Programming Assignment 9:

DUE: Before 11:59pm Thursday, March 14

Follow all the same style and commenting guidelines as expressed in previous programming assignments.

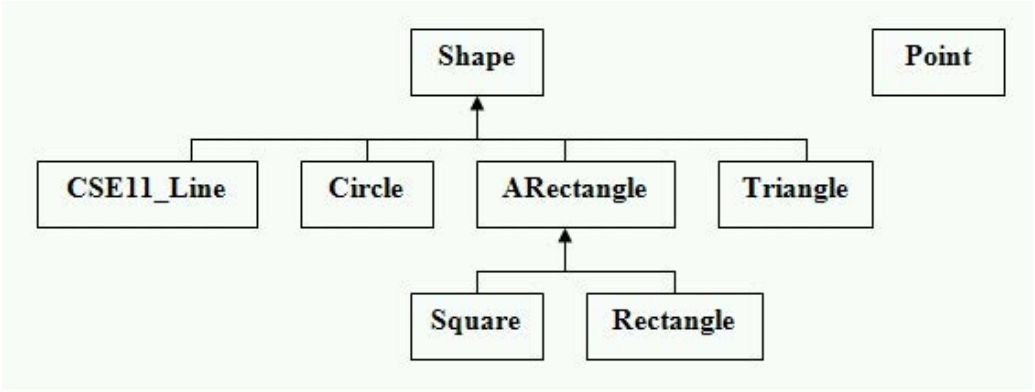
All files related to this HW need to be in a directory named pa9 in your cs11 course-specific account on ieng6.ucsd.edu.

README

You are required to provide a text file named README, NOT Readme.txt, README.pdf, or README.doc, with your assignment in your pa1 directory. There should be no file extension after the file name README. Your README should include the following sections: Program Description: Explain how to work your program. What sort of inputs or events does it take and what are the expected outputs or results? If you need to make a decision how to implement something one way or another, explain how and why you chose to implement it that way?

Shape Hierarchy

Write a set of classes to implement a simple hierarchy of Shapes as follows:




```
import java.awt.*;
import objectdraw.*;

public abstract class Shape
  private String name;

  public Shape() { ... }
  public Shape( String name ) { ... }
  public String getName() { ... }
  private void setName( String name ) { ... }
  public abstract void move( int xDelta, int yDelta );
  public abstract void draw( DrawingCanvas canvas, Color c, boolean fill );


import java.awt.*;
import objectdraw.*;

public class CSE11_Line extends Shape
  private Point start;
  private Point end;

  public CSE11_Line() { ... }
  public CSE11_Line( Point start, Point end ) { ... }
  public CSE11_Line( CSE11_Line line ) { ... }
  public void move( int xDelta, int yDelta ) { ... }
  public String toString() { ... }
```

```
  public boolean equals( Object o ) { ... }
  public int hashCode() { ... }
  public void draw( DrawingCanvas canvas, Color c, boolean fill ) { ... }
and appropriate public get/accessor methods - for example: public Point getStart() ...
and private set/mutator methods - for example: private void setStart( Point p ) ...


import java.awt.*;
import objectdraw.*;

public class Circle extends Shape
  private Point center;
  private int radius;

  public Circle() { ... }
  public Circle( Point center, int radius ) { ... }
  public Circle( Circle c ) { ... }
  public void move( int xDelta, int yDelta ) { ... }
  public String toString() { ... }
  public boolean equals( Object o ) { ... }
  public int hashCode() { ... }
  public void draw( DrawingCanvas canvas, Color c, boolean fill ) { ... }
and appropriate get/accessor methods - for example: public Point getCenter() ...
and private set/mutator methods - for example: private void setCenter( Point center ) ...


public abstract class ARectangle extends Shape
  private Point upperLeft;      // Upper left corner - Common to all rectangular shapes

  public ARectangle() { ... }
  public ARectangle( String name, int x, int y ) { ... }
  public ARectangle( String name, Point upperLeft ) { ... }
  public ARectangle( ARectangle r ) { ... }
  public void move( int xDelta, int yDelta ) { ... }
  public String toString() {  ... }             // getName() + upperLeft
  public boolean equals( Object o ) { ... }     // std checks + upperLeft
  public int hashCode() { ... }
and appropriate get/accessor method - for example: public Point getUpperLeft() ...
and private set/mutator method - for example: private void setUpperLeft( Point p ) ...


import java.awt.*;
import objectdraw.*;

public class Rectangle extends ARectangle
  private int width;
  private int height;

  public Rectangle() { ... }
  public Rectangle(int x, int y, int width, int height) { ... }
  public Rectangle(Point upperLeft, int width, int height) { ... }
  public Rectangle(Rectangle r) { ... }
  public String toString() { ... }        // super.toString() + width + height
  public boolean equals(Object o) { ... } // super.equals() + width + height
  public void draw( DrawingCanvas canvas, Color c, boolean fill ) { ... }
and appropriate get/accessor methods - for example: public int getWidth() ...
and private set/mutator methods - for example: private void setWidth( int w ) ...

NOTE: upperLeft Point, move(), and hashCode() inherited from ARectangle


import java.awt.*;
```

```
import objectdraw.*;

public class Square extends ARectangle
  private int side;

  public Square() { ... }
  public Square( int x, int y, int side ) { ... }
  public Square( Point upperLeft, int side ) { ... }
  public Square( Square r ) { ... }
  public String toString() { ... }             // super.toString() + side
  public boolean equals( Object o ) { ... }    // super.equals() + side
  public void draw( DrawingCanvas canvas, Color c, boolean fill ) { ... }
and appropriate get/accessor methods - for example: public int getSide() ...
and private set/mutator methods - for example: private void setSide( int side ) ...

NOTE: upperLeft Point, move(), and hashCode() inherited from ARectangle


import java.awt.*;
import objectdraw.*;

public class Triangle extends Shape
  private Point p1;
  private Point p2;
  private Point p3;

  public Triangle() { ... }
  public Triangle( Point p1, Point p2, Point p3 ) { ... }
  public Triangle( Triangle tri ) { ... }
  public void move( int xDelta, int yDelta ) { ... }
  public String toString() { ... }
  public boolean equals( Object o ) { ... }
  public int hashCode() { ... }
  public void draw( DrawingCanvas canvas, Color c, boolean fill ) { ... }
and appropriate get/accessor methods - for example: public Point getP1() ...
and private set/mutator methods - for example: private void setP1( Point p1 ) ...
```

---------- Not part of the Shape hierarchy ----------

```
public class Point
  private int x;
  private int y;

  public Point(){ ... }
  public Point( int x, int y ) { ... }
  public Point( Point point ) { ... }
  public int getX() { ... }
  public int getY() { ... }
  private void setX( int x ) { ... }  // Private! so Points are immutable
  private void setY( int y ) { ... }  // Private! so Points are immutable
  public void move( int xDelta, int yDelta ) { ... }
  public String toString() { ... }
  public boolean equals( Object o ) { ... }
  public int hashCode() { ... }
```

Because the objectdraw library already has a Line type, we name our Line type CSE11_Line as to not confuse/conflict with the objectdraw library's Line. The draw() method in our CSE11_Line will use the objectdraw library's Line type.

Most of the above constructors and methods are self-explanatory. The move() method adjusts the shape xDelta pixels in the X direction and yDelta pixels in the Y coordinate. Just add these deltas to the shape's current X and Y location depending on how that shape's location is represented -- CSE11_Line: both start and end Points; Circle: center Point; Rectangle: upperLeftCorner Point; Triangle: all 3 Points.

All accessing of private data in each class must be made through the appropriate get/accessor and set/mutator methods; do not directly access data, including constructors. The only place where direct access is allowed is in the actual accessor/mutator methods.

The draw() method should create the appropriate objectdraw library object to draw on the canvas parameter. The boolean parameter fill indicates whether the graphical object should be filled or not (for example, for Circle whether a FilledOval or a FramedOval should be created). This has no meaning in CSE11_Line. If the Color parameter is null, use Color.BLACK (default).

Copy constructors copy the instance variables from an existing object of the same type to this newly created object. If the variable is a primitive data type, just copy the value of this primitive type from the existing object to this object (assignment through mutator method). If the variable is a reference to an object, create a new copy of the object this variable is referencing (by invoking that variable's copy ctor) and assign this resulting new copy of the object to this object's instance variable. This will result in a deep copy.

For example, in class Center's copy ctor,

```
public Circle( Circle c )
```

to set the center instance variable properly to a new Point based on the parameter's center Point:

```
    this.setCenter( new Point( c.getCenter() ) );
```

Test files are provided in ~/../public/PA9

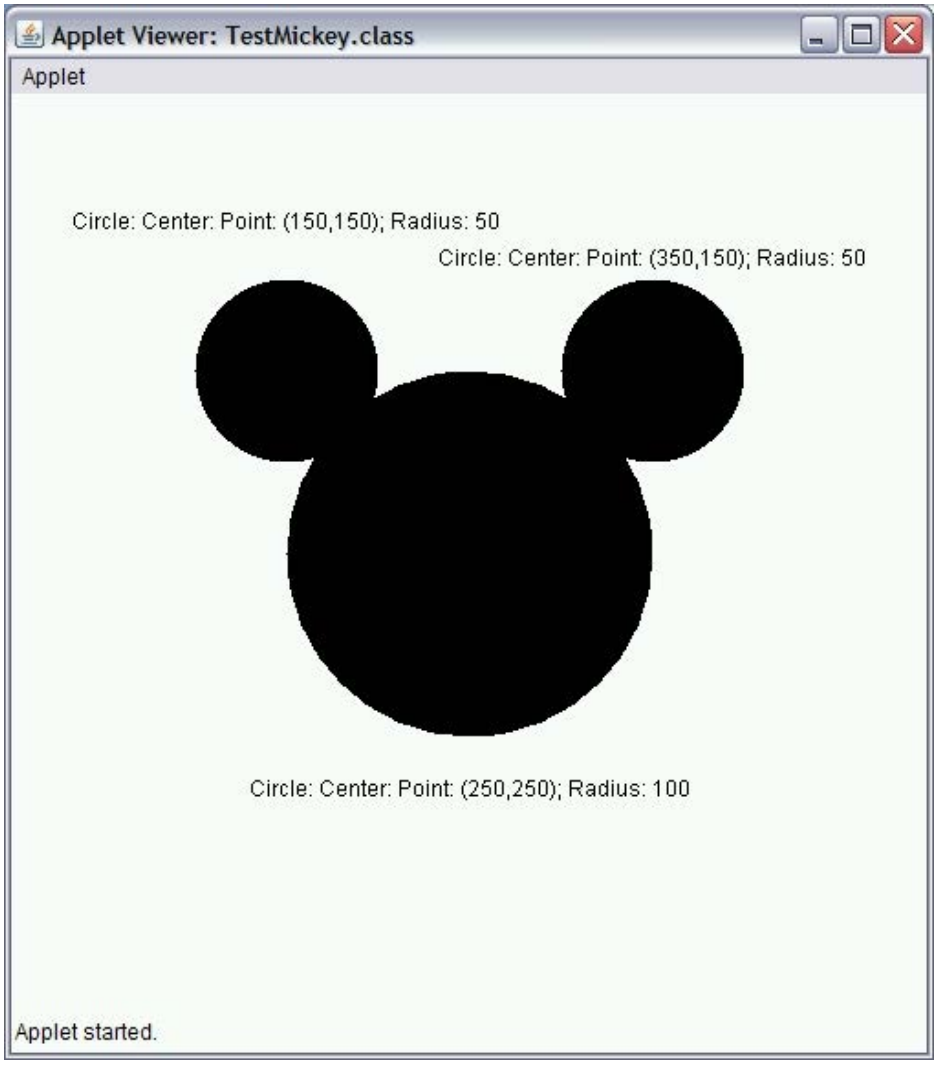
```
TestMickey.java (and TestMickey.html)

TestHouseWithDelays.java and TestHouseWithDelaysController.java
(and TestHouseWithDelays.html)
```

TestMickey uses class Point and class Circle. This is a good one to start with to test your class Shape, Circle, and Point.

TestHouseWithDelays draws a house using all the various shapes in a delayed fashion so you can see the different objects being drawn.

We will compile and use these test programs against your shapes sources to grade this assignment.

Example screen shot:


Applet Viewer: TestMickey.class
Applet
Circle: Center: Point: (150,150); Radius: 50
Circle: Center: Point: (350,150); Radius: 50
Circle: Center: Point: (250,250); Radius: 100
Applet started.

Note: The last Point coordinate represents the value of the last Point to draw a filled Triangle (roof) with multiple (framed) Triangles with a changing Y value of the upper Point. You will see this value change as you run the test program as the roof is filled in.

Files to be turned in:
Point.java, Shape.java, CSE11_Line.java, Circle.java, ARectangle.java, Square.java, Rectangle.java, Triangle.java, objectdraw.jar

Extra Credit

Add a new shape class named CSE11_Polygon. Class CSE11_Polygon should inherit from class Shape. It should have 3 constructors: a no-arg ctor (basically an empty polygon), a copy ctor (to perform a deep copy), and a ctor that takes an array of Points. This array of Points defines the closed polygon shape such that a CSE11_Line is drawn between the Points specified in the array with a line between the last Point and the first Point to close the polygon shape. Make sure to copy the array to a private instance variable (do not just perform a simple assignment; you need a full copy of the array so any changes to the actual argument array will not affect the polygon).

Override toString(), equals(), and hashCode() similar to the other shapes in this assignment.

Provide an appropriate implementation for draw() and move(). You can ignore the fill parameter -- just implement a framed polygon in the specified color.

Write a test driver program to test all the constructors and methods in this new class. Create several polygons with different number of Points, move them, copy them, draw them, check for equality, check that the copy ctor performs a deep vs. shallow copy, etc. For example, multi-point stars, parallelograms, pent/hex/octagons, trapezoids, non-symmetrical polygons, origami shapes, etc.

In your README file, explain your extra credit implementation and your test driver (what it is supposed to display and what parts of the new class it tests) and how to run your extra credit code.

DUE: Before 11:59pm Thursday, March 14