THE UNIVERSITY OF WESTERN ONTARIO

Computer Science 2212b - Winter 2014
Introduction to Software Engineering

Lab 4 - GUI Development with Swing

Part I
Introduction to Swing

Estimated Time: 45-60 minutes

1 Overview of Swing

Swing is a toolkit for creating Java desktop applications that have a graphical user interface (GUI). It is bundled with Java, and therefore provides a simple way of creating a cross-platform GUI application. Swing is a *lightweight* toolkit, meaning that all of its components (e.g. buttons, menus, etc.) are implemented in Java code, rather than calling the operating system to use its native components. This decreases the chance that multiplatform issues may arise, since no operating system-specific code is present in Swing, but also means that the components look somewhat foreign at times, in comparison to native GUI components.

In this lab, we will build a simple GUI application to learn the basics of Swing.

2 Getting Started

1. Change to your individual Git repository, create a `lab4` directory, and change to it:

```
$ cd ~/courses/cs2212/labs
$ mkdir lab4
$ cd lab4
```

2. Create a file `pom.xml` with the following elements:

   - `groupId`: `ca.uwo.csd.cs2212.USERNAME`
   - `artifactId`: `USERNAME-lab4`
   - `version`: `1.0-SNAPSHOT`

   Refer to lab 2 for the full details of creating a `pom.xml` (remember that you will also need a `modelVersion` tag). Of course, `USERNAME` should be replaced with your UWO username in **lower case**.

3. Edit the `pom.xml` file so that the JAR file that it produces is executable (refer to lab 2). Set the main class to `ca.uwo.csd.cs2212.USERNAME.App`.

4. Create the directory structure to host our project:

```
$ mkdir -p src/{main,test}/{java,resources}
```

5. Create the directory structure to house your Java package:

```
$ mkdir -p src/main/java/ca/uwo/csd/cs2212/USERNAME
```

   Again, `USERNAME` should be your UWO username in **lowercase**.

3 Creating a Window

1. Create a file `src/main/java/ca/uwo/csd/cs2212/USERNAME/App.java` with the following code. You can copy/paste it from https://gist.github.com/jsuwo/8877749#file-app-java:

```
package ca.uwo.csd.cs2212.USERNAME;

import javax.swing.SwingUtilities;

public class App {

  public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
      @Override
      public void run() {
        MainWindow window = new MainWindow();
        window.setVisible(true);
      }
    });
  }
}
```

   This method simply instantiates a `MainWindow` object (we will write this class next) and displays the window. The other details in the method are beyond the scope of this lab, but, in a nutshell, they ensure that the window is created on a separate thread to avoid tying up the current thread.

2. Create a file `src/main/java/ca/uwo/csd/cs2212/USERNAME/MainWindow.java` with the following code. You can copy/paste it from https://gist.github.com/jsuwo/8877749#file-mainwindow1-java:

```
package ca.uwo.csd.cs2212.USERNAME;

import javax.swing.JFrame;

public class MainWindow extends JFrame {

  public MainWindow() {
    this.initUI();
  }

  private void initUI() {
    this.setTitle("CS 2212 - Lab 4");
    this.setSize(800, 200);
    this.setLocationRelativeTo(null);
    this.setDefaultCloseOperation(EXIT_ON_CLOSE);
  }

}
```

   This class inherits from `JFrame`, which is a top-level Swing *container* – a component on which other components can be placed. The constructor calls the `initUI` method, which sets the title and size of the window, ensures that it is centered on the screen, and sets it to exit the program when the window is closed.

3. Package your code as a JAR file and run the JAR:

```
$ mvn package
$ java -jar target/jshantz4-lab3-1.0-SNAPSHOT.jar
```

   You should see a small window appear with the title you specified. Not very exciting, but it's a start!

4 Adding a Menu

Let's add a **File** menu to the window, with an **Exit** menu item that exits the application when clicked.

1. Edit `MainWindow.java` and add the following code. You can copy/paste the additions from https://gist.github.com/jsuwo/8877749#file-mainwindow2-java

```
.
.
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import javax.swing.JMenu;
```

```
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;

public class MainWindow extends JFrame {

  .
  .

  private void initUI() {
    .
    .
    this.setJMenuBar(this.createMenubar());
  }

  private JMenuBar createMenubar() {

    JMenuBar menubar = new JMenuBar();

    JMenu mnuFile = new JMenu("File");
    mnuFile.setMnemonic(KeyEvent.VK_F);

    JMenuItem mniFileExit = new JMenuItem("Exit");
    mniFileExit.setMnemonic(KeyEvent.VK_E);
    mniFileExit.setToolTipText("Exit application");
    mniFileExit.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent event) {
        System.exit(0);
      }
    });

    mnuFile.add(mniFileExit);
    menubar.add(mnuFile);

    return menubar;
  }
}
```

The `createMenubar` method creates and returns the menu bar, which is then added to the window in `initUI` using `JFrame`'s `setJMenuBar` method.

Notice that we use the `setMnemonic` method when creating `mnuFile` and `mniFileExit`. These set the keyboard mnemonics that allow power users to more easily navigate menus using the keyboard. For instance, on a Windows system, we could access the **File** menu quickly by pressing `Alt-F`, since we set its mnemonic to `KeyEvent.VK_F`.

Finally, examine the `addActionListener` call that is made on `mniFileExit`. This creates an instance of the `ActionListener` class, whose `actionPerformed` method will be called when the menu item is clicked. We could create the class elsewhere and simply instantiate an object here and pass it to `addActionListener`. However, that would be a lot of work for a simple menu item, so we are creating an *anonymous* class here: we both define the class and instantiate it in the same place[1].

2. Package your code as a JAR file and run the JAR.
3. Select the **File** menu and hover your mouse over the **Exit** menu item. Notice that a tool tip appears.
4. Click the **Exit** menu item and the program should exit.

5 Adding Some Controls

A *control* (also known as a *component* or a *widget*) is a reuseable GUI element that can be placed in a window. Examples include text boxes, check boxes, radio buttons, labels, etc. In this section, we'll add a few controls to our window.

1. Edit `MainWindow.java` and add the following code. You can copy/paste the additions from https://gist.github.com/jsuwo/8877749#file-mainwindow3-java:

[1]For more information on anonymous classes, see http://docs.oracle.com/javase/tutorial/java/javaOO/anonymousclasses.html.

```
.
.
import java.awt.Dimension;
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;
import javax.swing.ButtonGroup;
import javax.swing.JRadioButton;

public class MainWindow extends JFrame {

  private JTextField txtName;
  private JLabel lblGreeting;
  private JRadioButton radBachelors;
  private JRadioButton radMasters;
  private JRadioButton radPhd;

  public MainWindow() {
    this.initUI();
  }

  private void initUI() {
    .
    .
    this.add(this.createForm());
  }

  private JPanel createForm() {

    JPanel panel = new JPanel();
    JLabel lblName = new JLabel("Name:");
    JLabel lblDegree = new JLabel("Degree:");

    txtName = new JTextField();
    txtName.setPreferredSize(new Dimension(75,25));

    radBachelors = new JRadioButton("BSc");
    radMasters = new JRadioButton("MSc");
    radPhd = new JRadioButton("PhD");

    ButtonGroup grpDegree = new ButtonGroup();
    grpDegree.add(this.radBachelors);
    grpDegree.add(this.radMasters);
    grpDegree.add(this.radPhd);

    lblGreeting = new JLabel();

    JButton btnGreet = new JButton("Greet Me");

    btnGreet.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent event) {

        String msg = "Hello " + txtName.getText() + ", you are working on ";

        if (radBachelors.isSelected())
          msg += "your BSc.";
        else if (radMasters.isSelected())
          msg += "your MSc.";
        else if (radPhd.isSelected())
          msg += "your PhD.";
        else
          msg += "absolutely nothing.";

        lblGreeting.setText(msg);
      }
    });

    panel.add(lblName);
    panel.add(txtName);
    panel.add(lblDegree);
    panel.add(radBachelors);
    panel.add(radMasters);
    panel.add(radPhd);
    panel.add(btnGreet);
    panel.add(lblGreeting);


    return panel;
  }
}
```

The `createForm` method is quite large and we might like to break it up in practice, but we'll just leave it the

way it is for the purpose of this lab. The method begins by creating a `JPanel`, which will have all of the form controls added to it.

The creation of the form controls is relatively straightforward. Notice that an `ActionListener` is added to `btnGreet` to change the value of `lblGreeting` based on the user's input.

Finally, all of the controls are added to the `JPanel`, and the panel is returned to the `initUI` method where it will be added to the form.

2. Package your code as a JAR file and run the JAR.
3. Enter something in the **Name** field, and select a **Degree**. Click **Greet Me**. A greeting should appear.

We've seen a few basic controls, but they aren't arranged very nicely, are they? We'll fix that right up in the next section.

6 Layout Managers: `GroupLayout`

Swing provides a number of *layout managers* that take care of automatically arranging controls within a container (e.g. `JPanel`, `JFrame`, etc). Notice in the previous section that we used a `JPanel` to contain the form controls. Grouping controls into panels allows us to set different layouts on the various panels on the screen as needed, rather than having one layout for the entire window. Let's clean things up a bit.

1. Edit `MainWindow.java` and add the following code. You can copy/paste the additions from https://gist.github.com/jsuwo/8877749#file-mainwindow4-java:

```
.
.
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import javax.swing.GroupLayout;

public class MainWindow extends JFrame {

  .
  .

  private JPanel createForm() {

    .
    .

    GroupLayout layout = new GroupLayout(panel);

    layout.setAutoCreateGaps(true);
    layout.setAutoCreateContainerGaps(true);

    layout.setHorizontalGroup(
      layout.createSequentialGroup()
        .addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
          .addGroup(layout.createSequentialGroup()
            .addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
              .addComponent(lblName)
              .addComponent(lblDegree)
            )
            .addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
              .addComponent(txtName)
              .addGroup(layout.createSequentialGroup()
                .addComponent(radBachelors)
                .addComponent(radMasters)
                .addComponent(radPhd)
              )
            )
          )
          .addComponent(lblGreeting)
        )
        .addComponent(btnGreet)
    );
```

```
    layout.setVerticalGroup(
      layout.createSequentialGroup()
        .addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)
          .addComponent(lblName)
          .addComponent(txtName))
        .addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)
          .addComponent(lblDegree)
          .addComponent(radBachelors)
          .addComponent(radMasters)
          .addComponent(radPhd))
        .addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)
          .addComponent(lblGreeting)
          .addComponent(btnGreet))
    );

    // Replace all the panel.add(...) statements with the following statement
    panel.setLayout(layout);

    return panel;
  }
}
```

In the `createForm` method, be sure to remove the `panel.add(...)` statements from the previous example, replacing them with the code above. Here, we use the popular Java layout manager `GroupLayout`. Generally, a GUI designer would generate this code for us, but it's a good idea to understand how `GroupLayout` works since one does have to tweak designer-generated code from time to time.

In `GroupLayout`, we manage the horizontal and vertical layouts of our controls separately. For instance, examine the `setHorizontalGroup` statement above. With this statement, we are dividing our form into columns. The first column will contain `lblName` and `lblDegree`; the second column will contain `txtName`, along with all the radio buttons; and so on.

Similarly, in the `setVerticalGroup` statement above, we are dividing our form into rows. The first row will contain `lblName` and `txtName`; the second row will contain `lblDegree` and the radio buttons; and so on.

For more information on the `GroupLayout` layout manager, see
http://docs.oracle.com/javase/tutorial/uiswing/layout/group.html.

2. Package your code as a JAR file and run the JAR.
3. Try resizing the window and notice that the layout manager takes care of resizing the controls accordingly.

7 Layout Managers: `BorderLayout`

Another commonly used layout manager is `BorderLayout`, which divides a container into `NORTH`, `EAST`, `SOUTH`, `WEST`, and `CENTER` regions. In this section, we'll briefly look at how it is used.

1. Edit `MainWindow.java` and add the following code. You can copy/paste the additions from
https://gist.github.com/jsuwo/8877749#file-mainwindow5-java:

```
.
.
import java.awt.Color;
import java.awt.BorderLayout;

public class MainWindow extends JFrame {

  .
  .

  private void initUI() {

    // Replace the line "this.add(this.createForm());" with the following:

    JPanel pnlWest = new JPanel();
    pnlWest.setBackground(Color.BLUE);
```

```
    pnlWest.setPreferredSize(new Dimension(100,200));

    JPanel pnlEast = new JPanel();
    pnlEast.setBackground(Color.GREEN);
    pnlEast.setPreferredSize(new Dimension(100,200));

    JPanel pnlNorth = new JPanel();
    pnlNorth.setBackground(Color.ORANGE);
    pnlNorth.setPreferredSize(new Dimension(300,25));

    JPanel pnlSouth = new JPanel();
    pnlSouth.setBackground(Color.RED);
    pnlSouth.setPreferredSize(new Dimension(300,25));

    this.setLayout(new BorderLayout());

    this.add(this.createForm(), BorderLayout.CENTER);
    this.add(pnlEast, BorderLayout.EAST);
    this.add(pnlWest, BorderLayout.WEST);
    this.add(pnlNorth, BorderLayout.NORTH);
    this.add(pnlSouth, BorderLayout.SOUTH);
  }
}
```

In the code above, we simply create four empty panels for the sake of demonstrating the layout manager. In practice, we might add a `JToolBar` in the `NORTH` position, and we might use a `JPanel` in the `SOUTH` position as a status bar.

Notice with `BorderLayout` that when we add a control to a container, we specify the location in which the control should be added as the second argument to `add` (e.g. `BorderLayout.CENTER`).

2. Package your code as a JAR file and run the JAR.

8 Loading Resources

Often, we'll want to load resources such as images or icons and display them in our applications. In this section, we'll see how to store an image in our application's JAR file using Maven, and load it from the JAR at run-time.

1. Go to http://www.iconfinder.com
2. Search for the term `exit`.
3. Narrow the search to free icons.
4. Drag the slider near the top of the screen to show only icons **16 px** in size.
5. Download a free icon in PNG format.
6. Copy the downloaded icon to the `src/main/resources` directory. Name it `exit.png`. This is all we need to do to have Maven add the image to our JAR file – *convention over configuration!*
7. Repeat the process, but search for a smiley face. Name it `smile.png`.
8. Edit `MainWindow.java` and add the following code. You can copy/paste the additions from https://gist.github.com/jsuwo/8877749#file-mainwindow6-java:

```
.
.
import javax.swing.ImageIcon;
import javax.imageio.ImageIO;
import java.io.InputStream;

public class MainWindow extends JFrame {

  private final String ICON_EXIT = "exit.png";

  .
  .

  private JMenuBar createMenubar() {
```

```
    .
    .

    JMenuItem mniFileExit = new JMenuItem("Exit", loadIcon(ICON_EXIT));

    .
    .
  }

  private ImageIcon loadIcon(String filename) {

    try {
      InputStream in = MainWindow.class.getClassLoader().getResourceAsStream(filename);
      return new ImageIcon(ImageIO.read(in));
    }
    catch (Exception ex) {
      return null;
    }
  }

  .
  .
}
```

In the `loadIcon` method, we ask the current class' class loader to load the specified resource as a stream, which we then read in using `ImageIO.read`. The result is used to initialize a new `ImageIcon`.

Finally, we pass the resulting icon as the second argument to the `JMenuItem` constructor.

9. Repeat the process for `smile.png`. Add it to the `JButton` `btnGreet`. `JButton` also provides an overloaded constructor that accepts an `ImageIcon` as its second parameter.
10. Package your code as a JAR file and run the JAR. You should see your exit icon when you select the **File** menu. Similarly, your smile icon should appear on the **Greet Me** button.

9 Swing Resources

We've barely scratched the surface of Swing. There are many more components available to you, many of which are described here: http://docs.oracle.com/javase/tutorial/uiswing/components/index.html.

Some decent Swing tutorials include:

- http://docs.oracle.com/javase/tutorial/uiswing/index.html
- http://www.zetcode.com/tutorials/javaswingtutorial/

Finally, be sure to consult the `javax.swing` API documentation:
http://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html.

10 Submitting Your Lab

- Take a screenshot of your running application.
- Name the screenshot `lab4.png` and place it in your `lab4` directory.
- Commit your code and push to GitHub.
- To submit your lab, create the tag `lab4` and push it to GitHub. For a reminder on this process, see Lab 1.

Note that, for this lab, due to the difficulty in automatically testing GUI programs (it is possible, but more difficult), the automarker will not actually be running any tests on your lab. Instead, it will ensure your code compiles, and will then automatically mark your lab as complete. We will then later test your lab and will notify you if there are any problems with your lab that might require resubmission.