CSC 1051 - Lab 15

This lab is optional. It can be used to improve your grade by replacing a bad project or lab grade.

Objectives:
Learn about animated applets.

Assignment:
We will start with a basic framework for an animated Applet and add more features to it to gain an understanding for how each piece of it works.

```java
// Animation.java  * your name here *
// A simple animated applet.
// *******************************************************
import java.awt.*;
import java.applet.*;

public class Animation extends Applet implements Runnable {
    private Thread runner; // a process (runs the animation)
    private int xPos = 500;
    private int yPos = 500;

    // initializes applet
    public void start() {
        if (runner == null) // create the thread
            { runner = new Thread (this);
                runner.start();
            }
    }

    // stops applet
    public void stop() {
        if (runner != null) // when done, destroy the thread
            { runner = null;
                }
    }

```
// run animation
public void run()
{
    while (true) // infinite loop
    {
        repaint();
        try
        {
            Thread.sleep(10); // 10 milliseconds wait
        } // before repainting
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
    }
    // don’t need to understand all of this...
}

// paints (or repains) the applet
public void paint(Graphics g)
{
    g.fillOval( xPos, yPos, 20, 20); // a 20X20 circle
    xPos --;
    if (xPos < 10) // wrap around
        xPos = 500;
}

Note that an animated applet, in addition to extending the Applet class (extends Applet) also implements the Runnable interface, so you need to add
Implements Runnable to the class declaration heading. Interfaces and
threads that this program uses are beyond the scope or our course, but it is easy
to use the code in this example, to make some pretty nifty animations, by just
focusing on the parts of the code that you do understand.

1. Type in the code into a Java file. Compile and run it.

2. Make the following changes and observe their effect:
   a) Change the wait time to 30 milliseconds

   b) Change it so that the circle starts on the left and moves to the right.
      (Hint: you will need to change the xPos variable in the initialization and the
      paint() method.

   c) In the start() method, add some code to customize your applet. For
      example: try something like:
      
      setBackground(Color.red);
      resize(800,300);
d) Change the color of the circle (Hint: this is done in the paint() method).

3. Create a method move() for moving the circle to get it positioned for its next step in the animation and use this method to replace the code that changes xpos in the paint() method (use move() instead).

4. Now create another method randomMove() for creating a small random shift in the position of the circle. Experiment with replacing move() with randomMove() in the paint() method.

5. More things to try:
   a) Replace the circle (which is hardcoded in this example) with a Circle object or some other graphical object. That graphical object should have a position and a move() method that can be invoked through the object.
   b) Use an array of circles or other objects to create an animation with several things moving at the same time.