This is a *closed book* exam. You have 90 minutes to complete the exam. All intended answers will fit in the spaces provided. You may use the back of the preceding page for additional space if necessary, but be sure to mark your answers clearly.

Be sure to give yourself enough time to answer each question— the point values should help you manage your time.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
<th>Description</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>Loops</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>18</td>
<td>Classes</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>Conditions</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>14</td>
<td>Declarations</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>Recursion</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td>GUI Programming</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>90</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

I have neither given nor received aid on this examination.

Signature: ________________________________

Name: ________________________________
1. (10 points) ................................................................. Loops

The following is a simple but complete Java program that draws a picture. Draw a sketch of the picture it will produce.

```java
import objectdraw.*;

class Mystery extends WindowController {
    public void begin() {
        int i = 0;
        while (i < 3) {
            int j = 2 - i;
            double y = 100 - i * 10;
            while (j >= 0) {
                double x = 100 + j * 20 + i * 10;
                new FramedRect(x, y, 18, 8, canvas);
                j = j - 1;
            }
            i = i + 1;
        }
    }
}
```
2. (18 points) ........................................... Classes

This question asks you to write a class that models a reversible sign for a store window that says “open” on one side and “closed” on the other. For example, here is the open side for one such sign:

![Open Sign](image)

Of course, our version will look a little different. Here are the two sides of our sign:

![Open and Closed Signs](image)

The sign hangs from a single string connected to a thumb tack (the black circle). The string and tack are centered above the sign’s message board. We have already included the code to create the tack and string. Your first job will be to declare the instance variables and add the code to the constructor needed to place the rectangle and message on the screen. The constructor takes the x and y coordinates for the center of the tack and creates the sign accordingly.

The sign may be repositioned by pressing the mouse on the tack and dragging it. You can change the sign message from “Open” to “Closed”, and vice versa, by clicking on the sign’s message board. In order to support this functionality, the Sign class must implement a move method and a flipSign method, as well as tackContains and boardContains methods. Fill in those method bodies below.

Do not worry about centering the text in the message board— just put the text at the given offsets inside the board, and use the constants OPEN_MESSAGE and CLOSED_MESSAGE as the messages appearing on the sign.

```java
class Sign {

    // The width and height of the sign
    static final double SIGN_WIDTH = 60;
    static final double SIGN_HEIGHT = 20;

    // The offset from the sign's top-left corner for the
    // message.
    static final double TEXT_OFFSET_X = 5;
    static final double TEXT_OFFSET_Y = 3;

    // The messages to display
    static final String OPEN_MESSAGE = "OPEN";
    static final String CLOSED_MESSAGE = "CLOSED";

    // length of string
    static final double STRING_HEIGHT = 15;

    // Radius of tack holding up sign
    static final double TACK_RADIUS = 5;

    // The sign's message: "Open" or "Closed"
    String message = OPEN_MESSAGE;
    // The location of the sign
    Point location = new Point(TEXT_OFFSET_X, TEXT_OFFSET_Y);
    // The location of the tack
    Point tack = new Point(TEXT_OFFSET_X, TEXT_OFFSET_Y);

    // The location of the string
    Point string = new Point(TEXT_OFFSET_X, TEXT_OFFSET_Y);

    // The message board
    Rectangle board = new Rectangle(TEXT_OFFSET_X, TEXT_OFFSET_Y, SIGN_WIDTH, SIGN_HEIGHT);

    // The tack
    Circle tack = new Circle(TEXT_OFFSET_X, TEXT_OFFSET_Y, TACK_RADIUS);

    // The string
    Rectangle string = new Rectangle(TEXT_OFFSET_X, TEXT_OFFSET_Y, STRING_WIDTH, STRING_HEIGHT);

    // Constructor
    public Sign(Point location) {
        this.location = location;
        // Place the sign on the screen
        board.createWindow(location.getX(), location.getY(), SIGN_WIDTH, SIGN_HEIGHT);
        // Place the tack on the screen
        tack.createWindow(location.getX(), location.getY(), TACK_RADIUS);
        // Place the string on the screen
        string.createWindow(location.getX(), location.getY(), STRING_WIDTH, STRING_HEIGHT);
    }

    // Move the sign
    public void move(Point location) {
        board.setLocation(location.getX(), location.getY());
        tack.setLocation(location.getX(), location.getY());
        string.setLocation(location.getX(), location.getY());
    }

    // Flip the sign
    public void flipSign() {
        message = message.equals(OPEN_MESSAGE) ? CLOSED_MESSAGE : OPEN_MESSAGE;
    }

    public boolean isOpen() {
        return message.equals(OPEN_MESSAGE);
    }

    public boolean isClosed() {
        return message.equals(CLOSED_MESSAGE);
    }
}
```

```
// declare instance variables here:
private FilledOval tack;
private Line hanger;

public Sign(double x, double y, DrawingCanvas canvas) {
    tack = new FilledOval(x - TACK_RADIUS, y - TACK_RADIUS,
                          2*TACK_RADIUS, 2*TACK_RADIUS, canvas);
    hanger = new Line(x, y, x, y + STRING_HEIGHT, canvas);
}

public boolean tackContains(Location pt) {
}

public boolean boardContains(Location pt) {
public void move(double dx, double dy) {

}

public void flipSign() {

}

}
3. (12 points) ................................................................. Conditionals

Below you will find several code fragments involving the use of if statements. Each of these fragments is in some way less concise than it could be. For each fragment, show how to write code that is more concise and clear but that performs exactly the same function. Assume that point refers to a Location, somethingHappened is a boolean, box is a FramedRect and all other variable names are doubles.

(a) if ( box.contains(point) && point.getX() < midpoint ) {
    leftScore = leftScore + 1;
} else if ( box.contains(point) && point.getX() >= midpoint ) {
    rightScore = rightScore + 1;
}

(b) if ( point.getX() < left ) {
    leftScore = leftScore + 1;
} else if ( point.getX() >= left && point.getX() < right ) {
    rightScore = rightScore + 1;
} else if ( point.getX() >= right ) {
    rightScore = rightScore + 1;
}
(c) if ( point.getX() < midpoint && point.getY() < midpoint ) {
    somethingHappened = true;
} else {
    somethingHappened = false;
}
4. (16 points) Recursion

Consider the caterpillar shown below:

![Caterpillar Image]

This drawing is formed by constructing a number of body segments that look like:

![Body Segment Image]

followed by a single caterpillar head:

![Head Image]

We would like to write a program in which we can draw a caterpillar and then drag it around the screen using the mouse. The recursive structure of a caterpillar should be obvious. A caterpillar is either a single body segment (the caterpillar’s rear end) followed by (and slightly overlapped with) a caterpillar whose body has one fewer segments, or it is simply a caterpillar’s head. A complete implementation of a `CaterpillarHead` class is shown on the next page. This class is designed to serve as the base case for a recursive caterpillar definition. To complete the recursive definition, we need to implement a `Caterpillar` class that can construct caterpillars with multiple body parts. Both the `CaterpillarHead` class and the `Caterpillar` class will implement an interface named `CaterpillarInterface`. This interface is shown below.

```java
public interface CaterpillarInterface {
    public boolean contains(Location point);
    public void move(double dx, double dy);
    public void moveAndResize(double dx, double expansion);
}
```

The behavior associated with the `move` and `contains` methods should be familiar to you. We will discuss the function of the `moveAndResize` method in the last part of this problem.
We have provided a complete copy of the definition of the CaterpillarHead class below. You may want to refer back to this code as you answer the questions on the following pages.

```java
public class CaterpillarHead implements CaterpillarInterface {

    // Names for all the parts of the drawing of the head
    private FilledOval head;
    private FilledOval leftEye, rightEye;
    private FilledOval mouth;

    public CaterpillarHead( double x, double y, double diameter, DrawingCanvas canvas) {
        // construct the oval for the head
        head = new FilledOval(x, y, diameter, diameter, canvas);
        head.setColor(Color.gray);

        // add the eyes and mouth
        leftEye = new FilledOval(x + diameter/4, y + diameter/4, diameter/8, diameter/8, canvas);
        rightEye = new FilledOval(x + 5*diameter/8, y + diameter/4, diameter/8, diameter/8, canvas);
        mouth = new FilledOval(x + diameter/4, y + 5*diameter/8, diameter/2, diameter/8, canvas);
    }

    // move all parts of the caterpillar's head by the specified offsets
    public void move(double dx, double dy) {
        head.move(dx, dy);
        leftEye.move(dx, dy);
        rightEye.move(dx, dy);
        mouth.move(dx, dy);
    }

    // check to see if the head contains a given location
    public boolean contains(Location point) {
        return head.contains(point);
    }

    // move the head horizontally by dx
    public void moveAndResize(double dx, double expansion) {
        move(dx, 0);
    }
}
```
(a) Below you will find a portion of the definition of the **Caterpillar** class including the instance variable declarations and most of the definition of the constructor. In the constructor, we have omitted the condition used in the if statement and the actual parameter that should be included in the recursive invocation of the constructor.

What condition should be used in the if statement?

What should be used for the missing parameter of the recursive use of the **Caterpillar** constructor?

```java
public class Caterpillar implements CaterpillarInterface {
    private FilledOval segment;
    private FilledRect leg;
    private CaterpillarInterface rest;

    // Create a caterpillar with its tail end at (x,y), made up of "segments" body parts plus a head with each segment and the head having the diameters specified.
    public Caterpillar( double x, double y, double diameter, int segments, DrawingCanvas canvas){
        leg = new FilledRect( x + diameter/8, y+3*diameter/4,
                              diameter/4, diameter/2,
                              canvas);
        segment = new FilledOval( x,y, diameter,diameter, canvas);
        segment.setColor(Color.gray);

        // draw the rest of the caterpillar
        if ( __________________________ ) {
            rest = new CaterPillarHead( x+3*diameter/4, y-diameter/4 ,
                                         diameter, canvas);
        } else {
            rest = new Caterpillar( x+3*diameter/4, y, diameter, __________________________, canvas);
        }
    }

    // method bodies for move, contains and moveAndResize have been omitted
}
```
(b) Given the instance variable declarations and constructor shown above, provide an appropriate definition for the move method of the Caterpillar class.

    // move the entire caterpillar by the specified offsets
    public void move(double dx, double dy)
    {
    }
Real caterpillars inch along using a motion in which their bodies alternately contract and expand. The `moveAndResize` method is intended to make it possible to animate such motion with our `Caterpillar` class. Assuming we have declared

```java
Caterpillar inchy;
```
and initialized the variable `inchy`, then an invocation of the form

```java
inchy.moveAndResize( 50, 10 );
```
should move the segments and head of the caterpillar so that

- the leftmost segment of `inchy` moves 50 pixels to the right, and
- the other segments are repositioned so that each segment’s center is 10 pixels farther away from the center of the segment to its left that it had been. That is, the second segment would move 60 pixels to the right, the third would move 70 and so on.

An invocation of the form

```java
inchy.moveAndResize( 50, -10 );
```
would move the leftmost segment 50 pixels to the right while repositioning all the other segments so that they are 10 pixels closer together. The first segment would be moved 50 pixels, the second would only be moved 40 pixels, the third would only move 30 pixels and so on.

With such a method, we would be able to animate the caterpillar’s crawling motion by invoking `moveAndResize` alternately using positive and negative values for the expansion factor.

Below, please provide a definition for the `moveAndResize` method of the `Caterpillar` class:

```java
// move the parts of the caterpillar horizontally so that the rear end moves dx and the spacing between segments changes by the value of expansion
public void moveAndResize(double dx, double expansion)
{
}
```
5. (14 points) ........................................................... Declarations

On this and the following page, we have included the (almost) complete code for a class that implements an animated helium balloon. The class extends ActiveObject. The constructor draws a circle for the balloon together with a line representing a string tied to the balloon. The `run` method makes the balloon rise or fall depending how well inflated it is. The run method also shrinks the balloon each time it is moved to simulate gas leaking out. To make it fun to play with, the balloon class includes an `inflate` method. The idea is that the WindowController used with a HeliumBalloon would invoke `inflate` each time the user clicked. That way, if the balloon starts to fall, the user can pump it up a bit to make it float again. The balloon class is designed to limit how much the balloon can be inflated or deflated.

While the class is nearly complete, a few of the names that are used in the class are never declared. In particular, if you examine the class carefully, you will notice that although the names

- speed
- bottom
- flation cord
- startTime
- elapsedTime
- balloon

are used in the code, there are no declarations for any of these names in the program. (To make it easier to find the uses of these names within the program, we have underlined them). We would like you to add the declarations for the names shown above that are needed to make the class complete. In doing this remember: 1) The only modification you should make to the code is the addition of declarations. 2) You should declare each name in the most appropriate way (i.e., as an instance variable or as a variable local to a method or as a formal parameter to a method). 3) You should make each name’s declaration as local as possible while making the class correct.

We have attempted to leave enough extra space in each area of the program where declarations could be added to enable you to write the declarations you believe are needed in their appropriate places.

```java
public class HeliumBalloon extends ActiveObject {
    private static final double DELAY = 60;

    // rates at which balloon grows and shrinks
    private static final double INFLATION = 2;
    private static final double DEFLATION = .2;

    // dimensions for the balloon and cord
    private static final double MINSIZE = 70;
    private static final double FLOATSIZE = MINSIZE*1.2;
    private static final double MAXSIZE = 1.6*MINSIZE;
    private static final double STRINGLEN = 90;

    public HeliumBalloon(DrawingCanvas canvas) {

        // construct the balloon at (0,0)
        balloon = new FilledOval(0,0,MINSIZE, MINSIZE, canvas);

        cord = new Line(MINSIZE/2, MINSIZE, MINSIZE/2, MINSIZE+STRINGLEN, canvas);

        // move the balloon to the bottom and center of the canvas
        bottom = canvas.getHeight() - MINSIZE - STRINGLEN;
    }
}
```
move(canvas.getWidth()/2 - MINSIZE/2, bottom);

start();
}

// move the balloon by a specified offset
public void move(double dx, double dy){
    balloon.move(dx,dy);
    cord.move(dx,dy);
}

// change the width and height of the balloon while keeping it
// centered on the same point and keeping it connected to the cord
private void inOrDeFlate(){
    if (flation > 0 && balloon.getWidth() < MAXSIZE ||
        flation < 0 && balloon.getWidth() > MINSIZE ) {
        // change the size
        balloon.setWidth(balloon.getWidth() + flation);
        balloon.setHeight(balloon.getWidth());

        // recenter the balloon
        balloon.move(-flation/2,-flation);
    }
}

// increase the size of the balloon a bit
public void inflate(){
    inOrDeFlate(INFLATION);
}

public void run(){

    startTime = System.currentTimeMillis();
    while (true) {
        pause(DELAY);
        elapsedTime = System.currentTimeMillis()-startTime;
        startTime = startTime + elapsedTime;

        // move the balloon up or down depending on its current size
        speed = (FLOATSIZE-balloon.getWidth())/1000;
        if (balloon.getY() < bottom || speed < 0 ){
            move(0, elapsedTime*speed);
        }

        // let a little helium escape
        inOrDeFlate(-DEFLATION);
    }
}
6. (20 points) ................................................................. GUI Components

For this problem, we want you to write a class that has the following appearance.

When you press the Bigger button, the text will grow in size. Clicking the Smaller button will make
the text appear in a smaller font (unless the size would become less than 0).

We have provided the code to display some text with an appropriate initial size and to add the buttons
to the display, but you must write the rest of the class so that the program behaves as desired when
someone clicks on the buttons. If you add any additional methods to the class, write them in the space
provided.

class FontSizer extends WindowController
{

    // Location of text
    private static final double TEXT_LEFT = 5;
    private static final double TEXT_TOP = 5;

    // The initial size of the text
    private static final int INITIAL_SIZE = 36;

    // Amount to increase/decrease size on each click
    private static final int SIZE_INCREMENT = 4;

    // The text object the will be adjusted
    private Text example;

    private JButton startButton, stopButton;

    // Add other instance vars here
public void begin() {
    example = new Text("Banana", TEXT_LEFT, TEXT_TOP, canvas);
    example.setFontSize(INITIAL_SIZE);

    // Create buttons for selecting figures
    startButton = new JButton("Smaller");
    stopButton = new JButton("Bigger");

    JPanel buttonPanel = new JPanel();
    buttonPanel.add(startButton);
    buttonPanel.add(stopButton);

    Container contentPane = getContentPane();
    contentPane.add(buttonPanel, BorderLayout.SOUTH);
    contentPane.validate();
}
}