Introduction to the Assignment

In this assignment, you will get practice with recursion. There are three parts to the lab. The first two parts use recursive methods to draw interesting pictures. The last part uses a recursive structure.

Part 1: Recursive Tunnel

In this part, you will write a program that draws the following image:

![Recursive Tunnel Image]

The starting font is Verdana, bold, 60 points. The starting rectangle size is 600 pixels wide and 400 pixels tall. On each level of recursion, the width and height of the rectangle and the font size should all be 20% smaller. The recursion should stop when the height of the rectangle is 20 pixels or less.

Step 1: Create the window with the black border.
Step 2: Draw the outermost layer

Define a method with the following signature:

```java
private void paintOnce(int left, int top, int width, int height,
                        int fontSize, Graphics g)
```

This method should draw the white outline for the rectangle and the word RECURSION centered at the bottom of the rectangle.

To draw this image, you will need to define the paintComponent method and use the drawRect and drawString methods, like we did in the IdCard lab at the beginning of the semester. To center the string, you will need to determine its size at the given font size. You will need to call the setFont and getFontMetrics methods in the Graphics class, and then use the getStringBounds method in the FontMetrics class.

Call your paintOnce method from paintComponent. At this point, your window should look like this:

![New window with RECURSION centered at the bottom](image)

Step 3: Recursion

Define a method with the following signature:

```java
private void paintRecursively(int left, int top, int width, int height,
                               int fontSize, Graphics g)
```

Remember the basic principles of recursion:

- Define a base case. When should the recursion stop?
- How do you define a smaller subproblem?

Change paintComponent to call paintRecursively instead of paintOnce. Magic!
Part 2: Sierpinski’s Gasket

In this part, you will write a program that does another recursive drawing. Specifically, you will draw a triangle with smaller and smaller triangles inside of it, making a shape known as Sierpinski’s Gasket (http://en.wikipedia.org/wiki/Sierpinski_triangle).

Here are some snapshots of Sierpinski’s Gasket at various levels of development:

- This is the final shape that we want to draw.

- We start by drawing a single large triangle.

- Next, we find the midpoint of each line of the original triangle. Using these points, we draw the 3 upright triangles inside the big triangle. Note that the upside down triangle is not drawn explicitly.
Next, repeat the process with each of the triangles drawn in the previous stage.

Continue until the length of an edge of the triangle reaches some minimum value.

**Doing the Assignment**

**Step 1: Create a window with a white area to draw on**

**Step 2: Draw a single large triangle**

Now, write a method called `drawTriangles`. Eventually, this will become your recursive method, but for now just place code in it to draw a single triangle. You should pass it 3 `Point2D` objects that represent the corners of the triangle as well as the `Graphics` object that you will draw on.

`Point2D` is an abstract class in the Java libraries that encapsulates an x, y pair. We will use the `Point2D.Double` class that extends this abstract class:

```java
public Point2D.Double (double x, double y)
```

The triangles look best if they are equilateral (all edges have the same length), but that is not really necessary, so just pick three starting points so that the bottom line is horizontal.

You will need to make method calls that are defined in the `Point2D` abstract class as well as methods defined in the `Graphics` class to draw the triangles. Please refer to the online Java API to find methods to use.

At some point, you will likely discover that you need to convert from a double to an int. Remember that the way to do that is with casting:

```java
int myInt = (int) myDouble;
```

This causes Java to truncate the double to turn it into an int.
Call your `drawTriangles` method from `paintComponent` and you should see a single large triangle.

Hint: Your code will be easier to understand if you pick your variable names carefully. You will also find it helpful to write private methods to do some of the work to avoid repeating code within your `drawTriangles` methods.

**Step 3: Draw a triangle that recurses in one direction only**

Rather than just plunge into drawing the entire gasket, take it slowly and draw just one of the interior triangles when you recurse. For example, if you just recurse in the lower left direction, you will end up with something looking like the figure at the right.

Remember the basic principles of recursion:

- Define a base case. When should the recursion stop?
- How do you define a smaller subproblem? You need to call `drawTriangles` recursively, so you will need to determine where the corners of the interior triangles go. Notice that when we draw a triangle in the bottom left, its bottom left corner is the same and the other two corners are midpoints of the left and bottom edges of the larger triangle. The midpoint can be found by averaging the x value of the endpoints and the y value of the endpoints.

**Step 4: Draw the complete gasket**

Once you are successfully drawing the figure shown on this page, add more recursive calls to repeatedly draw triangles in each of the top and bottom right of the outer triangle.

**Part 3: Grab a Tiger by the Tail**

In this part, you will write several simple classes that allow the user to draw a tiger consisting of a head and a tail consisting of a number of tail parts. Then, if the user depresses the mouse on the end of the tail and drags, the tail will move around like a snake, pulling the head. Please look at the movie online at [http://www.mtholyoke.edu/~blerner/cs201/Labs/Lab10/Tiger.mov](http://www.mtholyoke.edu/~blerner/cs201/Labs/Lab10/Tiger.mov) for clarification. On the first mouse press, when the user drags the mouse, the tiger’s head is drawn followed by a series of small circles. On subsequent mouse presses, if the user clicks in the last circle, the tail moves such that the last circle follows the mouse and all previous circles follow the circle that succeeds them. The head moves to be centered on the location of the first circle in the tail.

**Doing the Assignment**

**Step 1: Create the head**

You will need a simple user interface for this application. You will need to create a class that extends `JComponent` and has a `MouseListener` and `MouseMotionListener` attached to it. `MouseListener`’s methods are called when the user presses or releases the mouse button:

```java
public void mousePressed (MouseEvent event)
public void mouseReleased (MouseEvent event)
```
MouseMotionListener’s methods are called when the user drags the mouse (that is, moves
the mouse while the mouse button is depressed).

    public void mouseDragged (MouseEvent event)

These interfaces have additional methods that you will need to declare, but you can leave
their bodies empty since you do not want to do anything in those cases:

    public void mouseClicked (MouseEvent event)
    public void mouseEntered (MouseEvent event)
    public void mouseExited (MouseEvent event)
    public void mouseMoved (MouseEvent event)

The tiger is a recursive structure consisting of a head and circles that form a tail. As with
Scribbler, you will need an interface, called TigerPart, and two implementing classes, called
TigerHead and TigerTailPart.

When the user first presses the mouse, create a TigerHead centered where the user presses
down the mouse button.

You can read the tiger head image from the file by calling:

    ImageIcon tigerHeadImage = new ImageIcon ("tigersmall.gif");

To display it, use the paintIcon method defined in ImageIcon. (Refer to the online Java API
for details.)

**Step 2: Create a tail**

As the user drags the mouse, you should create TigerTailPart objects. Each TigerTailPart has
an associated shape, which is an Ellipse2D.

The tail part should be created with its x and y values being where the mouse is when
mouseDragged is called.

This program differs from the Scribbler example in that your program only needs to draw a
single chain. After the user releases the mouse button, the program should enter "moving"
mode and interpret the mouse presses and drags as attempts to move the existing tiger,
rather than drawing additional tigers.

**Step 3: Moving the end of the tail**

The first step in moving the tiger is determining when the user has clicked on the end of the
tail.

Now that you can determine which link is the end of the tail, you can write a method to
move the end of the tail. Give this method 2 parameters to indicate the distance it should
move in the x & y directions. Initially, just show that you can drag the end of the tail
around, leaving the other parts of the tiger stationary.

**Step 4: Moving the tiger**

Modify the method that you wrote in step 3 so that it recursively moves the rest of the tiger.
Think about what parameters this method will need in order to move tiger parts to the right
places.

Magic! You should be able to drag your tiger around the screen!
### Grading

<table>
<thead>
<tr>
<th>Score</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Step 1: Create the window with black background</td>
</tr>
<tr>
<td>5</td>
<td>Step 2: Draw the outermost layer</td>
</tr>
<tr>
<td>10</td>
<td>Step 3: Recursion</td>
</tr>
<tr>
<td>5</td>
<td>Step 1: Create a window with a white area to draw on</td>
</tr>
<tr>
<td>5</td>
<td>Step 2: Draw a single large triangle</td>
</tr>
<tr>
<td>10</td>
<td>Step 3: Draw a triangle that recurses in one direction only</td>
</tr>
<tr>
<td>5</td>
<td>Step 4: Draw the complete gasket</td>
</tr>
<tr>
<td>5</td>
<td>Step 1: Create the head</td>
</tr>
<tr>
<td>5</td>
<td>Step 2: Clicking a tail</td>
</tr>
<tr>
<td>5</td>
<td>Step 3: Moving the end of the tail</td>
</tr>
<tr>
<td>5</td>
<td>Step 4: Moving the tiger</td>
</tr>
<tr>
<td>15</td>
<td>Comments</td>
</tr>
<tr>
<td>20</td>
<td>Style</td>
</tr>
</tbody>
</table>

### Turning in Your Work

Submit your source code as three separate jar files on Ella.