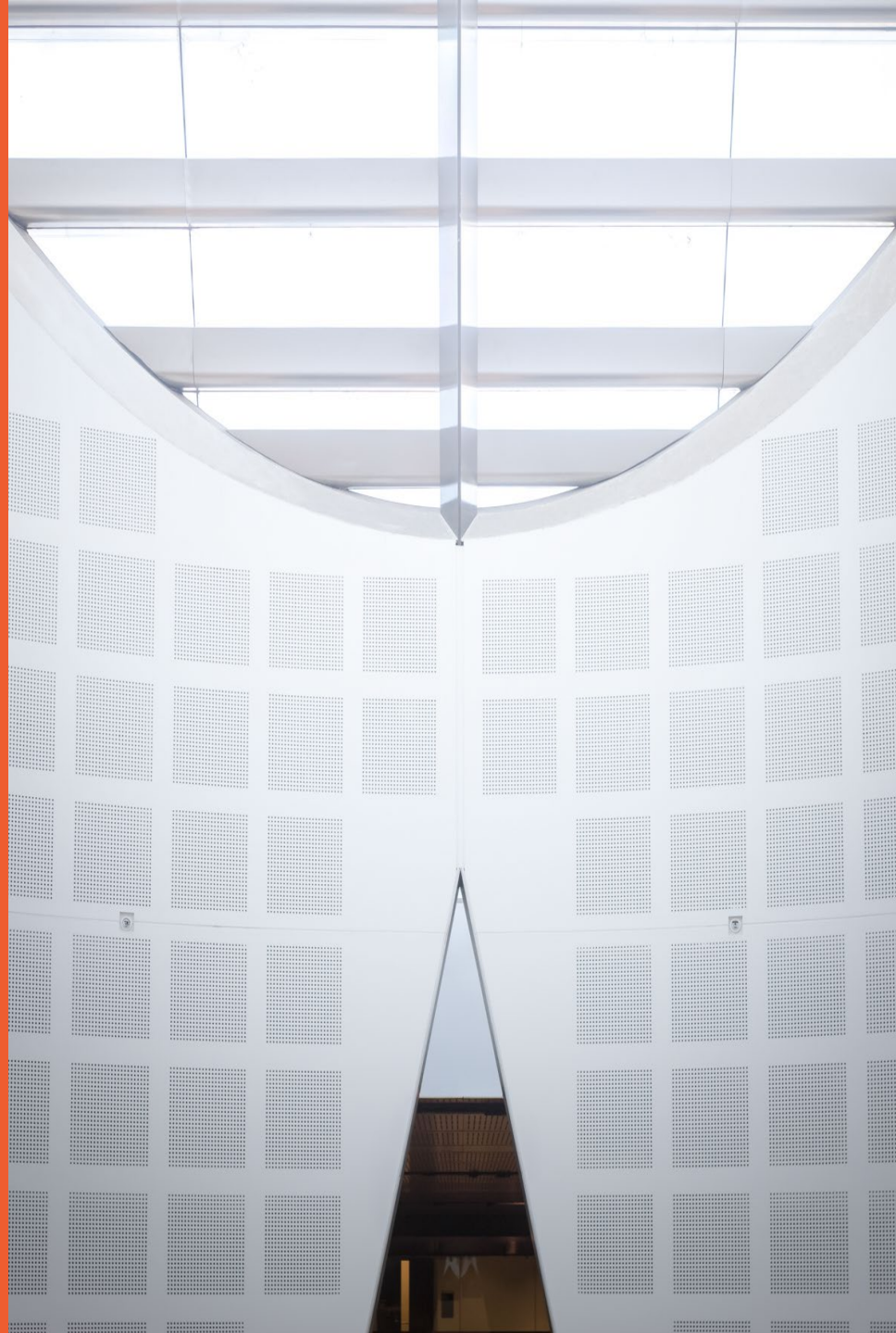
COMP9103: Software Development in Java

W11: Graphical User Interfaces

Presented by
Dr Ali Anaissi
School of Computer Science

THE UNIVERSITY OF SYDNEY



Introduction

- GUI – Graphical User Interface
  - Various visual components, e.g. text fields, buttons, drop-down list boxes
  - Order of input is more flexible – able to switch between items
  - Text can be displayed using various fonts
  - Images and colour can be used

GUIs in Java

- Earliest version was called the "Abstract Windowing Toolkit" (AWT)
  - Classes/Interfaces are in the `java.awt` package, and its subpackages
  - The idea was that the operating system's normal look and feel would be how things appeared
- Current GUIs use the "Swing Toolkit"
  - Built on top of the AWT framework, the additional classes and interfaces are found in the `javax.swing` package
  - A consistent appearance regardless of operating system

Elements of GUI-based programs

- Components
- Containers
- Layout Manager
- Events and Event Handlers

Crating GUI applications

- We need to know
  - How to create components such as buttons, labels, checkboxes, radio buttons, etc.
  - How to assemble them inside a frame according to a preferred arrangement or order of the components.
  - How to perform actions on the components such as
    - *click a button*
    - *select an item from list*
    - *select an item from combobox*
    - *select an item from radio button*

JFrame

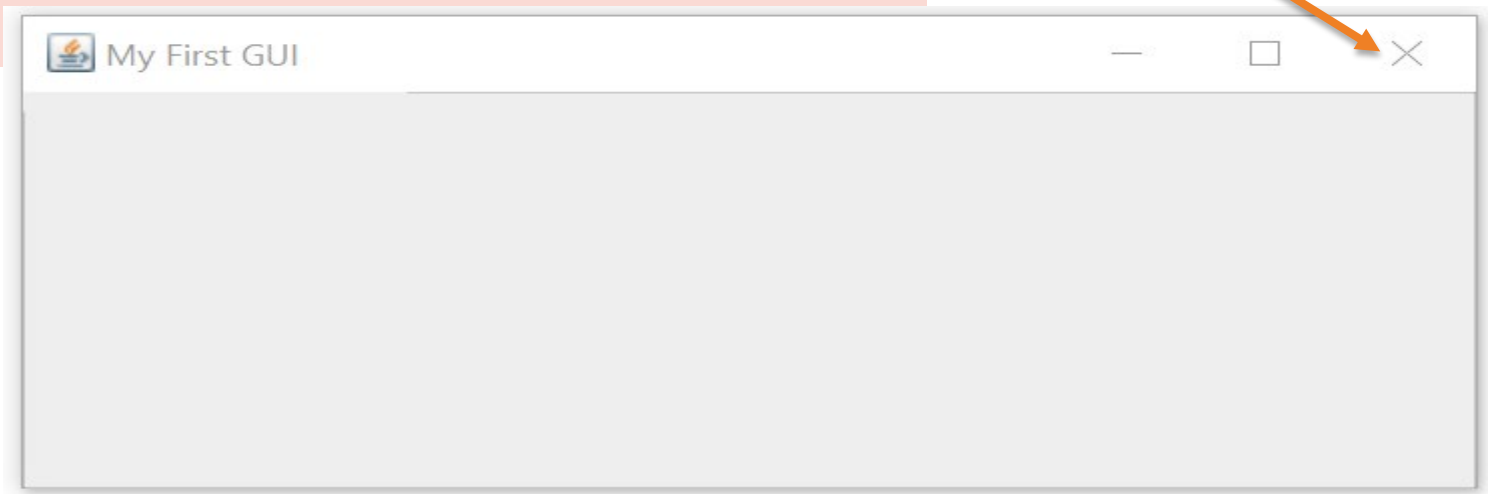
- Standalone windows created using JFrame
- JFrame is a class in `javax.swing` package
  - It contains data (which we can't access)
  - It provides methods (which we *can* use)
- We instantiate in the same way as we instantiate other classes
  - Using the new operator

```
JFrame jf = new JFrame("My first GUI");
```

Example

```
public class GUI {
    private JFrame jf;
    public GUI() {
        jf = new JFrame("My First GUI");
        jf.setSize(500, 300);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);

    }
}
```

```
public class Main {

 public static void main(String[] args)
    {
        new GUI();
    }
}
```



JTextField

- Provides a box for the user to enter text

- Constructor requires to be told how many characters wide the field should be, e.g.

```
JTextField numT = new JTextField(15);
```

- Find out what the user has typed:

```
String num1 = numT.getText();
```

JLabel

- Used purely for “documenting” your window for the user
- Usually (not always) just contain a piece of text which describes another component.

```
JLabel numL = new JLabel("First Number:" );
```

JTextArea

- A multi-line text input component
- Constructor needs to be told how many rows and columns it should show on the screen at once:

```
JTextArea results = new JTextArea(4,30);
```

- It has many methods that are common to JTextField
  - Both are subclasses of JTextComponent

Buttons

- GUI buttons fall into various categories:
  - push button – a generic button that initiates some action
  - check box – a button that can be toggled on or off
  - radio buttons – a set of buttons that provide a set of mutually exclusive options
- Radio buttons must work as a group; only one can be toggled on at a time
- Radio buttons are grouped using the ButtonGroup class

JButton

- Buttons are created in swing using JButton class.
- Similar to JLabel you should instantiate it with a text so that you can push the button.

```
JButton exit =  new JButton ("Exit");
```

JCheckBox

- You can create a checkbox using JCheckBox class that is nothing but an item which can be selected or deselected.
- A checkbox generates one ItemEvent and one ActionEvent when it is clicked (selected or deselected).
- It is generally enough you handle any one of these two events.
- You can check whether a checkbox is selected or not using isSelected() method.
- The getText() returns the label of a checkbox.

JRadioButton

- Radio buttons are created using JRadioButton class.
- Radiobuttons allow us to select any one of the buttons from a group of buttons.
  - To have a group behavior, all radio buttons are added to ButtonGroup object.
- Radio buttons fire ItemEvent and ActionEvent when it gets clicked.
  - Generate ItemEvents
    - An ItemListener can respond
  - Generate ActionEvents
    - An ActionListener can respond
- You can check whether a radio button is selected or not using isSelected() method.
- The getText() returns the label of a radio button .

Radio Buttons

- Declaration and initialization:

```
JRadioButton small =  new JRadioButton ("small", false);
JRadioButton large =  new JRadioButton ("large");
```

- But one needs to put them in a group so they become mutually exclusive

```
ButtonGroup sizeGroup = new ButtonGroup();
sizeGroup.add(small);
sizeGroup.add(large);
```

Layout Managers

- The swing components are added to a frame according to the specified layout.

- The Layout Managers
  - Provid a way to arrange GUI components in a container
  - Provide basic layout capabilities
  - Implement the interface LayoutManager

Layout Managers

- BorderLayout: Default for Jframe.
  - Use when you add components to a maximum of five sections arranged in north, south, east, west, and center positions
- FlowLayout: Default for Jpanel.
  - Use when you need to add components from left to right, Flowlayout automatically moves to the next row when needed, and each component takes its preferred size
- GridLayout: Use when you need to add components into a grid of rows and columns; each component is the same size

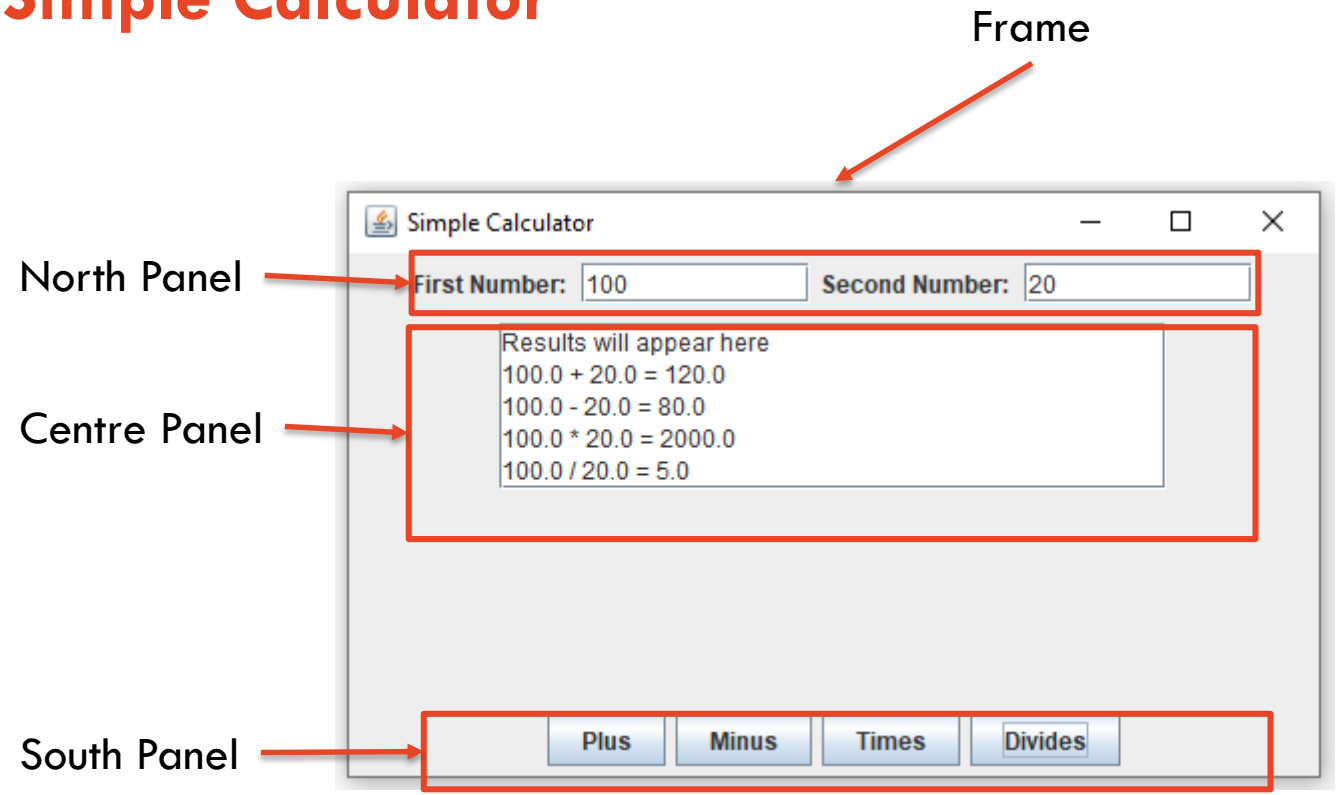
JPanel

- JPanel is a nice container that can group components together

- It is used when you want to group several components into a single frame using different layout managers.
- The JPanel is the extensively used container in swing because it behaves like a frame and you can do all things whatever you do with frames.

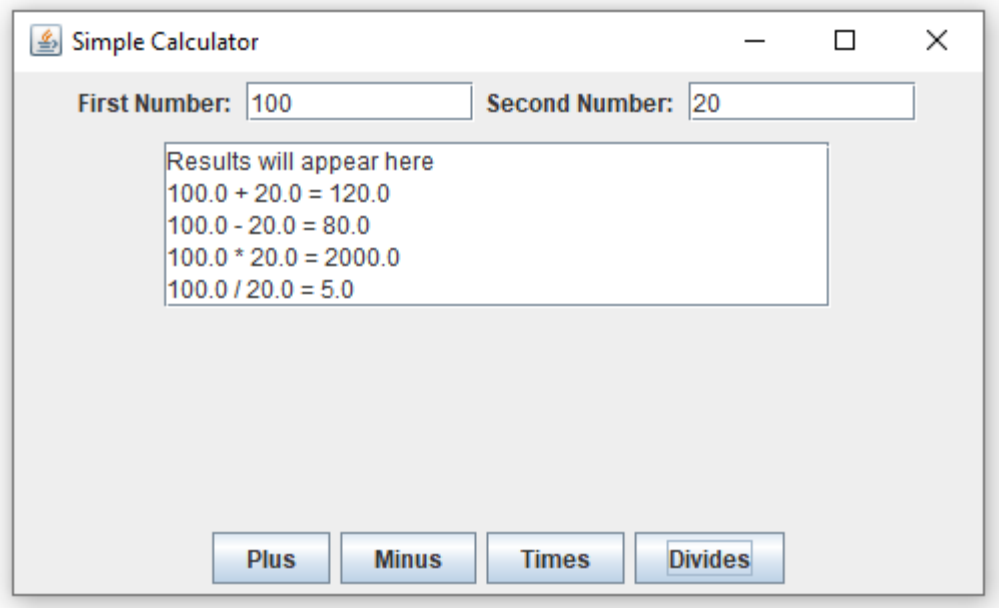
```
JPanel northPanel = new JPanel();
```

Simple Calculator



Example

```
public class GUI {
    private JButton add, min, mul, div;
    private JLabel numL1, numL2;
    private JTextField numT1, numT2;
    private JTextArea outputTextArea;
    private JPanel northPanel, southPanel, centerPanel;
    private JFrame jf;
}
```

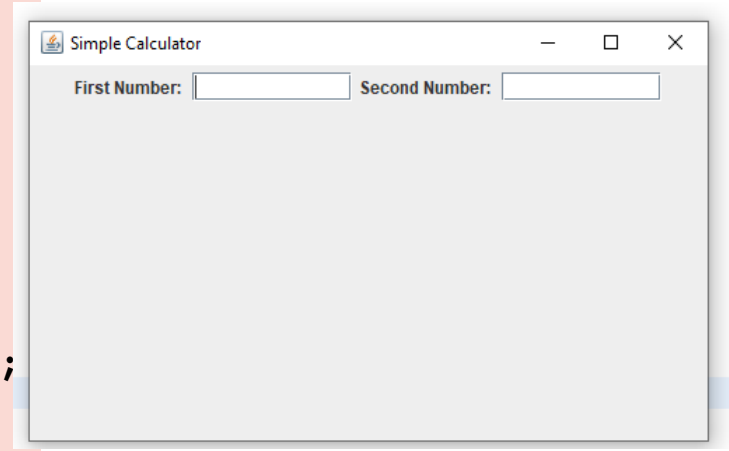


Example

```
public class GUI {
    ....
    public GUI() {
        jf = new JFrame("Simple Calculator");
        northPanel = new JPanel();
        numL1 = new JLabel("First Number: ");
        numL2 = new JLabel("Second Number: ");
        numT1 = new JTextField(10);
        numT2 = new JTextField(10);
        northPanel.add(numL1);
        northPanel.add(numT1);
        northPanel.add(numL2);
        northPanel.add(numT2);
        jf.add(northPanel, BorderLayout.NORTH);
        jf.setVisible(true);
        jf.setSize(500, 300);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

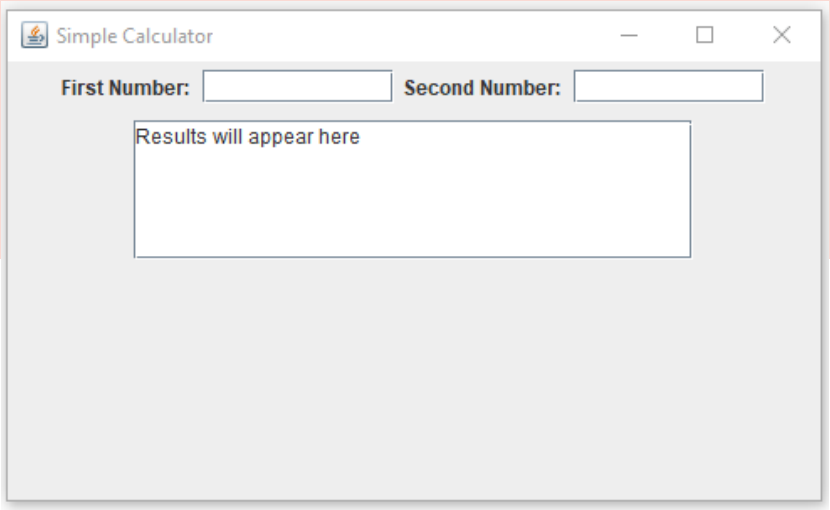
Default layout of JPanel is FlowLayout

```
public class Main {

 public static void main(String[] args)
    {
        new GUI();
     }
}
```



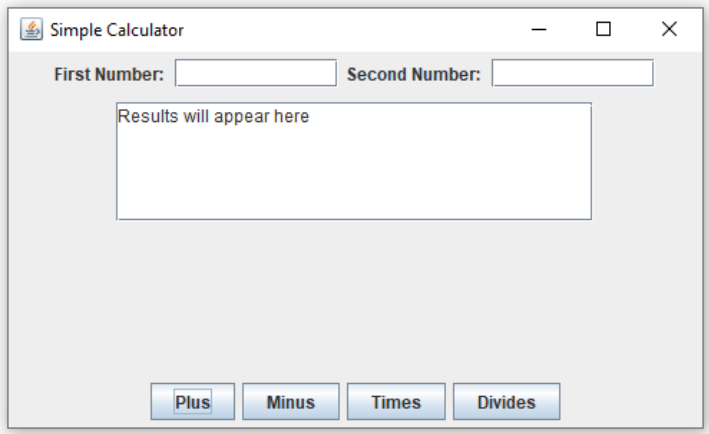
Example

```
public class GUI {
    ....
    public GUI() {
        ....
        centerPanel = new JPanel();
        outputTextArea =new JTextArea("Results will appear here");
        outputTextArea.setColumns(30);
        outputTextArea.setRows(5);
        JScrollPane scrollPane = new JScrollPane(outputTextArea);
        centerPanel.add(scrollPane);
        jf.add(centerPanel, BorderLayout.CENTER);
         ....

        }
}
```



Example

```
public class GUI {
    ....
    public GUI() {
        ....
        southPanel = new JPanel();
        plus = new JButton("Plus");
        minus = new JButton("Minus");
        times = new JButton("Times");
        divides = new JButton("Divides");
        southPanel.add(plus);
        southPanel.add(minus);
        southPanel.add(times);
        southPanel.add(divides);
        jf.add(southPanel, BorderLayout.SOUTH);
        ....
    }
}
```



Event-Driven Programming

- When something happens, an 'Event' is generated, announced, and possibly some part of the code will react to its occurrence.

- Event-driven programming:
  - Programming to react to events

- GUI programs are event-driven programs:
  - We need to provide logic to **react** to user interaction with the components

Basics of event Handling

- All interfaces, classes and methods that are required for handling events are defined in awt.event package.

- Each component in swing will report (or fire) all events that may happen to it.

- If you are interested in a particular event, then you will register a listener for this event that will perform the necessary operations through its implementation.

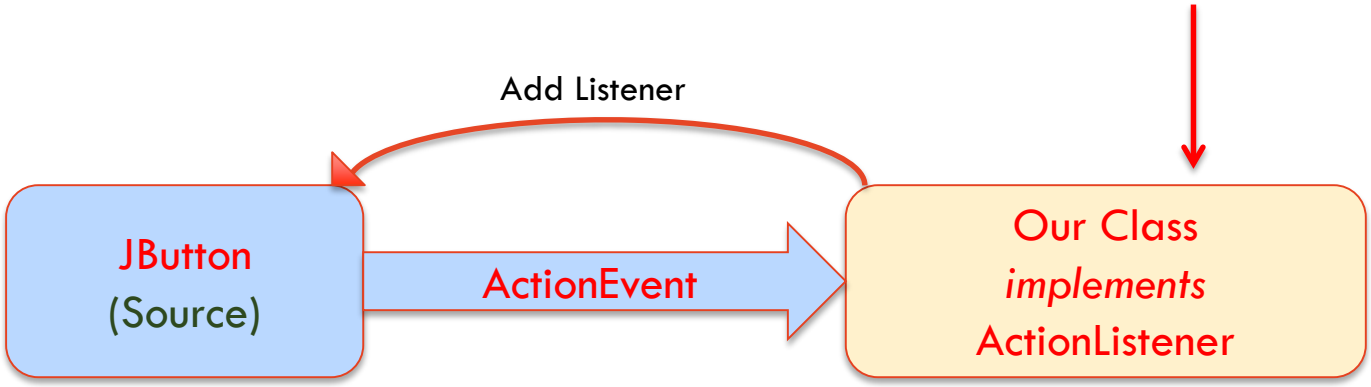
- Therefore event handling has two parts:
  - register listeners for all interested events of a component
  - provide implementations for the listeners that will do the task for you.

ActionEvent / ActionListener

- For instance in the previous example shown in, we have created a button.
- We are interested in an action or event namely button press.
- The button reports to us using an event ActionEvent. So we need to register a listener to this button using addActionListener() method.
- This method will take an argument that is an object that implements the ActionListener interface.
  - The ActionListener interface has a method actionPerformed() that takes ActionEvent as an argument.
  - Therefore the operation you want to do while the button is pressed should be nested inside this actionPerformed() method. That is, actionPerformed() method will be called when the button is pressed.

Event Handling

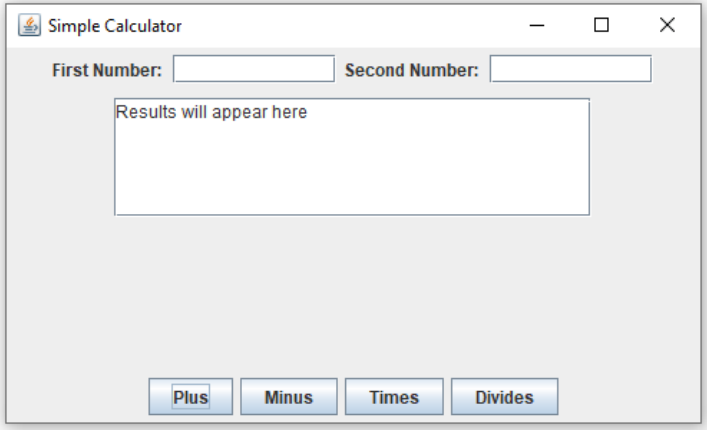
- There are three participants in Java AWT (and swing) event handling:
  - An **Event** object (e.g. an ActionEvent, WindowEvent)
  - The event's source (e.g. a JButton object)
  - The event's listener (receiver of the event, we write this)



Example

```
public class GUI  implements ActionListener {
        ……
    public GUI() {
        ……
        JPanel southPanel = new JPanel();
        plus = new JButton("Plus");
        minus = new JButton("Minus");
        times = new JButton("Times");
        divides = new JButton("Divides");
        plus.addActionListener(this);
        minus.addActionListener(this);
        times.addActionListener(this);
        divides.addActionListener(this);
        ……
    }
    @Override
      public void actionPerformed(ActionEvent ae) {

      }
}
```

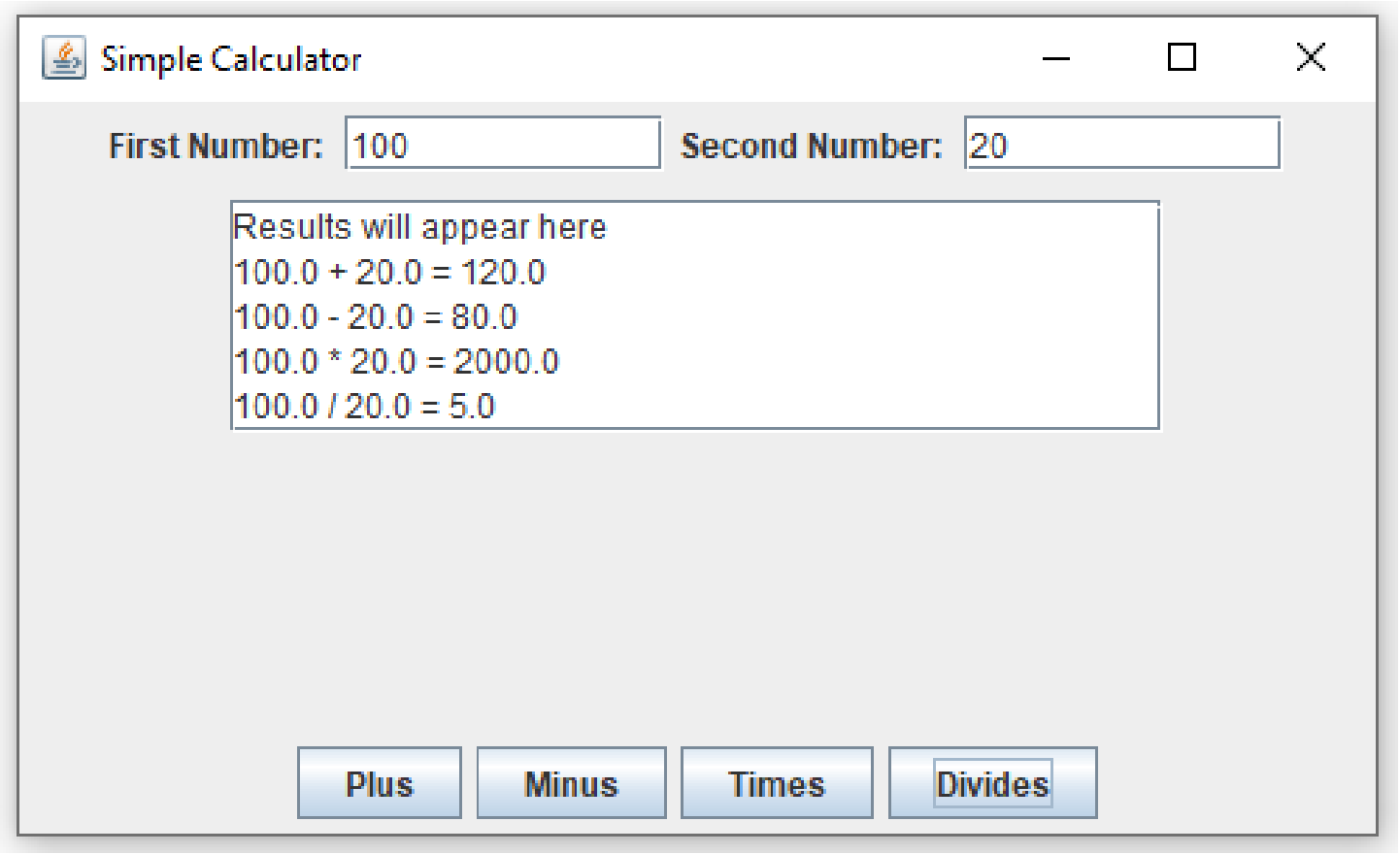
```
public class Main {

 public static void main(String[] args)
    {
          new GUI();
     }
}
```



Example

```
public class GUI  implements ActionListener {
        ……
     @Override
       public void actionPerformed(ActionEvent ae) {
           double num1, num2;
         try {
               num1 = Double.parseDouble(numT1.getText());
               num2 = Double.parseDouble(numT2.getText());
               if(ae.getSource() == plus)
                       outputTextArea.append("\n"+ num1+ " + "+ num2 +" = "+String.valueOf(num1+num2));
               else if(ae.getSource() == minus)
                       outputTextArea.append("\n"+num1+ " - "+ num2 +" = "+String.valueOf(num1 - num2));
               else if(ae.getSource() == times)
                       outputTextArea.append("\n"+num1+ " * "+ num2 +" = "+String.valueOf(num1 * num2));
               else if(ae.getSource() == divides)
                       outputTextArea.append("\n"+num1+ " / "+ num2 +" = "+String.valueOf(num1 / num2));
           }
         catch(Exception e) {
                JOptionPane.showMessageDialog(jf, "Invalid values");
           }
       }
}
```

Example


Simple Calculator
First Number:
100
Second Number:
20
Results will appear here
100.0 + 20.0 = 120.0
100.0 - 20.0 = 80.0
100.0 * 20.0 = 2000.0
100.0 / 20.0 = 5.0
Plus
Minus
Times
Divides

JComboBox

- JComboBox allows us to select any one of the set of items.
- It is also called dropdown list as the full list will be displayed when the down arrow button is clicked.
- The JComboBox will fire ItemEvent and ActionEvent when an item is selected.
- The getSelectedItem() returns a currently selected item as String while addItem() adds an Object to combo box.
- You can populate combo box using array of objects or a Vector directly.

```
String[] Mstatus = {"Single", "Married", "Engaged"};
JComboBox cb = new JComboBox(Mstatus);
```

JList

- JList and is another interesting JComponent that is very similar to JCombo-Box.
- From JList you can select one item, or multiple items.
- A list can be initialized using an array or vector of objects and displayed in one or more columns.
- The items from list can be selected in three ways:
  - single click
  - multiple click (by holding Ctrl-key and clicking)
  - range click (by holding Shift- key and clicking)

```
String[] degree = {“Bachelors”, “Masters”, “Doctoral”};
JList lst = new JList(degree);
```

JList

- The JList fires ListSelectionEvent when a single item or multiple items are selected.
- In order to process list clicks, ListSelectionListener interface should be implemented by overridding valueChanged() method with the handling behavior you want to include for the selection.
- The getSelectedValuesList() returns a List containing all selected items from JList.
- The getValueIsAdjusting() returns true if the user is still manipulating the selection.
  - Otherwise, getSelectedValuesList() returns a util.List instance

Vector class

- Like an ArrayList – stores a collection of objects
- Declaration:

```
Vector< Element-Type >  variableName;
```

- Instantiation:

```
variableName = new Vector< Element-Type > ();
```

- Methods:
  - add(*element*)
  - remove(*element*)
  - remove(*index*)

Menus in Swing GUIs

Made up of three different objects:

- JMenuItem
  - Controls a menu item in a menu
  - Needs an associated ActionListener
- JMenu
  - Controls an <u>individual</u> menu, consists of JMenuItems
  - Order of JMenuItem objects determines visual order
- JMenuBar
  - Controls a <u>set of menus</u> at the top of the window

Menus in Swing GUIs

```
MenuBar mb = new JMenuBar();
JMenu mFile = new JMenu("File");
JMenu mEdit = new JMenu("Edit");
JMenu mTools = new JMenu("Tools");
JMenu mHelp = new JMenu("Help");

JMenuItem miNew = new JMenuItem("New");
JMenuItem miOpen = new JMenuItem("Open");
JMenuItem miSaveas = new JMenuItem("SaveAs");
JMenuItem miExit = new JMenuItem("Exit");
```

```
// add menu items to menu
  mFile.add(miNew);
  mFile.add(miOpen);
  mFile.add(miSaveas);
  mFile.add(miExit);

  //add menu to menubar
  mb.add(mFile);
  mb.add(mEdit);
  mb.add(mTools);
  mb.add(mHelp);

  //add menubar to frame
  myFrame.setJMenuBar(mb);
```

Questions?