C212/A592 Spring 17 Lab8

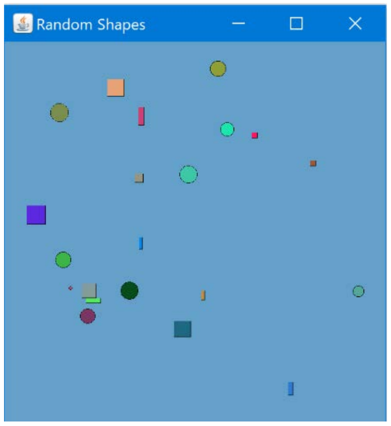
Intro to Software Systems

Instructions:

- Review the requirements given below and complete your work. Please submit all files through Canvas.
- The grading scheme is provided on Canvas

Objectives

- The goals of this lab are to review some very important OOP concepts that come up in job interviews:
    - inheritance
    - method overloading, overriding
    - constructor chaining
    - polymorphism
    - abstract classes



Lab 8 Random Shape Generator

Part 1 – Introducing **abstract Shape** class

- Using your shape class
    - Add an *abstract draw(Graphics g)* method to your abstract Shape class
- Create a Rectangle, Ellipse, and Triangle class that are all extended from an abstract Shape class (given below). Extend the Rectangle class to make a Square class. Extend the Ellipse class to make a Circle class. The shape classes will contain state information for the object to be drawn
    - All the different shapes will need a *draw* method but each one will be slightly different
    - All extended classes should also implement a `toString` method – it is up to you how you want to represent the Shape as a String
    - All extended classes should also implement a `equals` method to compare two shapes
    - The Circle, Ellipse, Triangle, Square, and Rectangle should have the same number of constructors with the same arguments, **in addition to** any other fields that are needed to create those shapes
        - To initialize the private instance fields of the Shape class you will need to make calls to the super class constructor. Private fields and methods of a super class are not directly accessible in the subclasses as if you were in the same class
        - Then initialize fields specific to the particular Shape class you are implementing

- If done correctly the Square classes will have the least amount of code because it can reuse most of the Rectangle class methods. Similarly Circle class should reuse code from Ellipse class.
    - Hint: use super class constructors correctly and all this class should have is constructors and a `toString` method

- Using your classes, implement the Random Shape Generator application
    - Pressing the c, e, r, s, or t key draws a circle, ellipse, rectangle, square or a triangle respectively.

- Other requirements are:
    - Shapes should be of random sizes
    - Shapes should be of random colors
    - Shapes should be in random locations
    - Your background should be a different color than white
    - Does not matter if shapes overlap a bit!

- **Note**: Some of the methods will not be used now and we will come back to them to add more functionality. The way I designed this lab might feel somewhat contrived, which was intentional to emphasize the concepts.
- Below is the API for the shape class:
    - all methods that are not abstract are common to all shapes
    - the abstract methods are methods all shapes will have, but will be different for each shape

```
// we have not covered awt graphcis yet, but you do not need to know much for this lab
// we will be covering that next week
import java.awt.Color;

abstract class Shape {

  private Color fillColor;
  private Color borderColor;
  private Boolean isFilled;
  private Point Location;

  // the three constructors initialize the instance fields
  public Shape(Color fillColor, borderColor, int x, int y) { }

  // set borderColor to Black since not provided
  public Shape(Color fillColor, int x, int y) { }

  // set fillColor to white and border color to black
  public Shape(int x, int y) { }

  public void setFillColor(Color c) { }
  public Color getFillColor() { }

  public void setBorderColor(Color c) { }
```

```
    public Color getBorderColor() { }

   public void setLocation(Point pt) { }
   public Point getLocation() { }

    // Note:  subclasses of Shape do not inherent private members so we need methods the subclasses
    // can use to get the x and y values from the private Point instance field
    public int getX() { }
    public void setX(int x) { }
    public int getY() { }
    public void setY(int y) { }

    // if fillColor is white returns true, else returns false
    public boolean isFilled() { }

    // moves location by dx and dy
    private void moveLocation(int dx, int dy) { }

    abstract double getArea();
    abstract double getPerimeter();
}
```