In this lab, you will create a FactoryController, which will be a separate JFrame that holds various changeable run-time options for the factory simulation. You will move the JSlider, currently on the bottom right of the factory window, to this new FactoryController window. You will also add tabs to the FactoryController so that it can handle any future additions. Finally, you will create a pop-up when the factory ends that displays the total elapsed time.

**Part 1 – Creating a new JFrame**

The factory already has a JSlider that allows control over the speed at which the factory runs. However, a user usually only modifies the speed a few times, if at all, during a simulation. Currently, the JSlider eats up a lot of space in the GUI, so let’s instead create a new JFrame to contain the JSlider.

First, create a new class called FactoryController that extends JFrame. Since this class is only used by the FactoryClientGUI and won’t have any use outside of that, let’s create it as an inner class of FactoryClientGUI. Do not make it a separate top-level class.

```java
class FactoryController extends JFrame{
    private static final long serialVersionUID = 7573324399771995953L;

    public FactoryController() {
        super("Factory Controller");
        setSize(320,240);
        add(simulationSpeedController);
        setVisible(true);
        setLocationRelativeTo(null);
    }
}
```

*Note: setLocationRelativeTo(Component c); sets the location of the JFrame with respect to another Component. Passing null to this centers the JFrame on the screen.*

Now create a new instance of the FactoryController in our FactoryClientGUI class.

```java
private FactoryController factoryController;
```
private void initializeVariables() {
    ...
    factoryController = new FactoryController();
}

Don’t forget to remove the simulationSpeedController from the previous GUI or it won’t show up on our new FactoryController.

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

    Box bottomBox = Box.createHorizontalBox();
    bottomBox.add(messageTextAreaScrollPane);
    //bottomBox.add(simulationSpeedController);

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

When you start the FactoryClient, you should see the following:

Note: The FactoryController will appear behind the FactoryClientGUI. This is fine for now! We are going to make some more changes that will fix this in part 2.
Part 2 – Creating a menu

We want the FactoryController to only display when the user requests it. We can achieve this by adding a simple menu to the FactoryClientGUI.

This menu should only have one JMenuItem, labeled “Controller”. Once clicked, the FactoryController should be shown.

Create this new menu in a separate private method. Call this method in the FactoryClientGUI constructor.

```java
private void createMenu() {
    JMenuBar menu = new JMenuBar();
    JMenuItem controller = new JMenuItem("Controller");
    controller.addActionListener(new ActionListener(){
        @Override
        public void actionPerformed(ActionEvent arg0) {
            factoryController.setVisible(true);
        }
    });
    menu.add(controller);
    setJMenuBar(menu);
}
```

Now that we have a way to open the FactoryController, set the initial visibility to be false in its constructor.

```java
public FactoryController() {
    super("Factory Controller");
    setSize(320,240);
    add(simulationSpeedController);
    setVisible(false);
    setLocationRelativeTo(null);
}
```

Now when the menu item is clicked, the FactoryController will appear. When the controller’s window is closed out by clicking the “X”, the window will be hidden. This is the default behavior of the “X” on a JFrame.
Part 3 – Controller GUI

Our controller doesn’t look very good. There aren’t any labels, so it may be difficult for users to know where to move the slider.

In the FactoryController constructor, create a Dictionary to map Integers to JLabels.

```java
Dictionary<Integer, JLabel> labelTable = new Hashtable<Integer, JLabel>();
```

In this Dictionary, add each simulation speed along with a description for each.

```java
labelTable.put(FactoryController.Constants.simulation_0x, new JLabel("Paused"));
labelTable.put(FactoryController.Constants.simulation_1x, new JLabel("Normal"));
labelTable.put(FactoryController.Constants.simulation_2x, new JLabel("Double"));
labelTable.put(FactoryController.Constants.simulation_3x, new JLabel("Triple"));
```

Finally, set the simulationSpeedController's labelTable to this new Dictionary, and enable the labels to be painted.

```java
simulationSpeedController.setLabelTable(labelTable);
simulationSpeedController.setPaintLabels(true);
```

The FactoryController should now look like this:

![Factory Controller GUI](image)

Let’s make the Controller look a bit nicer. Add a TitledBorder around the slider, and move it to the bottom.

```java
simulationSpeedController.setBorder(new TitledBorder("Speed Controller"));
add(simulationSpeedController, BorderLayout.SOUTH);
```
The Controller should look like this now:

![Factory Controller](image)

Let's add some buttons that give some more control to the factory. We will add buttons to pause, continue, and restart the factory.

Since there are only three buttons, let's use the rest of the space for them. We will use GridBagLayout for these buttons.

```java
JPanel buttonBox = new JPanel();
buttonBox.setLayout(new GridBagLayout());
GridBagConstraints gbc = new GridBagConstraints();
```

Now, set the GridBagConstraints.

Since buttons can vary in length, we want each of the components to take up as much horizontal space as they can. We can enable this by setting the fill to horizontal.

```java
gbc.fill = GridBagConstraints.HORIZONTAL;
```

Also, we want the buttons to be spaced out. Add weight in the y-direction for each component so they try to take up as much space as possible.

```java
gbc.weighty = 1;
```
As each component is added, change the y location of each.

```java
JButton pauseButton = new JButton("Pause");
gbc.gridy = 1;
buttonBox.add(pauseButton, gbc);

JButton continueButton = new JButton("Continue");
gbc.gridy = 2;
buttonBox.add(continueButton, gbc);

JButton resetButton = new JButton("Reset");
gbc.gridy = 3;
buttonBox.add(resetButton, gbc);
```

Finally, add the panel to the JFrame.

```java
add(buttonBox);
```

*Note: The buttons won’t do anything yet. Their actions will be implemented in another lab.*

This menu looks pretty good for controlling time in the simulation, but we will probably want more controls in the future. Let’s create two tabs – one for this menu, and one for later use.

Add a private JTabbedPane member variable to the FactoryController class.

```java
private JTabbedPane tabbedPane;
```

Create some helper methods to add new tabs in the constructor since each tab can require a lot of code to make. Migrate the code needed to make the three buttons and the JSlider into createTimePanel(). Remember to make sure the JPanel is the one adding the internal JPanels.
private void createTimePanel() {
    JPanel timePanel = new JPanel();
    timePanel.setLayout(new BorderLayout());
    timePanel.add(buttonBox);
    timePanel.add(simulationSpeedController, BorderLayout.SOUTH);
    tabbedPane.add("Time", timePanel);
}

private void createOtherPanel() {
    JPanel otherPanel = new JPanel();
    tabbedPane.add("Other", otherPanel);
}

public FactoryController() {
    super("Factory Controller");
    setSize(320, 240);
    tabbedPane = new JTabbedPane();

    createTimePanel();
    createOtherPanel();

    add(tabbedPane);

    setVisible(false);
    setLocationRelativeTo(null);
}

The final window should look like this:
Part 4 – End of simulation pop-up

Add a private double member variable called totalTime to the FactorySimulation class.

```java
private double totalTime = 0.0;
```

In the update(double) method, add deltaTime to totalTime.

```java
totalTime += deltaTime;
```

When the factory ends, create a pop-up window that displays the amount of seconds that passed during the simulation.

```java
JOptionPane.showMessageDialog(null,
    "Total time: " + totalTime + "s",
    "Simulation Over!",
    JOptionPane.INFORMATION_MESSAGE);
```

*Note: When the double is converted to a string, it will not be formatted neatly. You can use a DecimalFormat to format the double. The example shown below will print the time with three decimal places.*

```java
DecimalFormat threePlaces = new DecimalFormat(".###");
```

```java
JOptionPane.showMessageDialog(null,
    "Total time: " + threePlaces.format(totalTime) + "s",
    "Simulation Over!",
    JOptionPane.INFORMATION_MESSAGE);
```

*Note: Since we used delta time, the time from the simulation will be the total time in terms of factory time! Real time will only be reported if the factory is run at normal speed.*
Part 5 – Questions

Answer the following questions to see if you understood some of the theory used in this lab.

1. Name four different layout managers in Java.
2. Explain the behavior of these layout managers.
3. Is the preferred height and width of the components acknowledged by each layout manager?
Expand on This

Let's add another piece of functionality to the final pop-up window. In the previous lab, you (should have) created a way for the factory reports and completion reports to be saved into files after the factory is done running. Since this could potentially start spamming your hard drive with a bunch of useless files, let's prompt the user to save one or both of the files after the factory is done. It can be displayed at the same time as the “Simulation Over!” window.

Before anything happens, let's delete all of these files prior to doing this. That way we'll know if the expanded program auto-saved the files or not.

Since this part of the lab is open for you to do, there won't be much direct tutorial here. However, we have some hints to get you going:

1. **Find out** how to get buttons in the JOptionPane to call some code. Once that is done, it is all plug and play from there. This is a good place to use anonymous inner classes if applicable.

2. It is highly recommended that you find a way to be able to call the code that saves the files outside of the FactorySimulation's update() method.

3. Also, don't forget to close the window once you have clicked some button by setting the visibility to false.

Remember, there is no one right way of doing this. Be creative, and don't be afraid to talk to your CP. Good luck!