Introduction

In this lab, you will add visual paths for the workers. When the worker begins to navigate to a location, a series of circles will illuminate the path to be taken. As the workers walk along their path, the visual markers will disappear. The color of the path will also change based on how many workers are traveling along it. Finally, you will also create a FactoryProgressPanel, a panel that displays a progress bar for the factory.

Part 1 – Marking paths

We will want to keep track of when a path should be drawn on a FactoryNode. Add a private member variable in FactoryNode that acts as a counter. If the counter is above zero, it will indicate that a worker is taking the path.

    private int path;

Add two methods, mark() and unmark(), to increment and decrement this path variable respectively. **Do not forget to put 'synchronized' in the declaration, which you will learn about in the future.**

    public synchronized void mark() {
        path++;
    }

    public synchronized void unMark() {
        path--;
    }

Look inside the findShortestPath(factoryNode) method. In here, a Stack of FactoryNodes is returned.

    if(current.fNode == mDestinationNode) return makePath(start, current);

Instead of returning the Stack right away, increment each node’s path variable by calling the mark() method.

    if(current.fNode == mDestinationNode) {
        Stack<FactoryNode> path = makePath(start, current);
        Iterator<FactoryNode> iter = path.iterator();
        while(iter.hasNext()) {
            iter.next().mark();
        }
        return path;
    }

Note: Make sure you import java.util.Iterator, instead of any other Iterator that may be available.
Next, decrement this variable as the FactoryWorker arrives to the node. Alter the FactoryWorker’s update(double) method as shown in the sample below:

```java
if (mDestinationNode != null) {
    if (moveTowards(mNextNode, deltaTime * Constants.workerSpeed)) {
        if (!mShortestPath.isEmpty()) {
            // if we have somewhere else to go, save that location
            mNextNode = mShortestPath.pop();
            mCurrentNode.unMark();
        }
        // if we arrived at the location, signal the worker thread
        if (mCurrentNode == mDestinationNode) {
            mDestinationNode.unMark();
            atLocation.signal();
        }
    }
}
```

**Part 2 – Drawing paths**

Now we can draw the paths on the nodes. We will draw a circle in the center of each FactoryNode if its path value is greater than 0 and if there isn’t an object placed on the node. The draw(...) method is called automatically to render the node in the GUI.

```java
@Override
public void draw(Graphics g, Point mouseLocation) {
    g.setColor(Color.BLACK);
    if (mObject == null) {
        g.setColor(Color.WHITE); // draw a border, makes it look like a tiled grid in the factory
        g.drawRect(renderBounds.x, renderBounds.y, renderBounds.width, renderBounds.height);
        if (path > 0) {
            g.fillOval(renderBounds.x + renderBounds.width/4, // left side 25% in
                        renderBounds.y + renderBounds.height/4, // top 25% in
                        renderBounds.width/2, // take up half the grid
                        renderBounds.height/2); // take up half the grid
        }
    } else {
        mObject.draw(g, mouseLocation);
    }
}
```

If we run the factory, it should look similar to the sample below:

![Computer Factory Image]
Part 3 – Adding Color

The white dots are a good indicator, but it would be nice to have a visualization of traffic with different colors. Add the following array of colors in the FactoryNode class.

```java
private static final Color[] pathColors = {
    new Color(0,0,255),
    new Color(0,128,255),
    new Color(0,255,255),
    new Color(0,255,128),
    new Color(0,255,0),
    new Color(128,255,0),
    new Color(255,255,0),
    new Color(255,128,0),
    new Color(255,0,0)
};
```

*Note: The values passed into the color constructor are Red, Green, Blue values. Based on how many workers are following the path we can choose a color based on a smooth gradient from 1-blue to 9+-red.*

Now, in the FactoryNode.draw(Graphics) method, set the color based on how many workers are currently following the path.

```java
int colorCode = path-1;
if(colorCode >= pathColors.length) colorCode = pathColors.length-1;
g.setColor(pathColors[colorCode]);
```

*Note: Make sure to offset the value by 1, since 0 should not have any color.*

Now if you run the FactoryClient, there should be colorful paths!
Part 4 – The FactoryProgressPanel

Create a new class named FactoryProgressPanel that extends JPanel. We will use this panel to draw a custom progress bar for the factory.

This class should keep track of the JTable of the Factory, so pass a JTable into the constructor.

```java
public class FactoryProgressPanel extends JPanel{
    private static final long serialVersionUID = -387552977080137779L;

    private JTable mTable;
    private TableModel mModel;

    FactoryProgressPanel(JTable inTable) {
        mTable = inTable;
        mModel = mTable.getModel();
    }
}
```

Since we want to draw onto this Panel, override the paintComponent(Graphics) method.

```java
@Override
protected void paintComponent(Graphics g) {
}
```

First, make sure the parent paintComponent(Graphics) method is called. This will draw all the essential parts of a normal JPanel.

```java
@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
}
```

Let’s put a title onto the Panel. To do this, we will use g.drawString(String,int,int). By default, however, the Font will be too small. Let’s create a new font that scales with the size of the window.

First, get the dimensions of the Panel we are drawing onto.

```java
int w = this.getWidth();
int h = this.getHeight();
```

Then, create a font that has a width of 1/24\textsuperscript{th} the size of the panel per character.

*Note: The choice of $1/24^{th}$ is arbitrary.*

Also, make the font Bold and Italic by passing in the appropriate flags to the constructor.

```java
Font font = new Font("Times New Roman", Font.BOLD|Font.ITALIC, w/24);
g.setFont(font);
```

Now that we have a font, we want to draw a String at the center of the panel. We must find the dimensions of the font in order to center it correctly.
Make a title member variable:

```java
private final String title = "Factory Progress";
```

```java
int strwidth = g.getFontMetrics(font).stringWidth(title);
g.drawString(title, (w - strwidth) / 2, g.getFontMetrics(font).getHeight());
```

**Note:** The `drawString()` method starts to draw at the bottom-left corner instead of the more traditional top-left corner. This is why we need to lower the y-value by its own height.

**Note:** (0,0) is the top-left corner. Higher y-values are drawn near the bottom of the panel, and higher x-values are drawn near the right of the panel.

In order to draw a progress bar, we need the JTable’s statistics. Let’s collect these from the `TableModel`.

```java
double total = 0;
double started = 0;
double completed = 0;
for(int i = 0; i < mModel.getRowCount(); ++i) {
    total += (int)mModel.getValueAt(i, Constants.totalNameIndex);
    started += (int)mModel.getValueAt(i, Constants.startedIndex);
    completed += (int)mModel.getValueAt(i, Constants.completedIndex);
}
```

**Note:** We are using doubles so we can make more accurate calculations later.

We are going to calculate a frame for the progress bar. First, create a constant border size of 20 pixels.

```java
final int border = 20;
```

Then calculate the dimensions of the frame of the progress bar.

```java
int frameX = border;
int frameY = border+g.getFontMetrics(font).getHeight();
int frameW = w-border-border;
int frameH = h-border-border-g.getFontMetrics(font).getHeight();
```

Use these values to draw a black border.

```java
g.drawRect(frameX-1, frameY-1, frameW+1, frameH+1);
```

Offset the values by 1 so they do not get covered up by the actual bar. Finally, we are going to calculate the positions of the progress bar, and draw them.

```java
int startedWidth = (int) ((started/total)*frameW);
int completedWidth = (int) ((completed/total)*frameW);
int nostatusWidth = frameW-(startedWidth+completedWidth);
```

```java
g.setColor(Color.GREEN);
g.fillRect(frameX, frameY, completedWidth, frameH);
g.setColor(Color.YELLOW);
g.fillRect(frameX + completedWidth, frameY, startedWidth, frameH);
g.setColor(Color.RED);
g.fillRect(frameX + startedWidth + completedWidth, frameY, nostatusWidth, frameH);
```
Part 5 – Adding the panel

Now that the FactoryProgressPanel is complete, we can add it to the FactoryClientGUI.

Let’s place it where the controller used to be.

```java
private void createGUI() {
    setSize(Constants.factoryGUIwidth, Constants.factoryGUIheight);
    setLayout(new BorderLayout());
    JScrollPane messageTextAreaScrollPane = new JScrollPane(messageTextArea);

    Box bottomBox = Box.createHorizontalBox();
    bottomBox.add(messageTextAreaScrollPane);
    bottomBox.add(new FactoryProgressPanel(productTable));

    add(factoryPanel, BorderLayout.CENTER);
    add(bottomBox, BorderLayout.SOUTH);
    add(tableScrollPane, BorderLayout.EAST);
}
```

Lastly we need the panel to update itself. Otherwise, nothing will be drawn to the screen.

We could update the Panel from the Workers or the Table, but those classes shouldn’t have to know about the existence of this class so that it can be updated.

Instead, let’s create a thread that updates itself every 33ms, or 30 frames per second (fps).

*Note: You haven’t learned about threads yet – just copy and paste this code.*

Have the FactoryProgressPanel implement the Runnable interface.

```java
public class FactoryProgressPanel extends JPanel implements Runnable
```

Modify the FactoryProgressPanel constructor.

```java
FactoryProgressPanel(JTable inTable) {
    mTable = inTable;
    mModel = mTable.getModel();
    new Thread(this).start();
}
```

And lastly, add this run() method:

```java
@Override
public void run() {
    while(true) {
        try {
            Thread.sleep(33);
            repaint();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```
Now the panel should draw and update itself.
Expand on This

The FactoryProgressPanel could be a bit confusing to a user. It would be helpful to display the actual percentage inside the bar. Add a String displaying the percentage of that product's status in comparison to the total products centered within the areas of the bars. If the bar is 0% do not display the percentage.

<table>
<thead>
<tr>
<th></th>
<th>24%</th>
<th>24%</th>
<th>52%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completed</td>
<td>Started</td>
<td>Not Started</td>
<td></td>
</tr>
</tbody>
</table>

Hints:

1. You can do this all within FactoryProgressPanel.java by adding to the code written in this lab.
2. You might need to re-size and recolor the font before you print it.
3. Check out the syntax for g.drawString(...). Make sure to take note of what it takes as parameters.

Remember, there is no one right way of doing this. Be creative and ask your CP if you need help.