf1.setVisible(true);
            i = vf1.readInt("Enter a integer");
            vf1.println("You entered: " + i);

            d = vf1.readDouble("Enter a double");
            vf1.println("You entered: " + d);

            s = vf1.readString("Enter your name");
            vf1.println("Hello, " + s);

            d = d+ (double) i;   // try summing up the values
            vf1.println("The sum of the values entered is " + d);
      }
}
```

This new program will prompt the user for three types, for an integer, double, and a string. The output is all sent to the viewframe graphical window:

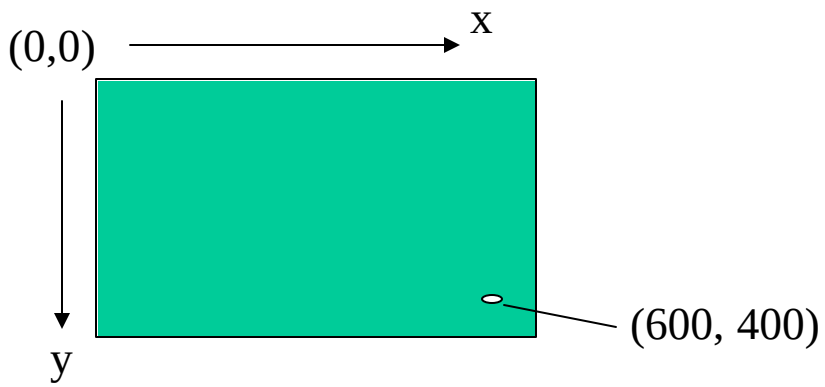


Documentation of the viewframe window is scattered throughout the first few chapters of the book and is all condensed in one place on the floppy disk that came with the lab book. It is contained in the docs directory.

Drawing Graphics

We can use a *Canvas* object within the ViewFrame object to draw graphics. This is described on page 48 of the lab book. As the name implies, the Canvas object behaves like a blank painting. We are then able to draw items on the canvas.

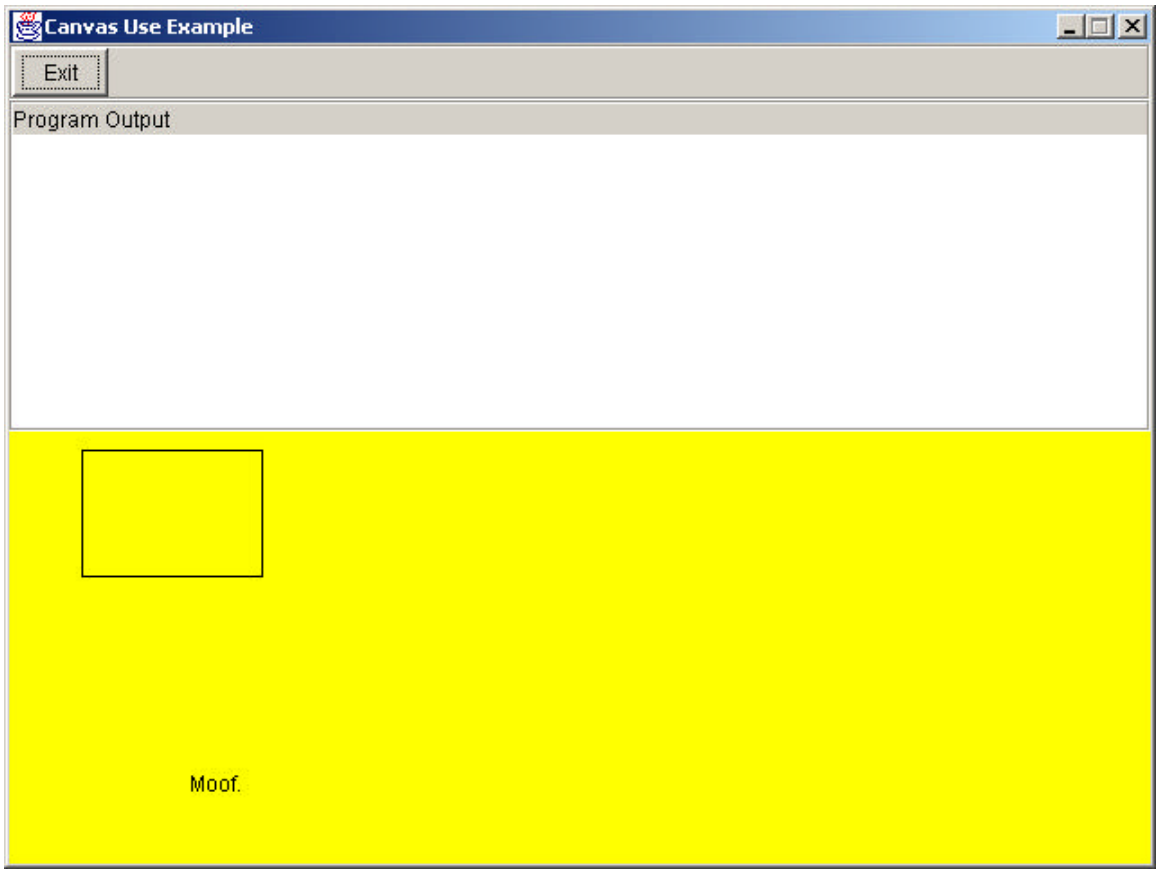
The graphics screen is set up as a grid of pixels where the upper left coordinate is 0,0. This is relative to where the canvas is on the viewframe. The x coordinate then grows out to the right, and the y coordinate grows down toward the bottom.



Here is some sample code that illustrates one way to draw graphics (there is a more common and also more powerful way, which is extending the frame, we will see later).

```
import java.awt.*;
import LabPkg.*;

class DrawExample {
      public static void main(String[] args) {
            ViewFrame vf = new ViewFrame("Canvas Use Example");
            vf.setVisible(true);
            vf.setResizable(false);  // Don't allow window resizing

            Canvas c = new Canvas();
            // Set its size or the size will be 0 by 0
            c.setSize(ViewFrame.WIDTH, ViewFrame.HEIGHT / 2);
            c.setBackground(Color.yellow);
            vf.setCanvas(c);  // Add canvas to ViewFrame

            Useful.pause(20); // Wait 2 seconds for frame to be visible
            Graphics g = c.getGraphics();
            g.drawRect(40,10,100,70);
            g.drawString("Moof.", 100, 200);
      }
}
```

This code produces the following image:



In this code, we are importing java.awt.* to get access to some of the graphics functions. After setting the ViewFrame to be visible, we are setting it to disallow resizing by the

user. To allow resizing we'll need to implement this differently if we want the image to be drawn correctly after resizing (there is a paint() method that is automatically invoked when a frame is resized, and we'd have to put the drawing code in this method so that it redraws the canvas correctly. Otherwise we'd just get empty white stuff in the resized areas).

When we create a ViewFrame, by default it is about 640 pixels wide by 480 pixels tall.

Next, we create a *Canvas* object and add it to the ViewFrame. We have to tell Java how big we want the canvas object to be. In this case, we make it the full width of the ViewFrame, but only half its height. By default, the canvas is added at the bottom of the ViewFrame. Also note that vf.WIDTH and vf.HEIGHT includes pixels that make up the border and title bar. To show where the canvas is on the ViewFrame, we set its background color to yellow.

Next, we invoke Useful.pause(20), which is another LabPkg class. This method makes the program pause for 2 seconds. The input parameter is the number of tenths of seconds to pause. This ensures that the ViewFrame has been resized and made visible by the time we start to draw, otherwise our drawing wouldn't show up correctly.

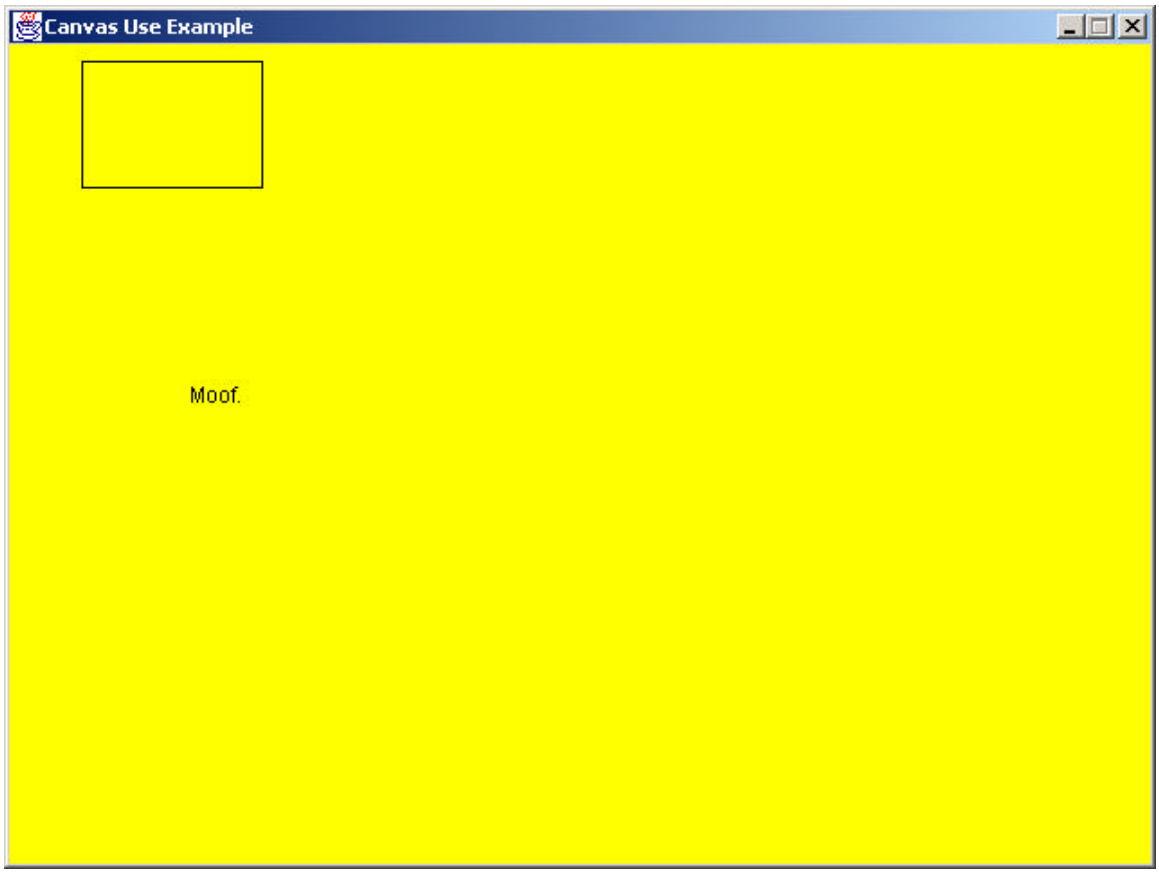
Finally, we access the *graphics* portion of the canvas via c.getGraphics(). This is what finally allows us to draw shapes.

The Graphics class supports many drawing functions. Here we are using drawRect and drawString. drawRect(x1,y1,width,height) draws a rectangle with the upper left corner at (x1,y1) and the specified width and height. drawString(string, x1, y1) draws the ASCII text string at coordinates x1, y1.

If we wanted to use the entire ViewFrame for the canvas, we could use:

```
import java.awt.*;
import LabPkg.*;
class DrawExample {
      public static void main(String[] args) {
            ViewFrame vf = new ViewFrame("Canvas Use Example");
            vf.setVisible(true);
            vf.setResizable(false);  // Don't allow window resizing

            Canvas c = new Canvas();
            // Set its size or the size will be 0 by 0
            c.setSize(ViewFrame.WIDTH, ViewFrame.HEIGHT - 24);
            c.setBackground(Color.yellow);
            vf.setCanvas(c);  // Add canvas to ViewFrame

            Useful.pause(20); // Wait 2 seconds for frame to be visible
            Graphics g = c.getGraphics();
            g.drawRect(40,10,100,70);
            g.drawString("Moof.", 100, 200);
      }
}
```

In this case, we used a canvas height of ViewFrame.HEIGHT – 24 to subtract off the pixels used for the title bar. This produces:



Note that the drawing is done relative to where the canvas is placed on the frame, not relative to the entire window.

Colors

Here are different colors available using the color class:

Color.black	Color.darkGray	Color.gray
Color.blue	Color.green	Color.lightGray
Color.cyan	Color.magenta	Color.orange
Color.pink	Color.red	Color.white
Color.yellow		

To create our own color, we can specify the color we want in RGB (red, green, blue). To do this, use:

```
new Color(redvalue, greenvalue, bluevalue);
```

where each value is in the range 0-255.

For example:

new Color(0,255,200);

Creates a green-blue color, with stronger green than blue.

More drawing functions

Here is a list of other drawing functions you can use with the graphics object:

setColor(color)
Sets the current color to a color defined as above
drawLine(int x1, int y1, int x2, int y2)
Draws a line in the current color from x1,y1 to x2,y2
Can use x2=x1 and y2=y1 to draw a single pixel
drawOval(int x, int y, int width, int height)
Draws an oval in the current color within a box at upper left corner x,y
With specified width and height in pixels
Use width = height to draw a circle
drawRect(int x, int y, int width, int height)
Draws a rectangle in the current color with upper left corner at x,y
With specified width and height in pixels
drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)
Draws an arc in the current color at the coordinates and specified angle
fillOval(int x, int y, int width, int height)
Draws an oval in the current color within a box at upper left corner x,y
With specified width and height in pixels
Use width = height to draw a circle
fillRect(int x, int y, int width, int height)
Fills a rectangle in the current color with upper left corner at x,y
With specified width and height in pixels
fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)
Fills an arc in the current color at the coordinates and specified angle

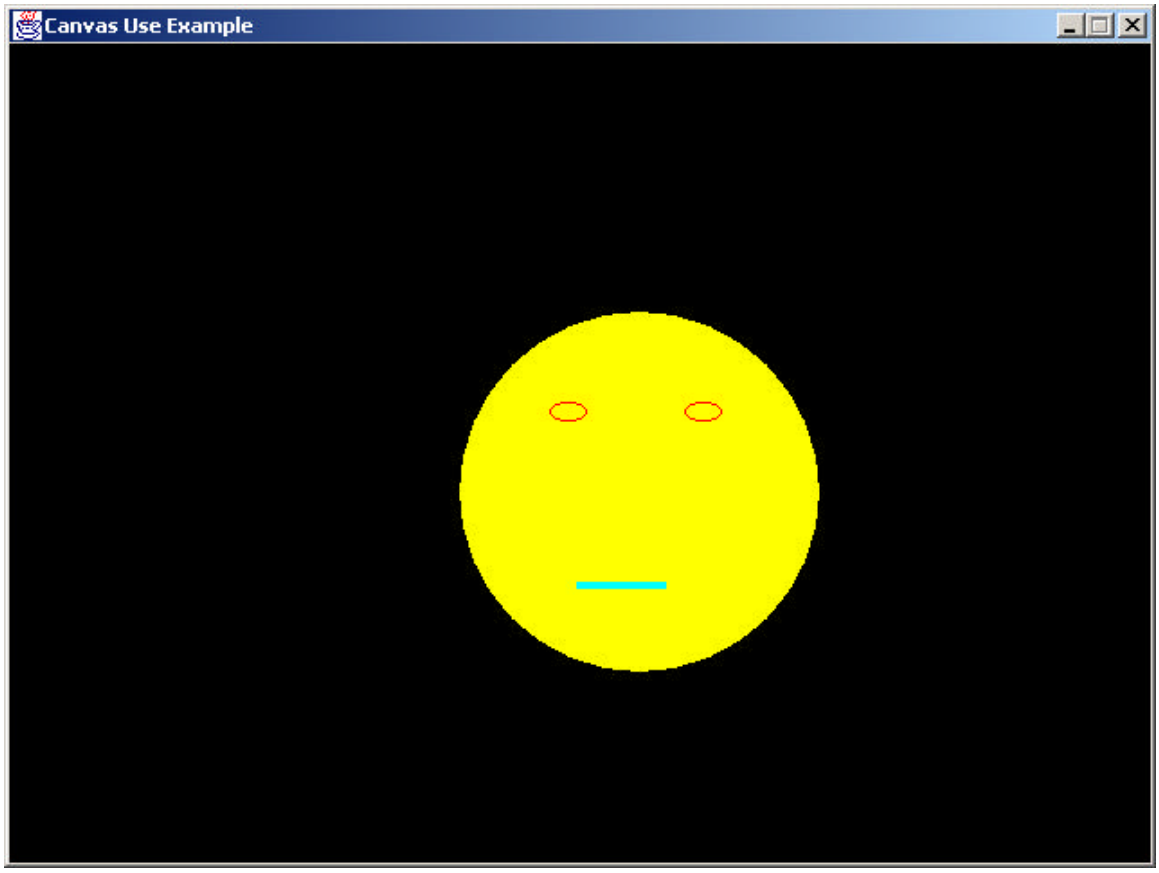
There are many more, for a full list see
http://java.sun.com/j2se/1.3/docs/api/java/awt/Graphics.html

Here is an example using some of these functions to draw an ambivalent yellow face:

```
import java.awt.*;
import LabPkg.*;

class DrawExample {
      public static void main(String[] args) {
            ViewFrame vf = new ViewFrame("Canvas Use Example");
            vf.setVisible(true);
            vf.setResizable(false);
```

```
            Canvas c = new Canvas();
            c.setSize(ViewFrame.WIDTH, ViewFrame.HEIGHT - 24);
            c.setBackground(Color.black);
            vf.setCanvas(c);

            Useful.pause(20);
            Graphics g = c.getGraphics();
            g.setColor(Color.yellow);
            g.fillOval(250,150,200,200);
            g.setColor(new Color(255,0,0));     // Red eyes
            g.drawOval(300,200,20,10);
            g.drawOval(375,200,20,10);
            g.setColor(new Color(0,255,255));   // Green-Blue mouth
            g.fillRect(315,300,50,4);
      }
}
```

This program draws the following image:



Can you figure out what the following code would do?

In file DrawFace.java:

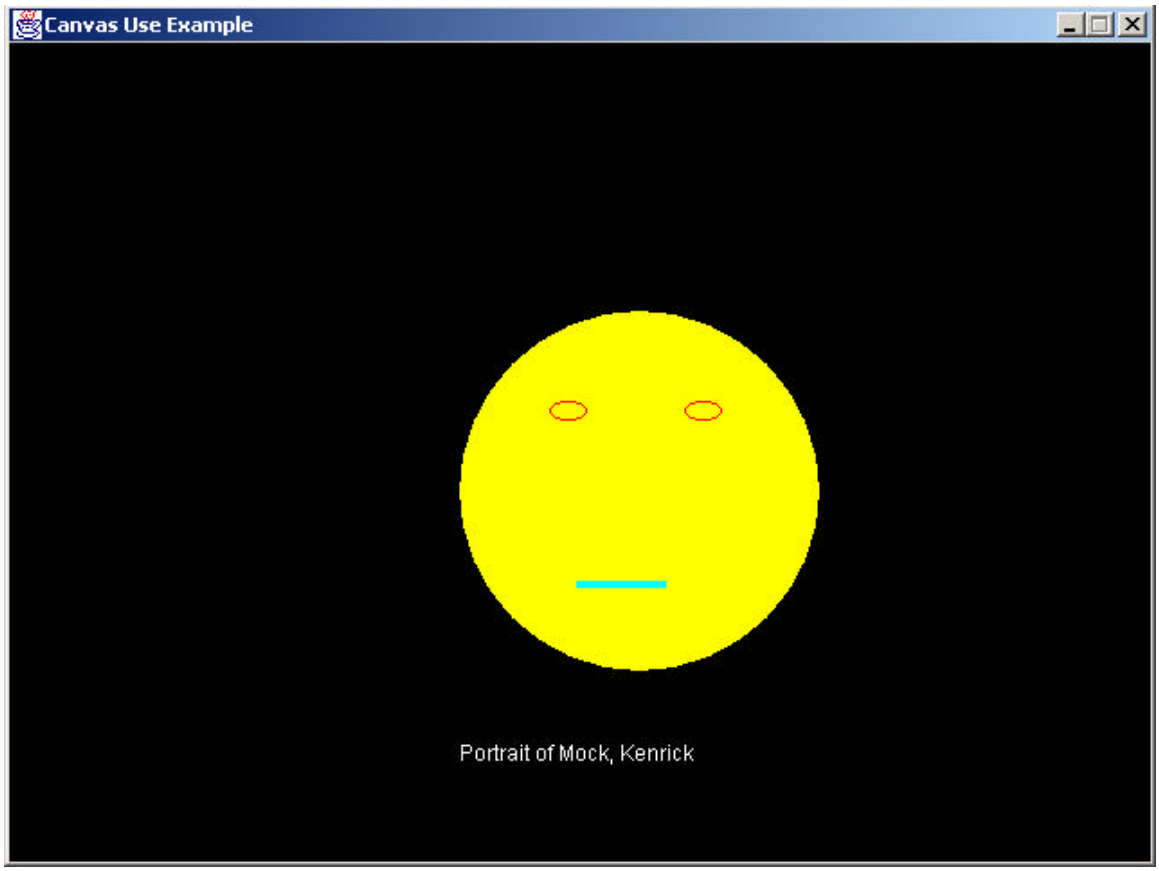
```
import java.awt.*;
import LabPkg.*;
class DrawFace {
      public void ShowPicture(String sCaption) {
            ViewFrame vf = new ViewFrame("Canvas Use Example");
            vf.setVisible(true);
            vf.setResizable(false);

            Canvas c = new Canvas();
            c.setSize(ViewFrame.WIDTH, ViewFrame.HEIGHT - 24);
            c.setBackground(Color.black);
            vf.setCanvas(c);

            Useful.pause(20);
            Graphics g = c.getGraphics();
            g.setColor(Color.yellow);
            g.fillOval(250,150,200,200);
            g.setColor(new Color(255,0,0));     // Red eyes
            g.drawOval(300,200,20,10);
            g.drawOval(375,200,20,10);
            g.setColor(new Color(0,255,255));   // Green-Blue mouth
            g.fillRect(315,300,50,4);
            g.setColor(Color.white);            // Add caption in white
            g.drawString(sCaption,250,400);
      }
}
```

In file UseFace.java:

```
class UseFace {
 public static void main(String[] args) {
      DrawFace face1, face2;

      face1 = new DrawFace();
      face2 = new DrawFace();
      face1.ShowPicture("Portrait of Mock, Kenrick");
      face2.ShowPicture("Portrait of Attrick, Jerry");
 }
}
```

This program will create two instances of the DrawFace object and display them. However, we've added a parameter to the ShowPicture method that accepts a string. We use this string as a caption, and it will vary for the two pictures. We will have much more to say about how we pass parameters around in the future. For now, just recall that we can create objects and this helps promote code reuse if we only have a few things different with respect to data that will be processed by the separate objects. In this example, the only thing that changes is the caption placed below the picture.

Output for one of these windows is shown below:



For a full list of all of the Java class documentation online, see
http://java.sun.com/j2se/1.3/docs/api/index.html