INTRPRG Lab Activity Manual

Laboratory Activity Manual

Shirley Chu

ABSTRACT

INTRPRG Laboratory Activity Manual …

Description of the Sections

The lab activity manual consists of different sections classified based on where and when the activity should be done. **Prelab activities** are done after class discussion using paper and pen only. **Lab activities** are normally, programming exercises expected to be answered in front of a computer. **Postlab activities** are done after completing the previous lab activity, not in front of a computer, to check whether the student was able to learn from the previous lab activity.

Reminders

When doing these activities, kindly remember the following:

1. BE ORGANIZED – Create a folder where you will be placing all the program files that you've created for the activities in this manual.
2. BE SURE – Email a copy of your file to yourself.
3. BE DILIGENT – Go through each and every the activity in this manual.
4. BE PATIENT – It is normal for you to encounter errors when programming, patiently debug each error.

Patience is a virtue. Virtue is not achieved in one day, but through years of hard work.

Check List

ACTIVITY	DATE STARTED	DATE FINISHED
☐ Lab Exercise 8 – Creating a Simple GUI		
☐ Lab Exercise 9 – Numeric Keypad Application		

Lab Activity

Lab Exercise 8 – Creating a Simple GUI

Name: _____ Date: _____

Section: _____

Lab Objectives

This lab activity gives the students a walkthrough on how to create a simple GUI.

Description of the Activity

In this activity, the students will be guided on how to create a simple GUI using Swing.

Step 1. Create a file, name it ASimpleGUI.java.

Step 2. Code the basic skeleton of a Java Application.

```
public class ASimpleGUI
{
   public static void main (String[] args)
   {
      ASimpleGUI app = new ASimpleGUI ();  //starts the java application
   }
}
```

Step 3. To create an application window, we use **`javax.swing.JFrame`**. Modify your code.

- Include the **`import`** statement for the **`javax.swing`** package,
- Extend the **`JFrame`** class,
- Create a constructor for your class,
- Call the constructor of the superclass (**`JFrame`**), and set the title of your window,
- Determine the action that will be performed when the window is closed, i.e. terminate the program.

```
import javax.swing.*;
public class ASimpleGUI extends JFrame
{  // the no-parameter constructor of ASimpleGUI
   public ASimpleGUI ()
   {
      super ("A Simple GUI Application");  // call to JFrame's constructor
                                           // param = title of your window
      // terminates the application when window is closed
      // EXIT_ON_CLOSE is a constant defined inside JFrame. Since
      // ASimpleGUI is an extension of JFrame, the constant can be
```

```
      // accessed directly.

      setDefaultCloseOperation (EXIT_ON_CLOSE);

      setSize (400, 200);  // sets the width and height of the component
      setVisible (true);   // must always be the last statement
                           // This makes the window visible.
   }
   public static void main (String[] args)
   {  ASimpleGUI app = new ASimpleGUI ();  // starts the java application
   }
}
```

Step 4. Compile and run your program. You will see a window similar to Figure 1 below.

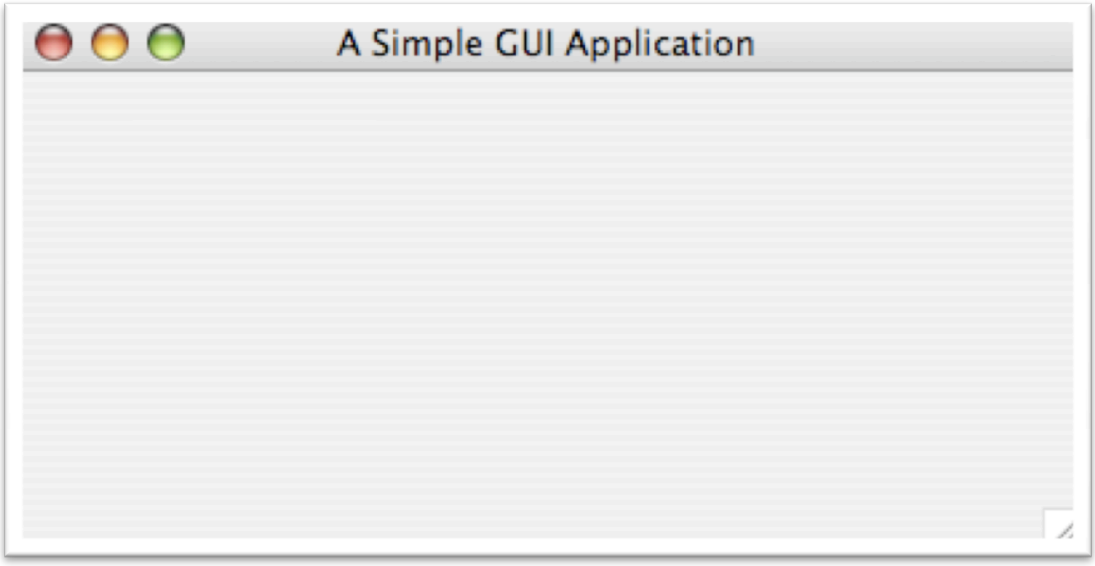

Figure 1. ASimpleGUI Window

Step 5. Determine the layout manager for your **`JFrame`**. The default layout manager for **`JFrame`** is **`BorderLayout`**. For the purpose of simplifying this tutorial, we will change the layout manager to **`FlowLayout`**. This layout manager arranges the components in a directional flow, similar to lines of texts in a paragraph.

- Insert the **`setLayout (new FlowLayout ())`** statement in your constructor.
- Include the **`import`** statement for the **`java.awt`** package.

Note that there is no visible effect for the statements added in this step, unless there are objects or components added to your frame.

```
   public ASimpleGUI ()
   {
      super ("A Simple GUI Application");
      setDefaultCloseOperation (EXIT_ON_CLOSE);
```

```
        setLayout (new FlowLayout ());  // changes the layout manager

        setSize (400, 200);
        setVisible (true);
    }
```

Step 6. Add components to the Window. Let's start with adding a label. Labels are for displaying short texts, images, or both. This type of component does not react to any input events. To add a label, we use the data type called **`JLabel`**.

Create a method called **initScreen** where all drawing of components for the frame happens here. This is just one way to write your program in an organized manner.

```
    public void initScreen ()
    {
        JLabel lblTitle;

        // instantiates a JLabel object, with text Name:
        lblTitle = new JLabel ("Name:  ");
        add (lblTitle);  // puts the label into the frame
                         // The add method is a method of one of the
                         // Container class, where JComponent inherits
                         // from; JComponent is the superclass of JFrame;
                         // and JFrame is the superclass of ASimpleGUI
    }
```

Before closing your file, make sure to insert the statement that invokes **initScreen**. Put it inside the constructor, after the **setLayout** statement.

Step 7. Compile and run your program.

Step 8. For single line text input, you may use **`JTextField`**. Again, declare, instantiate and add the component inside the **`initScreen()`** method.

```
        JTextField tfName;
        tfName = new JTextField (); // this version of the constructor
                                    // creates a text field with not text
        add (tfName);
```

When running your program, if you do not see the component that you declared and instantiated on the window, double check if you have **add**ed the component into the frame.

This (Figure 2) is what you should see, when you run your program.

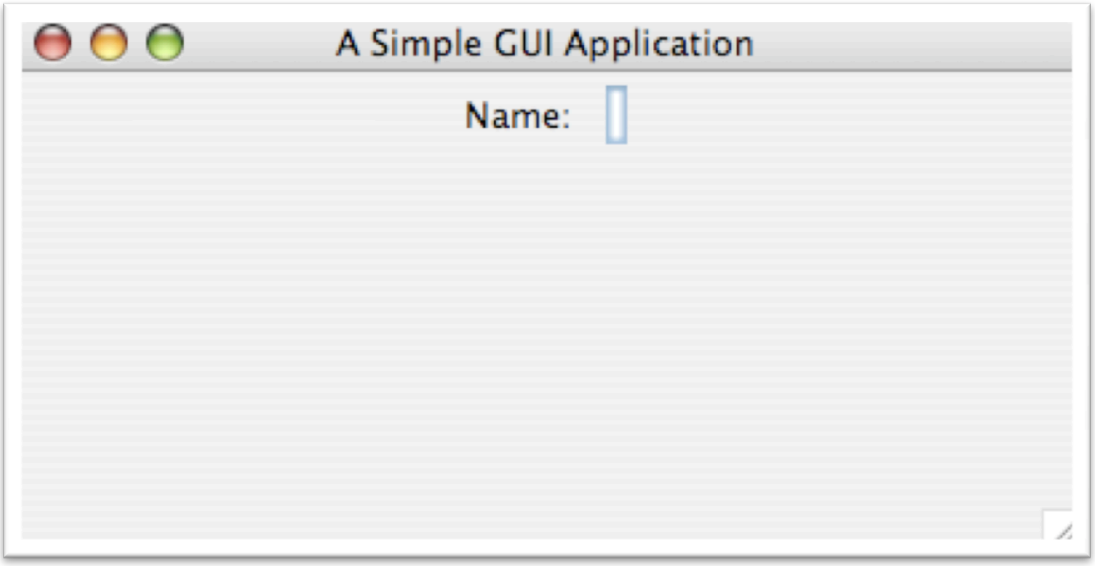

Figure 2. ASimpleGUI with label and text field added.

Step 9. To resize the text field, you may call the **`setSize()`** method. Indicate the width (in pixels) and height (in pixelds) of the text field. You may also use **`setColumns()`** to retain the *normal* height of the text field. In this case, indicate the number of columns.
Insert the statement **`tfName.setColumns (10);`** before the **`add(tfName);`** statement.

Step 10. Compile and run the program to see what happens.
Step 11. If you want to set the horizontal alignment of the text field, you may use the **`setHorizontalAlignment(<alignment_style>);`**. You may choose from the following alignment styles:

- **`JTextField.LEFT`**
- **`JTextField.CENTER`**
- **`JTextField.RIGHT`**
- **`JTextField.LEADING`**
- **`JTextField.TRAILING`**

Experiment on the different styles to see the effects.
Step 12. Compile and run your program.
Step 13. To add a button, use **`JButton`**. Similar to adding the previous components, declare, instantiate, and add the button.

```
JButton btnOk;
btnOk = new JButton ("Ok");  // creates a button with text Ok
add (btnOk);
```

Step 14. Compile and run your program. Note that nothing happens when the Ok button is pressed, since the action to be performed when the button is pressed is not yet programmed.
Step 15. To specify the effect of pressing a button, do the following.
a. Add the import statement for the **`java.awt.event`** package.

b. Include in your class definition, implements **`ActionListener`**. The **`ActionListener`** is responsible for receiving action events.
c. In the **`initScreen()`** method, insert the statement **`btnOk.addActionListener(this);`** after the button was created.
d. Implement the **`actionPerformed()`** method of the **`ActionListener`** interface inside your class.

```
public void actionPerformed (ActionEvent e)
 {
 }
```

e. Specify the action/s that will be executed when the button is pressed inside the **`actionPerformed()`** method. In our case, let us change the text of the button to Done.

```
public void actionPerformed (ActionEvent e)
{
   JButton  btn;

   if (e.getActionCommand ().equals ("Ok"))
   {  // when the command is Ok...

      // retrieve the source of this command
      btn = (JButton) e.getSource ();
      // change the text of the button to Done
      btn.setText ("Done");
   }
}
```

Note that if you press Ok, the button will now become Done. When you press Done, nothing will happen since no action(s) are specified when the Done command is issued.
f. Include the code that will change the text of the button back to Ok, when Done is pressed.

Resize the Window and take not of the effect of the positions of the components.

Usually, the components are directly added to the main screen. What is often done is that a panel is created and the components are added to the panel first. When all components are properly added already, the panel is added to the main screen.

Step 16. To add a panel, use **`JPanel`**. A panel is an invisible plane, where you can specify the layout manager and add components. Panels can be added to the frame or another panel.

Same as the previous steps in adding components, declare, instantiate and add the panel into your frame. Put these statements inside the **`initScreen()`** method.

```
// creates a panel and specifies the layout manager
JPanel panel = new JPanel (new BorderLayout ());
add (panel);
```

`BorderLayout` is a layout manager, which divides the panel into 5 parts, namely NORTH, SOUTH, WEST, CENTER and EAST.

Step 17. Create the following components: a label, three buttons with texts: Left, Right and Center. Put the declaration for the label outside the method but inside the class (i.e. before the constructor). Do not forget to add the action listener for each of the buttons.

Step 18. To add these components into the panel, the position where the component will be added must be specified since the layout manager being used is **`BorderLayout`**. For each area in the panel, you can only add one component.

```
// creates a panel and specifies the layout manager
JPanel panel = new JPanel (new BorderLayout ());
add (panel);

JButton btn;
btn = new JButton ("Left");
btn.addActionListener (this);
panel.add (btn, BorderLayout.WEST);  // fills the entire WEST area
                                     // with the button added

btn = new JButton ("Right");
btn.addActionListener (this);
panel.add (btn, BorderLayout.EAST);

btn = new JButton ("Center");
btn.addActionListener (this);
panel.add (btn, BorderLayout.CENTER);

lblDisplay = new JLabel ();
panel.add (lblDisplay, BorderLayout.SOUTH);
```

Step 19. Specify the actions to be executed when the buttons Left, Center and Right are clicked. When the Left button is clicked, the text Left will be displayed in the label added in the previous step with left horizontal alignment. When the Center button is clicked, the text Center will be pressed with center alignment. Finally, when the Right button is clicked, Right will be displayed with right alignment.

The following code should be added to your **`if`** statement inside the **`actionPerformed()`** method, for the actions to be taken when the Left button is pressed.

```
else if (e.getActionCommand ().equals ("Left"))
{
   // set the text of lblDisplay to Left
   lblDisplay.setText ("Left");

   // set the horizontal alignment to left
   lblDisplay.setHorizontalAlignment (JLabel.LEFT);
}
```

Do the same for the other two buttons, Center and Right.

Step 20. Compile, Run and observe what happens when the buttons are pressed.

Resize the Window and take not of the effect of the positions of the components.

Modify your code, such that the first three components added to the frame are first placed in a panel, and the panel is the component added to the frame. In other words,

- At the start of the **`initScreen()`** method, create a panel;
- Add the label Name, the text field, and the Ok button to the panel; and
- Add the panel to the frame.

Compile and execute your program.

Resize the Window and take not of the effect of the positions of the components.

Errors and Suggested Solution

ERROR MESSAGE	INTERPRETATION & SUGGESTED SOLUTION

`ASimpleGUI.java:2: cannot find symbol`
`symbol: class JFrame`
`public class ASimpleGUI extends JFrame`
`^`
	Interpretation:
	The literal **`JFrame`** is not recognized by the compiler.
	Suggested Solution:
	The **`JFrame`** class falls under the **`javax.swing`** package. Be sure that the **`import`** statement for the said package is included in your code.

`'{' expected`
`public class ASimpleGUI extend JFrame`
`^`
	Interpretation:
	The error says that an open brace **`{`** is missing at required at the specified point.
	Suggested Solution:
	In this case, the compiler was confused since the keyword **`extends`** is misspelled.

`')' expected`
`super (A Simple GUI Application);  // call to JFrame's constructor`
`^`
	Interpretation:
	The compiler is expecting a close parenthesis at the point specified.
	Suggested Solution:
	The class **`JFrame`** supplied many constructors (see **`JFrame`** API, Constructor Summary). In this case, the compiler thought that you were placing a variable as a parameter that is why it expected a close parenthesis at the point specified.
	However, our intention was to put a **`String`** for the title, as the parameter. Enclose the **`String`** parameter in double quotes.

ERROR MESSAGE	INTERPRETATION & SUGGESTED SOLUTION

`cannot find symbol`
`symbol  : class FlowLayout`
`location: class ASimpleGUI`
`setLayout (new FlowLayout ());  // changes the layout manager`
`^`
	Interpretation:
	The literal **`FlowLayout`** is not recognized by the Java compiler.
	Suggested Solution:
	`FlowLayout` belongs to the **`java.awt`** package. Make sure that the said package is imported.

`variable lblTitle might not have been initialized`
`add (lblTitle);  // puts the label into the frame`
`^`
	Interpretation:
	The variable **`lblTitle`** has no initial value at the time that it is used, as indicated.
	Suggested Solution:
	`JLabel` is a reference data type. Before accessing a reference data type, be sure to instantiate (or create) it first.
	To instantiate, the syntax is:
	`<variable> = new <constructor_call>;`
	Refer to the API Documentation for different constructors that can be called for the said reference variable.

ERROR MESSAGE	INTERPRETATION & SUGGESTED SOLUTION
cannot find symbol symbol : constructor JLabel(char) location: class javax.swing.JLabel lblTitle = new JLabel ('a'); ^ 1 error	**Interpretation:** The constructor call for **`JLabel`** as indicated here is not recognized by the compiler. **Suggested Solution:** Refer to the Constuctor Summary Section of **`JLabel`**'s API Documentation. Make sure that the constructor call you used matches one of the listed constructors, i.e. the data types of the parameters passed are compatible with one of the listed constructor's parameters.
cannot find symbol symbol: class ActionListener public class ASimpleGUI extends JFrame implements ActionListener ^	**Interpretation:** The symbol **`ActionListener`** is not recognized by the compiler. **Suggested Solution:** Make sure you have included the import statement for the Java package that contains the **`ActionListener`**. In this case, the **`java.awt.event`** package.

ERROR MESSAGE	INTERPRETATION & SUGGESTED SOLUTION

`ASimpleGUI is not abstract and does not override abstract method actionPerformed(java.awt.event.ActionEvent) in java.awt.event.ActionListener`
`public class ASimpleGUI extends JFrame implements ActionListener`
`       ^`
	Interpretation:
	The **`ActionListener`** is an interface[1]. The methods of the said interface must be implemented in your class.
	Suggested Solution:
	Check the methods of the interface(s) you implemented (refer to the API Documentation). Make sure you have implemented all the methods for each of the interface(s) your class implements.

`cannot find symbol`
`symbol  : variable getActionCommand`
`location: class java.awt.event.ActionEvent`
`      else if (e.getActionCommand .equals ("Left"))`
`                ^`
	Interpretation:
	For the Java compiler, the **`getActionCommand`** is an unknown symbol.
	Suggested Solution:
	`getActionCommand` is a no parameter method, a pair of parenthesis must be found after the method name, i.e. **`getActionCommand ()`**

[1] An interface is a group of related methods with empty bodies (as in no implementation specified).

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class ASimpleGUI extends JFrame implements ActionListener
{
   JLabel lblDisplay;

   // the no-parameter constructor of ASimpleGUI
   public ASimpleGUI ()
   {
      super ("A Simple GUI Application");  // call to JFrame's constructor
                                           // param = title of your window
      // terminates the program when window is closed
      // EXIT_ON_CLOSE is a constant defined inside JFrame. Since
      // ASimpleGUI is an extension of JFrame, the constant can be
      // accessed directly.
      setDefaultCloseOperation (EXIT_ON_CLOSE);
      setLayout (new FlowLayout ());   // changes the layout manager

      initScreen ();

      setSize (400, 200);  // sets the dimension of the window
      setVisible (true);   // must always be the last statement
                           // this makes the window visible.
   }

   /*  responsible for drawing/putting components into the frame */
   public void initScreen ()
   {
      JLabel lblTitle;

      // instantiates a JLabel object, with text Name:
      lblTitle = new JLabel ("Name:  ");
      add (lblTitle);  // puts the label into the frame
                       // The add method is a method of one of the
                       // Container class, where JComponent inherits
                       // from; JComponent is the superclass of JFrame;

      JTextField tfName;
      tfName = new JTextField (); // this version of the constructor
```

```
                                    // creates a text field with not text

      tfName.setColumns (10); // sets the number of columns for tfName
      add (tfName);

      JButton btnOk;
      btnOk = new JButton ("Ok");  // creates a button with text Ok
      btnOk.addActionListener (this);  // specifies action listener for this
                                       // button
      add (btnOk);

      // creates a panel and specifies the layout manager
      JPanel panel = new JPanel (new BorderLayout ());
      add (panel);

      JButton btn;
      btn = new JButton ("Left");
      btn.addActionListener (this);
      panel.add (btn, BorderLayout.WEST);  // fills the entire WEST area with
                                           // the button added

      btn = new JButton ("Center");
      btn.addActionListener (this);
      panel.add (btn, BorderLayout.CENTER);

      btn = new JButton ("Right");
      btn.addActionListener (this);
      panel.add (btn, BorderLayout.EAST);

      lblDisplay = new JLabel ();
      panel.add (lblDisplay, BorderLayout.SOUTH);
   }

   public void actionPerformed (ActionEvent e)
   {
      JButton btn;

      if (e.getActionCommand ().equals ("Ok"))
      {  // when the command is Ok...
         // retrieve the source of this command
         btn = (JButton) e.getSource ();
         // change the text of the button to Done
         btn.setText ("Done");
```

```
      }

      else if (e.getActionCommand ().equals ("Done"))
      {
         btn = (JButton) e.getSource ();
         btn.setText ("Ok");
      }
      else if (e.getActionCommand ().equals ("Left"))
      {
         // set the text of lblDisplay to Left
         lblDisplay.setText ("Left");
         // set the horizontal alignment to left
         lblDisplay.setHorizontalAlignment (JLabel.LEFT);
      }
      else if (e.getActionCommand ().equals ("Center"))
      {
         lblDisplay.setText ("Center");
         lblDisplay.setHorizontalAlignment (JLabel.CENTER);
      }
      else if (e.getActionCommand ().equals ("Right"))
      {
         lblDisplay.setText ("Right");
         lblDisplay.setHorizontalAlignment (JLabel.RIGHT);
      }
   }

   public static void main (String[] args)
   {  ASimpleGUI app = new ASimpleGUI ();  // starts the java application
   }
}
```

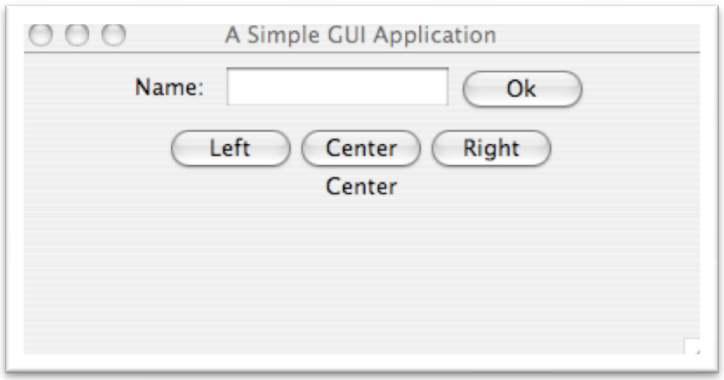

Figure 3. Screen shot of the ASimpleGUI Application

Lab Activity

Lab Exercise 9 – Number Keypad Application

Name: _____ Date: _____

Section: _____

Description of the Activity

Write a Java application that draws a number keypad on screen. When the user clicks on any of the numeric buttons, the corresponding number pressed will be appended to the display. For instance, when the sequence of buttons pressed is 7, 4, 6, . (dot), and 8, the display should show **`746.8`**.

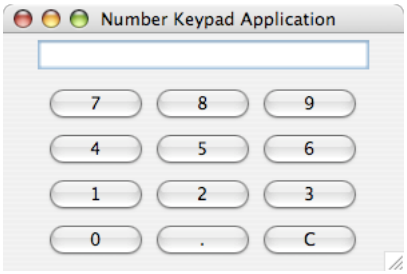

Figure 4. Sample Screen Shot of the Number Keypad Application

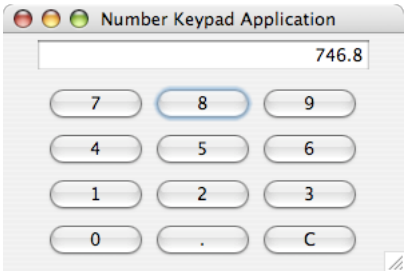

Figure 5. Sample Screen Shot of Number Keypad Application after a series of button clicks

The display area is a text field with number of columns set to 20. There are 12 buttons on screen, one for each digit, one for the dot (.), one for C. C stands for clear. When the user presses this button, anything on the display will be cleared or erased.