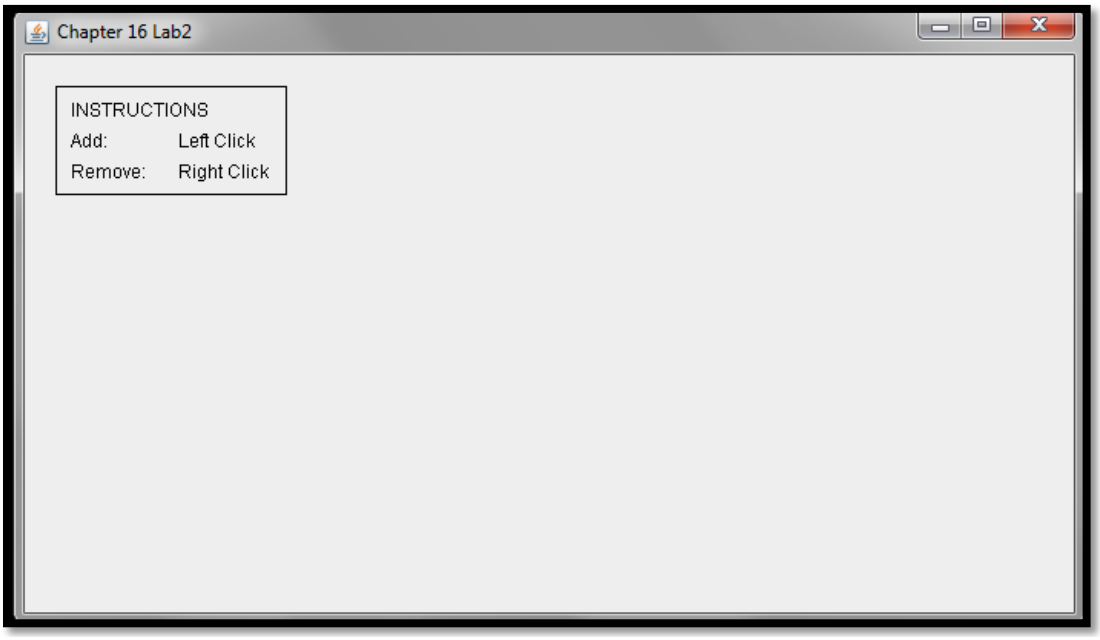
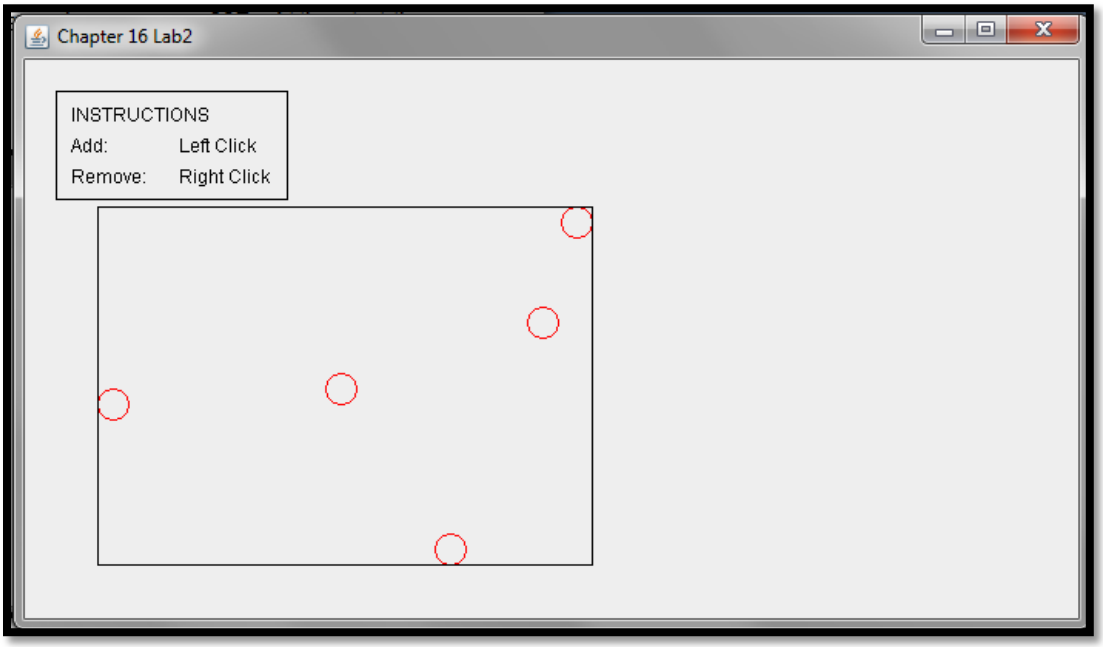
CIT 249: Java II
Chapter 16 Lab 2

In this program we will complete #39 under Programming exercises on page 637. At the start the program will display as:

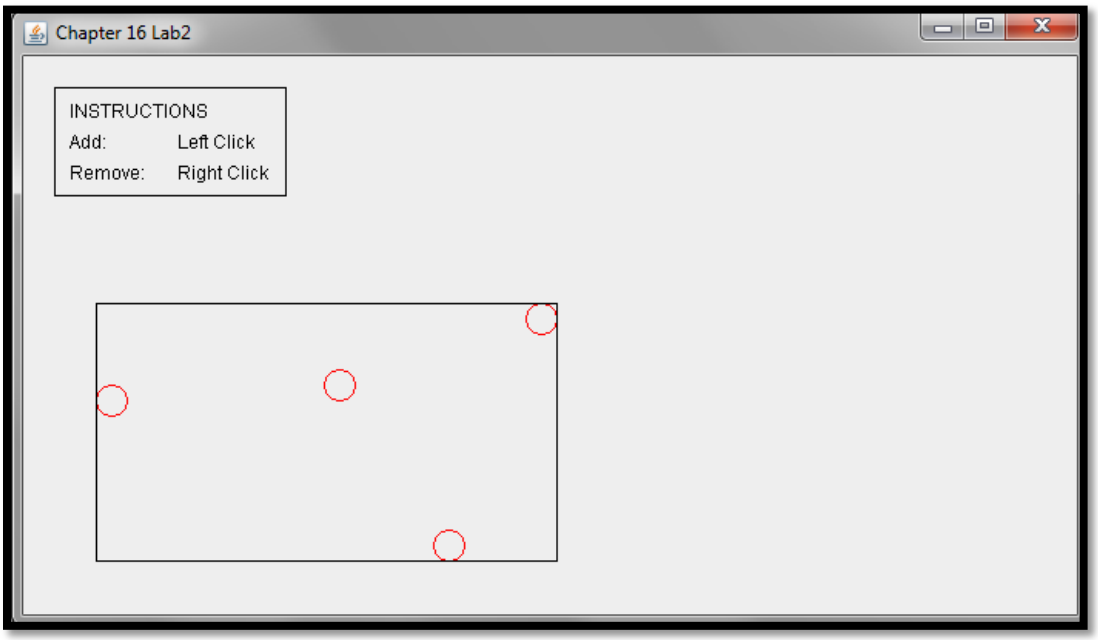


When the left mouse button is pressed:



A circle will display. After multiple presses of the left mouse button a rectangle will surround the circles. Here I pressed the left mouse button 5 times.

If a circle is selected by pressing the right mouse button, the circle selected will disappear.



Notice that the rectangle still surrounds the remaining circles.

MyRectangle2D class

1. Open a new document window and save the file as MyRectangle2D.java. This class will be responsible for setting the dimensions of a rectangle.
2. Type the purpose of the class within block comments:

   ```
   /*  Purpose:  Set the dimensions of a rectangle.  */
   ```

3. Type the class header and opening brace:

   ```
   public class MyRectangle2D
   {
   ```

4. Declare variables for the x, y, width and height. Type:

   ```
   private double x, y; // Center of the rectangle
   private double width, height;
   ```

5. Create the default constructor method by typing:

   ```
   public MyRectangle2D()
   {
     x = y = 0;
     width = height = 1;
   }
   ```

- Here we use an unusual way of setting the default values for x and y to zero. Since both will equal zero we set x to equal y which equals zero.

6. We overload the constructor method by typing:

```
public MyRectangle2D(double x, double y, double width, double height)
{
  this.x = x;
  this.y = y;
  this.width = width;
  this.height = height;
}
```

7. We create our accessor and mutator methods by typing:

```
public double getX()
{
  return x;
}

public void setX(double x)
{
  this.x = x;
}

public double getY()
{
  return y;
}

public void setY(double y)
{
  this.y = y;
}

public double getWidth()
{
  return width;
}

public void setWidth(double width)
{
  this.width = width;
}

public double getHeight()
{
  return height;
}

public void setHeight(double height)
```

```
{
  this.height = height;
}

public double getPerimeter()
{
  return 2 * (width + height);
}

public double getArea()
{
  return width * height;
}
```

8. Our first contains method is next. Type:

```
public boolean contains(double x, double y)
{
  return Math.abs(x - this.x) <= width / 2 && Math.abs(y - this.y) <= height / 2;
}
```

   - This method returns a boolean value using an expression. If the absolute value of x - the x declared in the class is less than or equal to the width divided by 2 AND the absolute value of y minus y declared in the class is less than or equal to height divided by 2, true will be returned, else false will be returned. Both expressions must be true for true to be returned.

9. We overload this method by typing:

```
public boolean contains(MyRectangle2D r)
{
  return contains(r.x - r.width / 2, r.y + r.height / 2) &&
    contains(r.x - r.width / 2, r.y - r.height / 2) &&
    contains(r.x + r.width / 2, r.y + r.height / 2) &&
    contains(r.x + r.width / 2, r.y - r.height / 2);
}
```

   - This is the same as the previous method except that the method has parameters of a MyRectangle2D object and it checks to see if certain conditions are true. All conditions must be true for true to be returned.

10. The overlaps() method checks for overlapping. Type:

```
public boolean overlaps(MyRectangle2D r)
{
  return Math.abs(this.x - r.x) <= (this.width + r.width) / 2 && Math.abs(this.y - r.y) <=
(this.height + r.height) / 2;
}
```

   - Again all conditions must be true for true to be returned.

11. Close the class.
12. Compile the class and fix any errors if necessary.

The Driver class

1. Open a new document and save the file as Ch16Lab2.java.
2. Our first step is to import the predefined classes we require. Type:

```
import java.awt.Point;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.util.ArrayList;
```

3. Type the class header and opening brace having the class extend JFrame.
4. In our constructor method we add a new View object to the frame. Type:

```
public Ch16Lab2()
{
  add(new View());
}
```

   - We will create the View class further on.

5. We create our main method as we have done in previous labs. Type:

```
/** Main method */
public static void main(String[] args)
{
  Ch16Lab2 frame = new Ch16Lab2();
  frame.setTitle("Chapter 16 Lab2");
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  frame.setSize(700, 400);
  frame.setLocationRelativeTo(null); // Center the frame
  frame.setVisible(true);
}
```

6. We next create a inner class named View. It will set up the screen and will extend the JPanel class. Type:

```
class View extends JPanel
{
```

7. First we construct an ArrayList that will contain Vertex objects. Type:

```
private ArrayList<Vertex> list = new ArrayList<Vertex>();
```

- An ArrayList is similar to an array except that it is expandable. ArrayList is a subclass of the Collections class which we will cover in a later chapter. Since JDK 1.5 you must specify what Collections will hold. In this case our ArrayList will hold Vertex objects. Both <Vertex> are required where specified above. If the ArrayList held String objects you would use <String>.

8. Our constructor method is responsible for adding the points on the screen with the click of the left mouse button, and to remove the point with a click of the right mouse button. Type:

```
View()
 {
  addMouseListener(new MouseAdapter()
  {
    public void mouseClicked(MouseEvent e)
    {
      if (e.getButton() == MouseEvent.BUTTON1) {

        // Add a vertex
        list.add(new Vertex(e.getPoint()));
        repaint();
      }
      else if (e.getButton() == MouseEvent.BUTTON3)
      {
        // remove a vertex
        Vertex c = getContainingVertex(e.getPoint());
        if (c != null) {
          list.remove(c);
          repaint();
        }
      }
    }
  });
 }
```

- The left mouse button is specified as MouseEvent.BUTTON1, where the right button is BUTTON3. Yes, the scroll button would be BUTTON2.
- If the left button is pressed we add a new Vertex point to the ArrayList.
- Else if the right button is pressed we remove the point.
- The screen must be repainted in either case.

9. The next method is written slightly different from what you are use to. Type:

```
Vertex getContainingVertex(Point p)
 {
  for (int i = 0; i < list.size(); i++)
    if (((Vertex)list.get(i)).contains(p))
      return (Vertex)(list.get(i));

  return null;
```

```
}
```

- Normally you see public at the beginning of the method header, and you could add it, but it isn't required here.
- A for loop is used to get the Vertices from the ArrayList and return it, else it returns null if the ArrayList contains none.

10. The paintComponent() method is used to paint everything on the screen. Type:

```
protected void paintComponent(Graphics g)
 {
  super.paintComponent(g);

  g.setColor(Color.RED);
  for (int i = 0; i < list.size(); i++) {
   g.drawOval(((Vertex)list.get(i)).getX() - Vertex.RADIUS,
      ((Vertex)list.get(i)).getY() - Vertex.RADIUS, 2 * Vertex.RADIUS, 2 * Vertex.RADIUS);
  }

  drawInstructions(g, 20, 20);

  // Draw the bounding rectangle
  if (list.size() > 0)
  {
   MyRectangle2D rectangle = getRectangle(list);
   g.drawRect((int)(rectangle.getX() - rectangle.getWidth() / 2 - Vertex.RADIUS),
      (int)(rectangle.getY() - rectangle.getHeight() / 2 - Vertex.RADIUS),
      (int)(rectangle.getWidth() + 2 * Vertex.RADIUS),
      (int)(rectangle.getHeight()) + 2 * Vertex.RADIUS);
  }
 }
```

- It invokes the superclass' constructor method passing to it the Graphics object.
- The color is set to red
- It goes through the ArrayList and draws ovals, getting the x, y, and radius.
- It invokes the drawInstructions() method passing to it the Graphics object and x and y values.
- A if statement is used to draw the bounding rectangle.

11. We construct a constant with the instructions as part of an array. Type:

```
final String[] instructions = {"INSTRUCTIONS", "Add:", "Left Click", "Remove:", "Right Click" };
```

12. The drawInstructions() method draws those instructions. Type:

```
void drawInstructions(Graphics g, int x, int y)
 {
  g.setColor(Color.BLACK);
  g.drawRect(x, y, x + 130, y + 50);
  g.drawString(instructions[0], x + 10, y + 20);
```

```
    for (int i = 1; i < instructions.length; i = i + 2)
    {
      g.drawString(instructions[i], x + 10, y + 20 + (i + 1) * 10);
      g.drawString(instructions[i + 1], x + 80, y + 20 + (i + 1) * 10);
    }
  }
```

13. Close the View class.
14. Our next class is a static class named Vertex. This class will be responsible for maintaining the location of the points on the screen. First the class header of:

```
static class Vertex
{
```

15. We create variables for the radius and x and y coordinates. Type:

```
final static int RADIUS = 10;
 int x, y;
```

16. We create the default constructor method and overload it twice. Type:

```
public Vertex()
 {
 }

  public Vertex(int x, int y)
  {
    this.x = x;
    this.y = y;
  }

  public Vertex(Point p)
  {
    this(p.x, p.y);
  }
```

- Notice that one is for the x and y coordinates and the last one is for the points on the screen.

17. We have accessor and mutator methods. Type:

```
public int getX()
 {
   return x;
 }

  public void setX(int x)
  {
    this.x = x;
  }
```

```
public int getY()
{
  return y;
}

public void setY(int y)
{
  this.y = y;
}
```

18. Next is our equals method. Type:

```
public boolean equals(Object o)
 {
  Vertex c = (Vertex)o;
  return c.getX() == x && c.getY() == y;
 }
```

- We construct a Vertex object that equals the Object object passed to the method. The (Vertex) casts the object to a Vertex object. This is permissible since the Object class is the superclass of other objects.

19. The next three methods are for returning the distance between points. Type:

```
public double getDistance(Vertex c)
 {
  return getDistance(x, y, c.x, c.y);
 }

 public static double getDistance(Vertex c1, Vertex c2)
 {
  return getDistance(c1.x, c1.y, c2.x, c2.y);
 }

 public static double getDistance(double x1, double y1, double x2, double y2) {
  return Math.sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
 }
```

20. The contains method return a boolean value. Type:

```
public boolean contains(Point p)
 {
  return getDistance(x, y, p.x, p.y) <= RADIUS;
 }
```

21. Close the Vertex class.
22. It may seem strange but we are still within the Ch16Lab2 class. The View and Vertex classes were inner classes. We have yet to complete the Ch16Lab2 class. We still have several methods to complete. The two inner classes centered mainly on the vertex points. The remainder of the methods in the Ch16Lab2 class will center on the rectangle. The first two are the getRectangle() method which is overloaded once. Type:

```
public static MyRectangle2D getRectangle(ArrayList list)
{
  double[][] points = new double[list.size()][2];

  for (int i = 0; i < points.length; i++) {
    points[i][0] = ((Vertex)(list.get(i))).x;
    points[i][1] = ((Vertex)(list.get(i))).y;
  }

  return getRectangle(points);
}

public static MyRectangle2D getRectangle(double[][] points)
{
  double minX = minX(points);
  double minY = minY(points);
  double maxX = maxX(points);
  double maxY = maxY(points);

  return new MyRectangle2D( (minX + maxX) / 2, (minY + maxY) / 2, maxX - minX, maxY -
minY);
}
```

- Look over the code carefully. Both return MyRectangle2D objects. The first has parameters of an ArrayList whereas the other a multidimensional array.
- The first uses a for loop to get the Vertex points and adds the points to the array and passes this to the other getRectangle() method. Yes they work in unison with each other.

23. Our other methods return the minX, maxX, minY, and maxY points. Type:

```
private static double minX(double[][] points)
{
  double minX = points[0][0];

  for (int i = 0; i < points.length; i++)
    if (minX > points[i][0])
      minX = points[i][0];

  return minX;
}

private static double maxX(double[][] points)
{
  double maxX = points[0][0];

  for (int i = 0; i < points.length; i++)
    if (maxX < points[i][0])
      maxX = points[i][0];
```

```
  return maxX;
}

private static double minY(double[][] points)
{
  double minY = points[0][1];

  for (int i = 0; i < points.length; i++)
    if (minY > points[i][1])
      minY = points[i][1];

  return minY;
}

private static double maxY(double[][] points)
{
  double maxY = points[0][1];

  for (int i = 0; i < points.length; i++)
    if (maxY < points[i][1])
      maxY = points[i][1];

  return maxY;
}
```

24. Close the class and compile the program. Fix any errors if necessary.
25. Run the program and uses the mouse buttons to add points onto the screen. Then use the right mouse button and click the points to remove them.
26. Compress ALL files into a single zip or rar file and submit.