



*Some of these exercises may be demonstrated in class, others you will be expected to practise on your own. Model solutions to these exercises will be made available on the class website.*

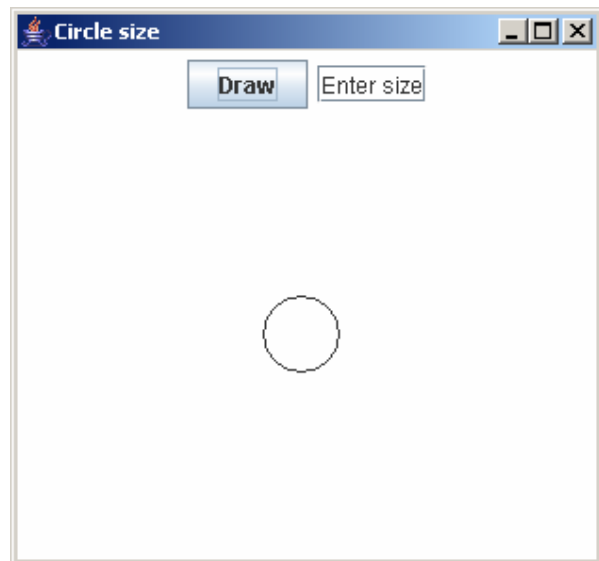
## Event handling (ActionEvents)

### Exercise 24.1

Complete the code for the program shown in the screen shot to the right.

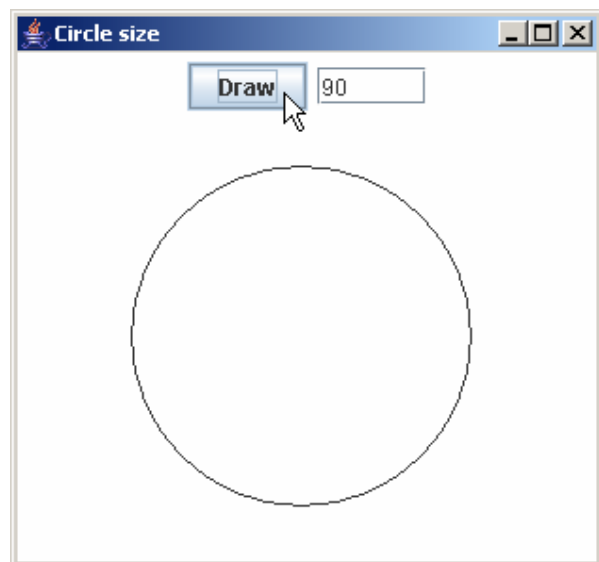
When the window first appears, a JButton labelled "Draw" and a JTextField initially containing the text "Enter size" should be displayed.

A circle should also be drawn in the centre of the window, with a radius of size 20 pixels. The width and the height of the window is 300 pixels.



The user can change the size of the circle that is drawn by entering a value in the JTextField and pressing the "Draw" button.

For example, in the screen shot to the right the user has entered the value 90 and then pressed the "Draw" button. The result of this is that a new circle is drawn, still in the centre of the window, but this time with radius 90.

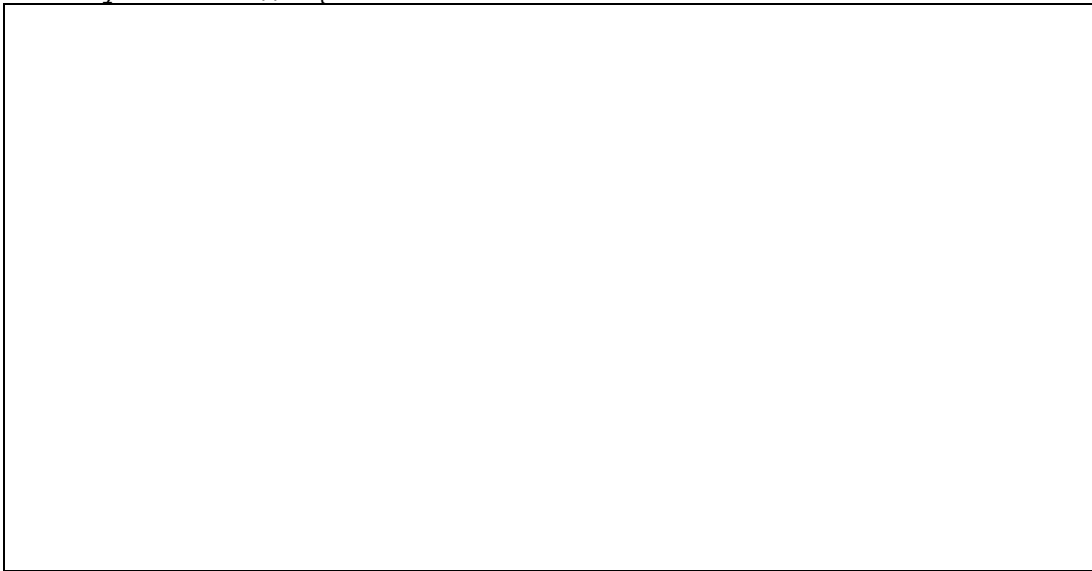


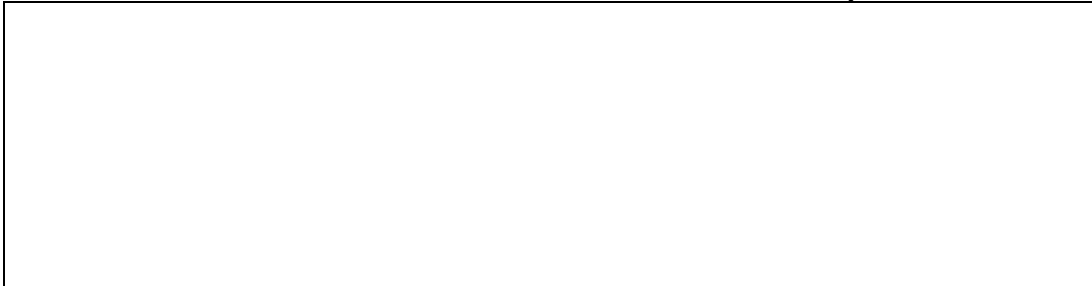
```


import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class MyJPanel extends JPanel implements
                                   ActionListener {

    private JButton bDraw;
    private JTextField tSize;
    private int size;

    public MyJPanel() {
        
    }

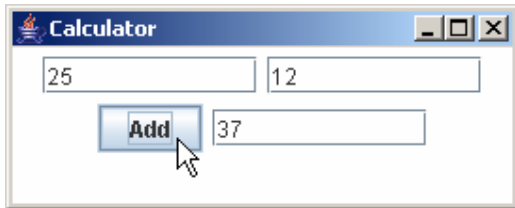
    public void actionPerformed(ActionEvent e) {
        
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        
    }
}

```

## Exercise 24.2

You need to complete the `actionPerformed()` method for the program shown on the right, which consists of 3 `JTextField`s and a `JButton`. The screenshot shows what the window looks like when the program first starts.



When the user enters values into the top two `JTextField`s and then presses the "Add" button, the sum of the entered numbers should be displayed in the third `JTextField`. An example of this is shown in the screenshot on the left.

Complete the `actionPerformed()` method for this program below:

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class MyJPanel extends JPanel implements
    ActionListener {

    private JTextField tOne, tTwo, tResult;
    private JButton bAdd;

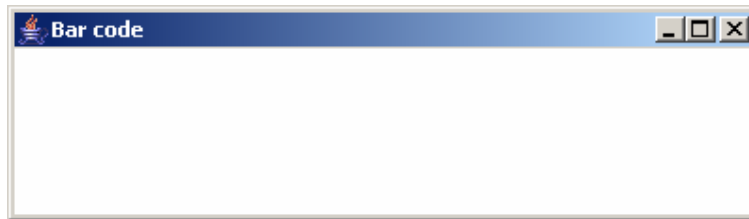
    public MyJPanel() {
        tOne = new JTextField(10);
        tTwo = new JTextField(10);
        tResult = new JTextField(10);
        bAdd = new JButton("Add");
        bAdd.addActionListener(this);
        add(tOne);
        add(tTwo);
        add(bAdd);
        add(tResult);
    }

    public void actionPerformed(ActionEvent e) {
        
    }
}
```

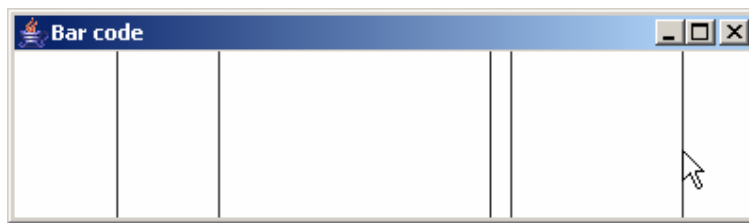
## Event handling (MouseEvents)

### Exercise 25.1

For this exercise, you should write an application which displays vertical lines on the window at each location where the mouse button is pressed. When the program first starts, as shown in the screenshot below, the window should be blank (the window is 300 pixels wide and 100 pixels high):

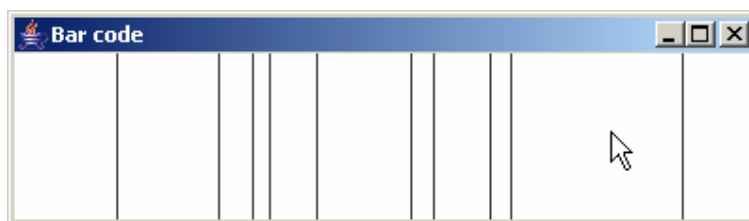


Each time the mouse button is pressed, a vertical line should be drawn on the window from top to bottom. The screenshot below shows the program after the mouse has been pressed five times:



There is a limit on the number of lines that can be displayed however. Only 10 lines can be displayed on the window in total – this value 10 is given by the constant `MAX_LINES` declared in the program.

For example, the screenshot below shows the window after the mouse has been pressed another five times.



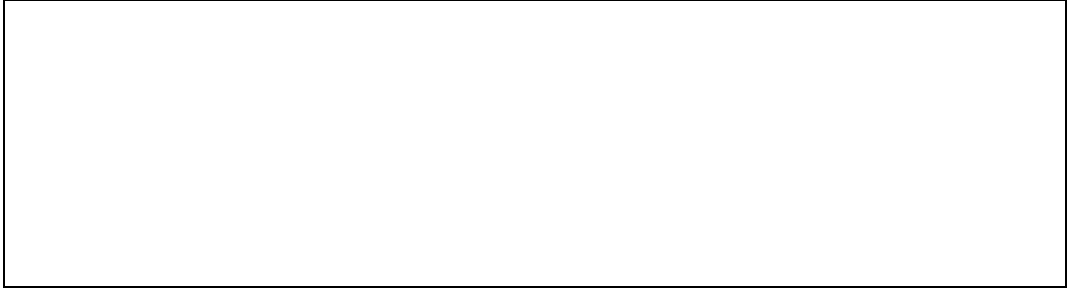
After this point, no further action takes place in the program when the mouse is pressed.

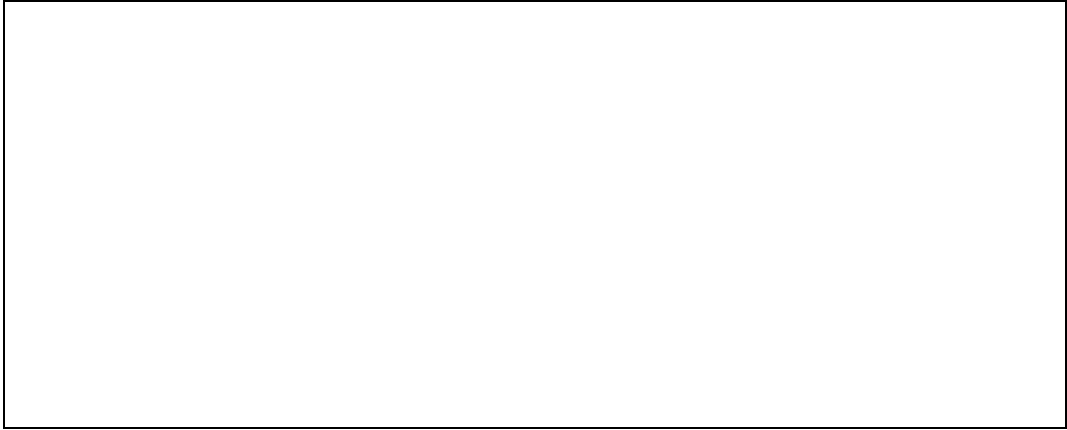
The instance variables have already been declared for you. Complete the rest of the source code on the next page:

```

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

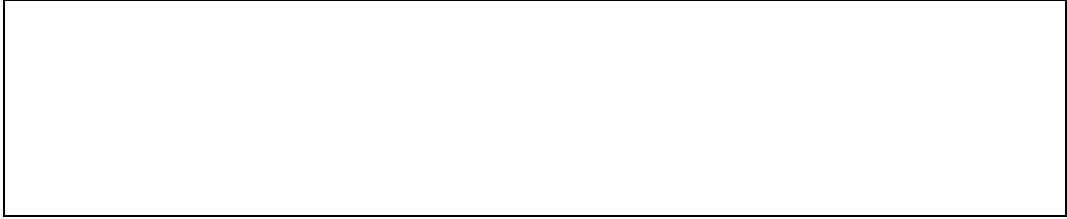
public class MyJPanel extends JPanel implements
                                                MouseListener {
    private final int MAX_LINES = 10;
    private int[] lines;
    private int numLines;

    public MyJPanel() {
        
    }

    public void mousePressed(MouseEvent e) {
        
    }

    public void mouseReleased(MouseEvent e) {}
    public void mouseClicked(MouseEvent e) {}
    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) {}

    public void paintComponent(Graphics g) {
        super.paintComponent(g);

        
    }
}

```

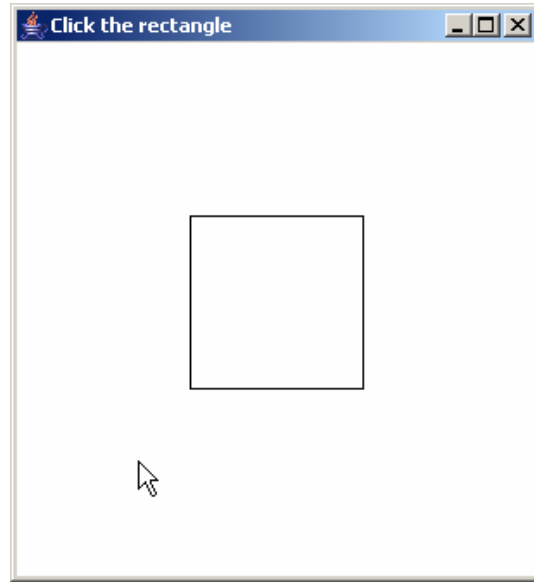
## Points and Rectangles

### Exercise 25.2

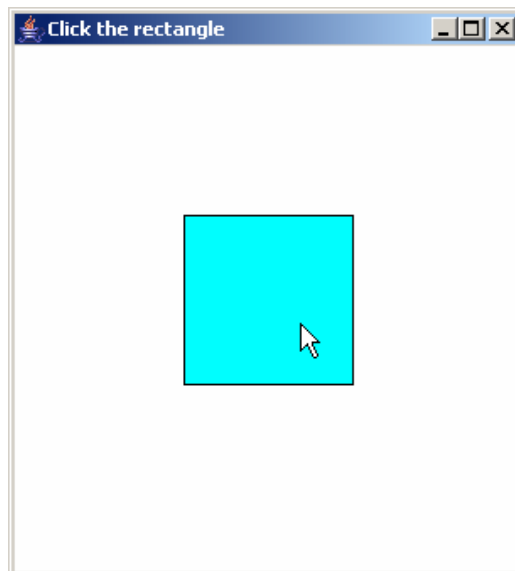
For this exercise, you need to complete the source code for the program described below.

When the window first appears, the outline of a rectangle is drawn in the centre. When the mouse button is pressed, if its location is inside the rectangle, the rectangle will be drawn filled with colour. If the location of the mouse press is outside the rectangle, the rectangle will be drawn in outline (as it appears initially).

For example, when the program first starts (or when the mouse is pressed outside the rectangle), the rectangle is drawn in outline as shown in the screenshot to the right:



If the mouse is pressed inside the rectangle, it is drawn filled with colour, as shown below:



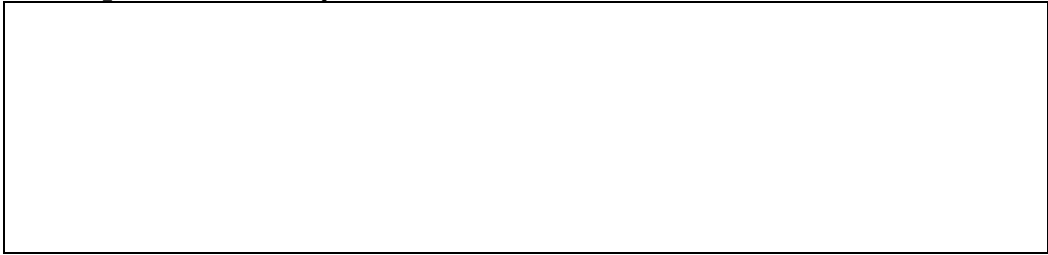
The window is 300 pixels wide and 300 pixels high, and the rectangle is 100 pixels wide and 100 pixels high and is positioned approximately in the middle of the window.

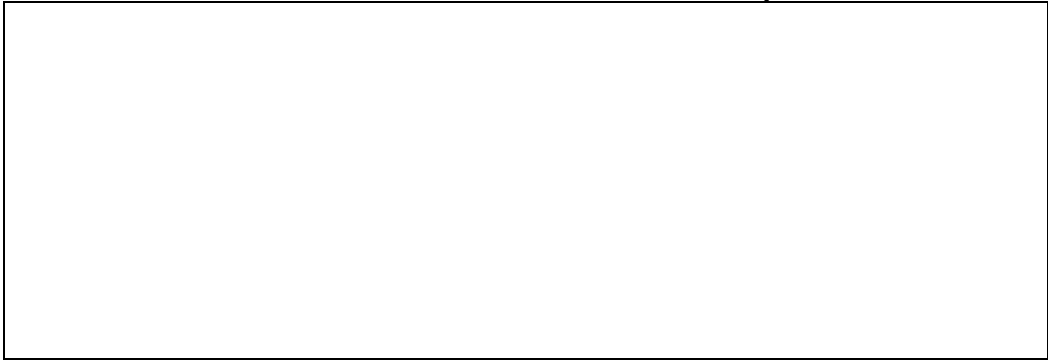
Complete the source code for this program on the next page. The instance variables have been declared for you.

```

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

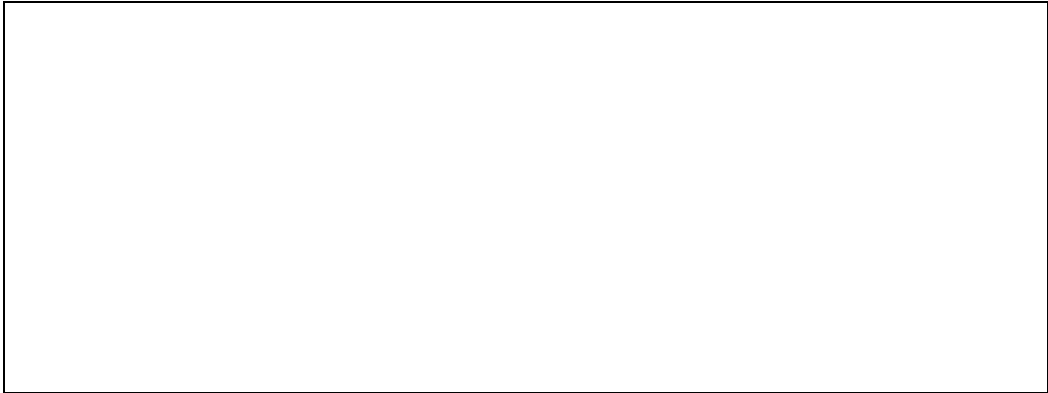
public class MyJPanel extends JPanel implements
    MouseListener {
    private Rectangle rect;
    private boolean clickInside;

    public MyJPanel() {
        
    }

    public void mousePressed(MouseEvent e) {
        
    }

    public void mouseReleased(MouseEvent e) {}
    public void mouseClicked(MouseEvent e) {}
    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) {}

    public void paintComponent(Graphics g) {
        super.paintComponent(g);

        
    }
}

```

### **Exercise 25.3:**

a) What would the output of the following code segment be?

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class MyJPanel extends JPanel {
    Point p;
    Rectangle r1, r2, r3;

    public MyJPanel() {
        p = new Point(80, 100);
        r1 = new Rectangle(50, 75, 50, 30);
        r2 = new Rectangle(110, 90, 40, 40);
        r3 = new Rectangle(80, 120, 90, 20);

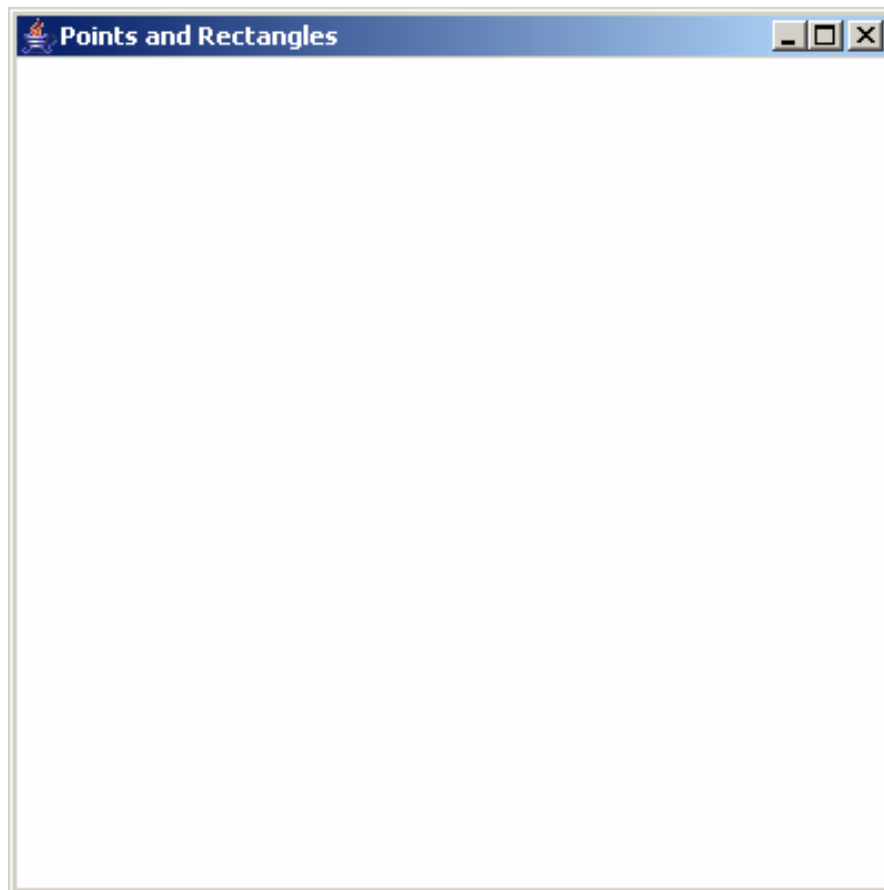
        System.out.println(r1.contains(p));
        System.out.println(r2.contains(p));

        System.out.println(r1.intersects(r2));
        System.out.println(r1.intersects(r3));
        System.out.println(r2.intersects(r3));
    }
}
```



b) If the following `paintComponent()` method was added to the `MyJPanel` class above, sketch what the graphical output of the program would look like in the window below.

```
public void paintComponent(Graphics g) {  
    super.paintComponent(g);  
  
    g.drawRect(r1.x, r1.y, r1.width, r1.height);  
    g.drawRect(r2.x, r2.y, r2.width, r2.height);  
    g.drawRect(r3.x, r3.y, r3.width, r3.height);  
  
    g.fillOval(p.x-2, p.y-2, 4, 4);  
}
```



### **Exercise 25.4:**

Complete the `leftmost()` method which is passed an array of `Point` objects as a parameter and returns a reference to the `Point` in the array which has the smallest `x` value (i.e. is the leftmost point).

If you complete the method correctly, then the code below:

```
Point[] pts = new Point[5];
pts[0] = new Point(10, 10);
pts[1] = new Point(20, 50);
pts[2] = new Point(5, 999);
pts[3] = new Point(25, 25);
pts[4] = new Point(10, -10);

Point p = leftmost(pts);
System.out.println(p.x + " , " + p.y);
```

would produce the output:

```
5 , 999
```

Complete the `leftmost()` method in the space below:

```
private Point leftmost(Point[] pts) {
```

```
}
```

## Arrays of Points

### Exercise 25.5:

The output of the following code segment:

```
Point[] pts = new Point[3];

pts[0] = new Point(200, 100);
pts[1] = new Point(100, 200);
pts[2] = new Point(300, 200);

System.out.println(pts[0].x + " , " + pts[0].y);
System.out.println(pts[1].x + " , " + pts[1].y);
System.out.println(pts[2].x + " , " + pts[2].y);
```

is given below:

```
200 , 100
100 , 200
300 , 200
```

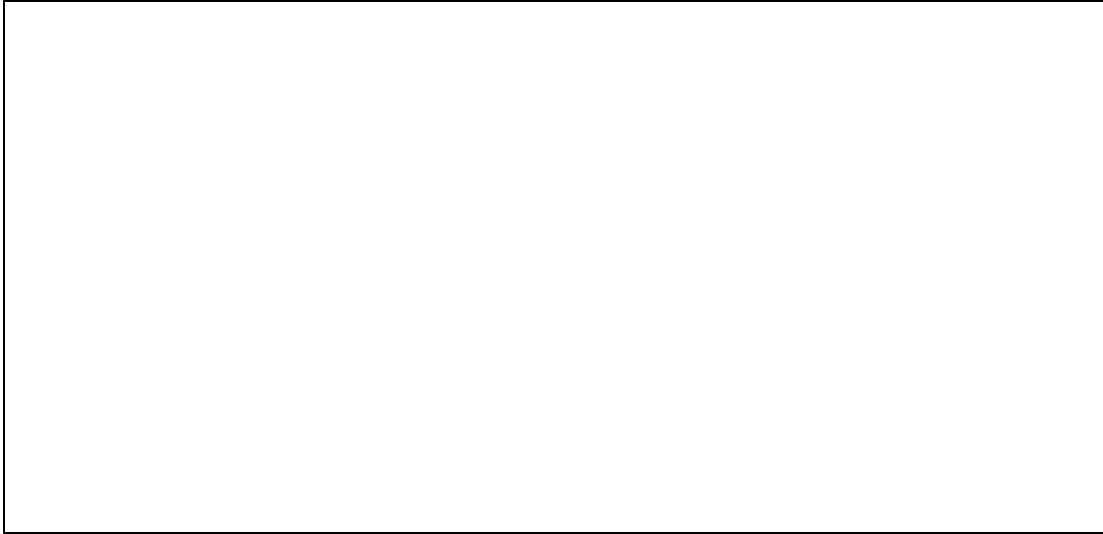
What would be the output if the following code is added after the statements above:

```
Point temp = pts[1];
pts[1] = pts[2];
pts[0].y = pts[1].x;
pts[2].x = pts[0].x;
pts[2] = temp;

System.out.println(pts[0].x + " , " + pts[0].y);
System.out.println(pts[1].x + " , " + pts[1].y);
System.out.println(pts[2].x + " , " + pts[2].y);
```

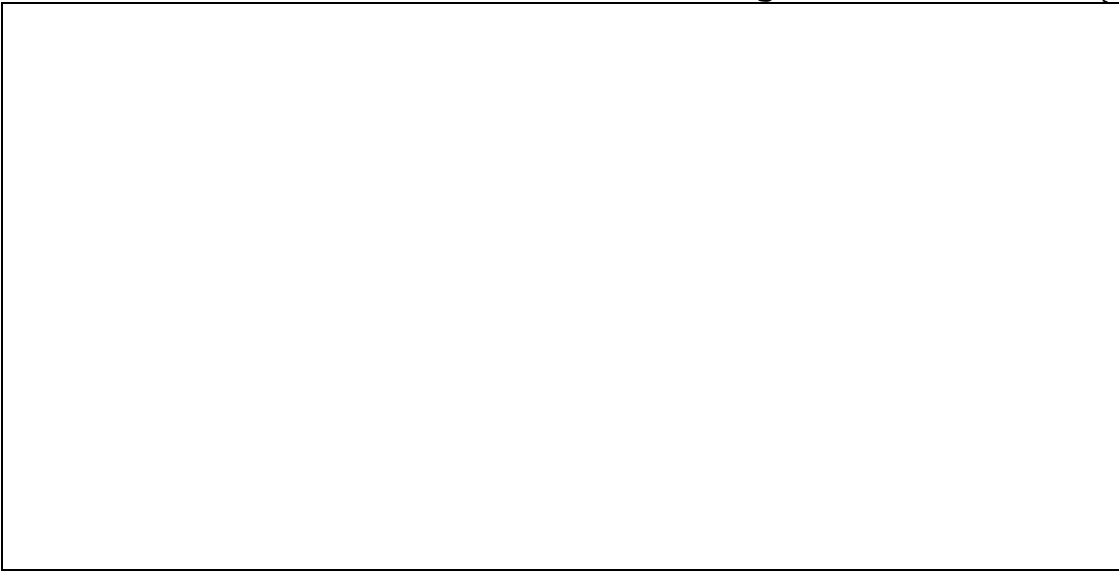
### **Exercise 25.6:**

Complete the `containsAll()` method which is passed a `Rectangle` object and an array of `Point` objects as parameters and returns `true` if all of the `Points` in the array are contained inside the position of the `Rectangle`:

```
private boolean containsAll(Rectangle rect,
                           Point[] pts) {  
      
}
```

### **Exercise 25.7:**

Complete the `intersectsAny()` method which is passed a `Rectangle` object and an array of `Rectangle` objects as parameters and returns `true` if the first `Rectangle` intersects with any of the `Rectangles` stored in the array:

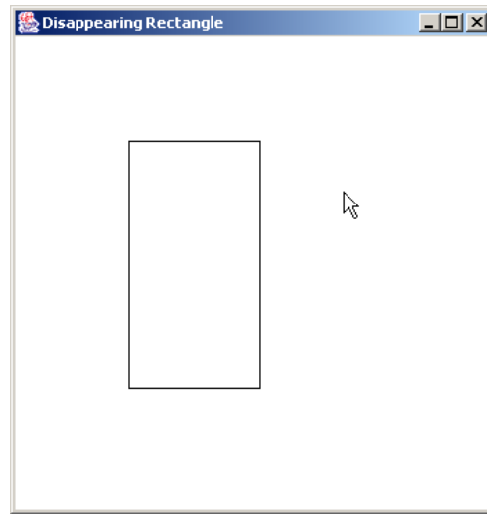
```
private boolean intersectsAny(Rectangle rect,
                             Rectangle[] theRects) {  
      
}
```

### **Exercise 25.8:**

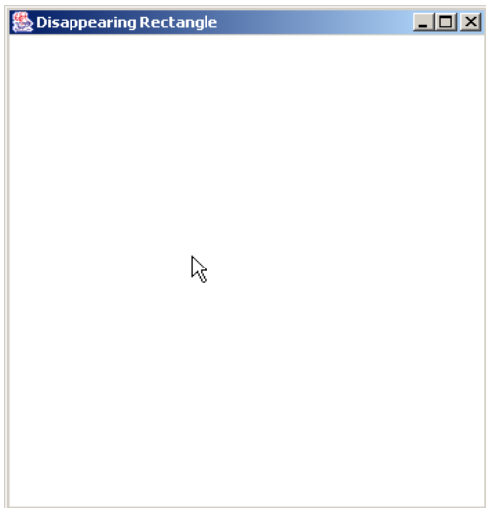
For this exercise you need to complete the source code for a program that behaves as follows.

When the program first starts, a rectangle should be drawn on the screen at some random location, and with a random width and a random height.

If the mouse button is pressed outside the rectangle, nothing should happen – as shown in the screenshot on the right:



However, if the mouse button is pressed anywhere inside the rectangle, the rectangle should disappear. This is shown in the screenshot below:



Make sure that if the mouse button is pressed after the rectangle has disappeared, the code you write would not generate any `NullPointerException`s.

The constructor method has been written for you – you need to complete the `mousePressed()` and `paintComponent()` methods.

```


import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class MyJPanel extends JPanel implements
                                                MouseListener {
    private Rectangle rect;

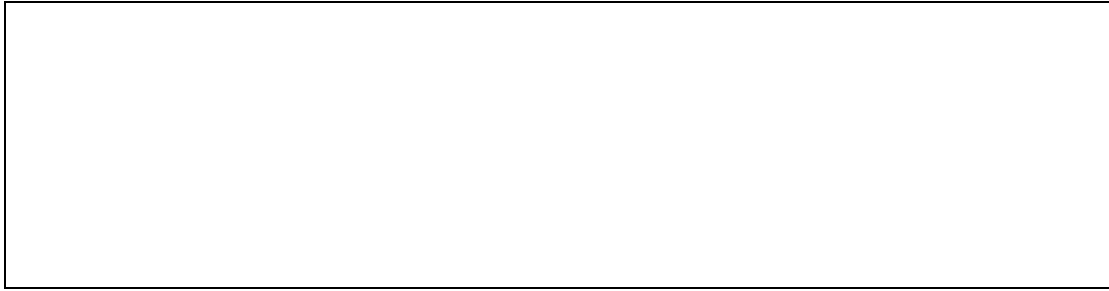
    public MyJPanel() {
        int x = (int)(Math.random() * 200);
        int y = (int)(Math.random() * 200);
        int width = (int)(Math.random() * 200);
        int height = (int)(Math.random() * 200);

        rect = new Rectangle(x, y, width, height);

        addMouseListener(this);
    }

    public void mousePressed(MouseEvent e) {

    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);


    }

    public void mouseReleased(MouseEvent e) {}
    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) {}
    public void mouseClicked(MouseEvent e) {}
}

```

## Animation

### Exercise 27.1:

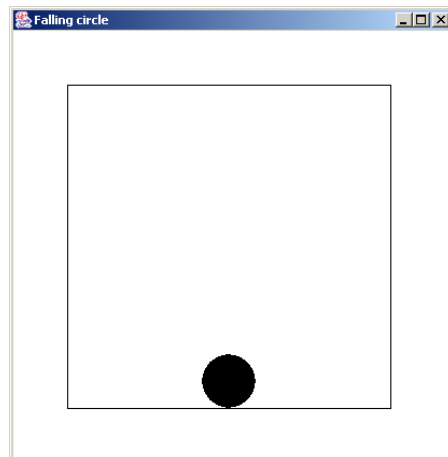
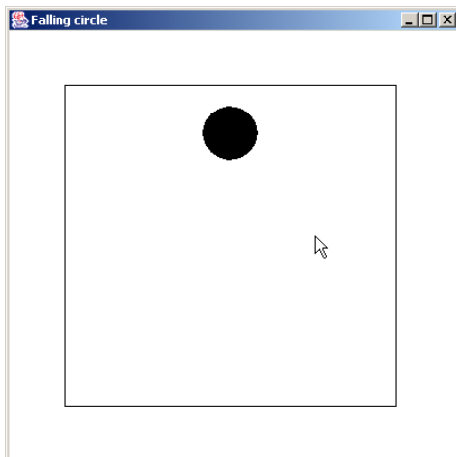
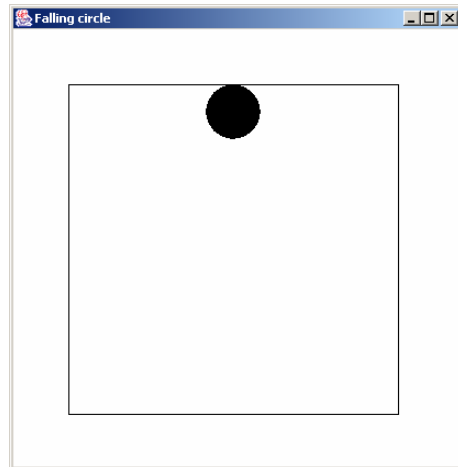
For this exercise, you need to complete the source code for the `MyJPanel` class so that the program performs exactly as described below.

When the window first appears, a rectangular border is drawn near the edges of the window. In addition, a filled-in circle is drawn so that it just touches the top edge of this rectangular border. The horizontal position of this circle is centred exactly in the middle of the rectangular border.

The screenshot on the right shows what the window should look like when it first appears.

When the mouse button is pressed anywhere inside the window, this circle should start moving down the screen as though it were falling. For example the screenshot on the left below shows the program shortly after the mouse button has been pressed.

The circle should continue to fall down the screen until it reaches the bottom edge of the rectangular boundary, at which point it must stop as shown on the right below.

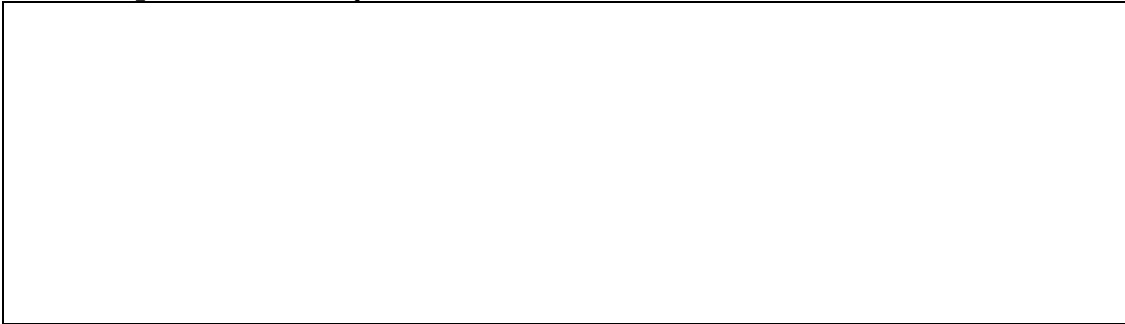



Notice, the `paintComponent()` method has been provided to you. You need to complete the constructor method, the `actionPerformed()` method and the `mousePressed()` method.

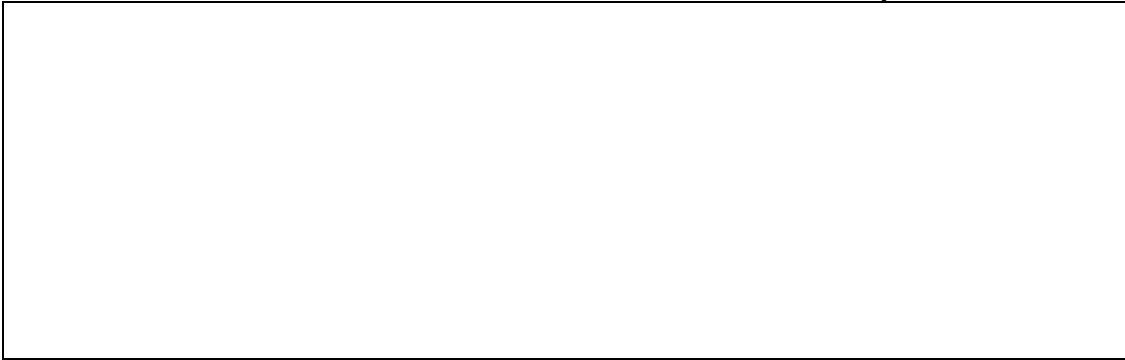
```

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class MyJPanel extends JPanel implements
    ActionListener, MouseListener {
    private Timer t;
    private Point pos;

    public MyJPanel() {
        
    }

    public void mousePressed(MouseEvent e) {
        
    }

    public void actionPerformed(ActionEvent e) {
        
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);

        g.fillOval(pos.x-25, pos.y-25, 50, 50);
        g.drawRect(50, 50, 300, 300);
    }

    public void mouseReleased(MouseEvent e) {}
    public void mouseClicked(MouseEvent e) {}
    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) {}
}

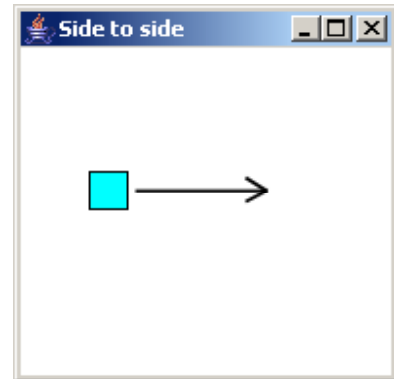
```



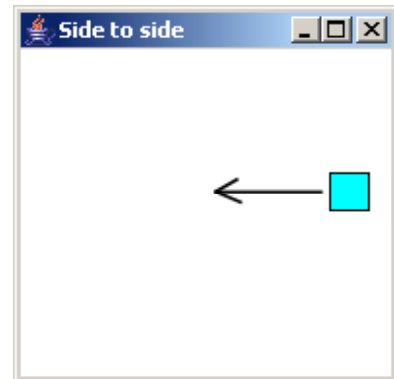
### Exercise 27.2:

For this exercise you need to complete the `MyJPanel` class for the program described below.

When the program first starts, a square is drawn near the left hand edge of the window. This square should start moving towards the right as indicated by the arrow in the screen shot to the right.



When the square reaches the right hand edge of the window, it should change direction and start moving back towards the left, as shown in the screenshot on the right.



The side length of the square should be 20 pixels, and you can assume the window is 200 pixels wide.

You don't need to worry about what happens when the square reaches the left hand edge of the window.

```
import java.awt.*;  
import javax.swing.*;  
import java.awt.event.*;
```

```
public class MyJPanel extends JPanel implements  
                                   ActionListener {  
    private Timer t;  
    private Rectangle theSquare;  
    private boolean movingLeft;  
  
    public MyJPanel() {  
        setBackground(Color.white);  

```



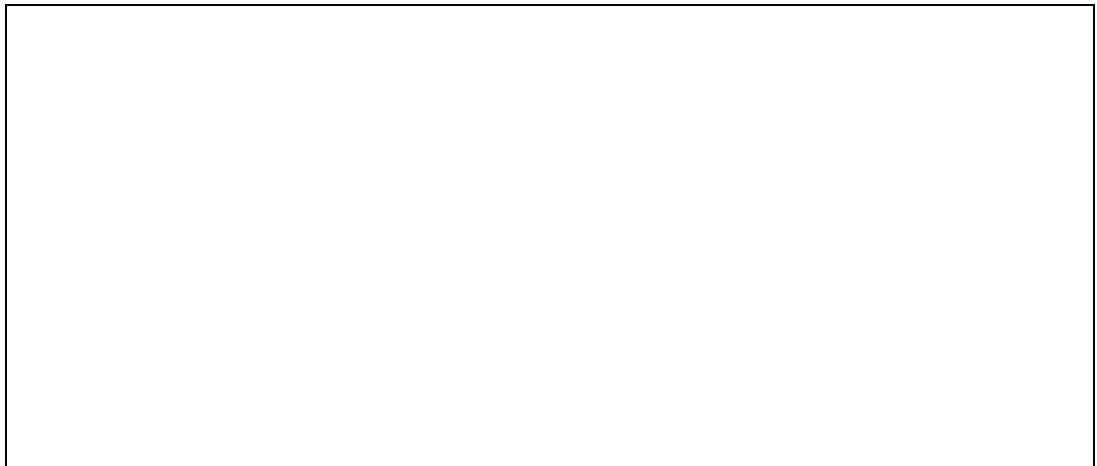
```
        t = new Timer(100, this);  
        t.start();  
    }
```

```
public void actionPerformed(ActionEvent e) {
```



```
}
```

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



```
}
```

```
}
```