Outline

- MVC
- JAVA GUI Classes
  - Jframes
  - Components
  - Containers/Layout Managers
- Interacting with GUI
  - ActionListener
  - actionPerformed
- Layouts
- Rational Number GUI

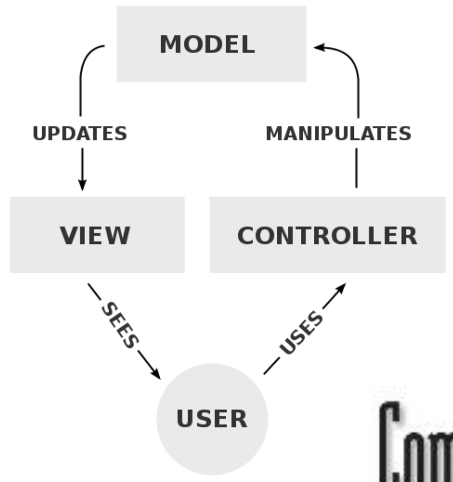
Console Applications

- All the user interaction we have done in this course has been handled through the console
- All user inputs must be entered by the keyboard and all outputs are made up of text in this one little window
- It is much easier for a user to interact with a program if it is visual instead of just text based
- In Java, we can build Graphical User Interfaces and run our program through them instead of always using the console

Model, View, Controller

Common design pattern for developing programs

- Model: Data & Methods used to work on the data
- View: GUI, presentation, and logic to process the GUI
- Controller: Coordinates Interactions between the View and the Model



View

- Previously, our view to the programs has been via the console window.

- However, more common application paradigm is to use a GUI to interact with your application

GUIs: Graphical User Interfaces

- Option Pane
- Jframes
- Containers
- Components
- Interacting with GUI
  - ActionListener
  - actionPerformed
- Layouts

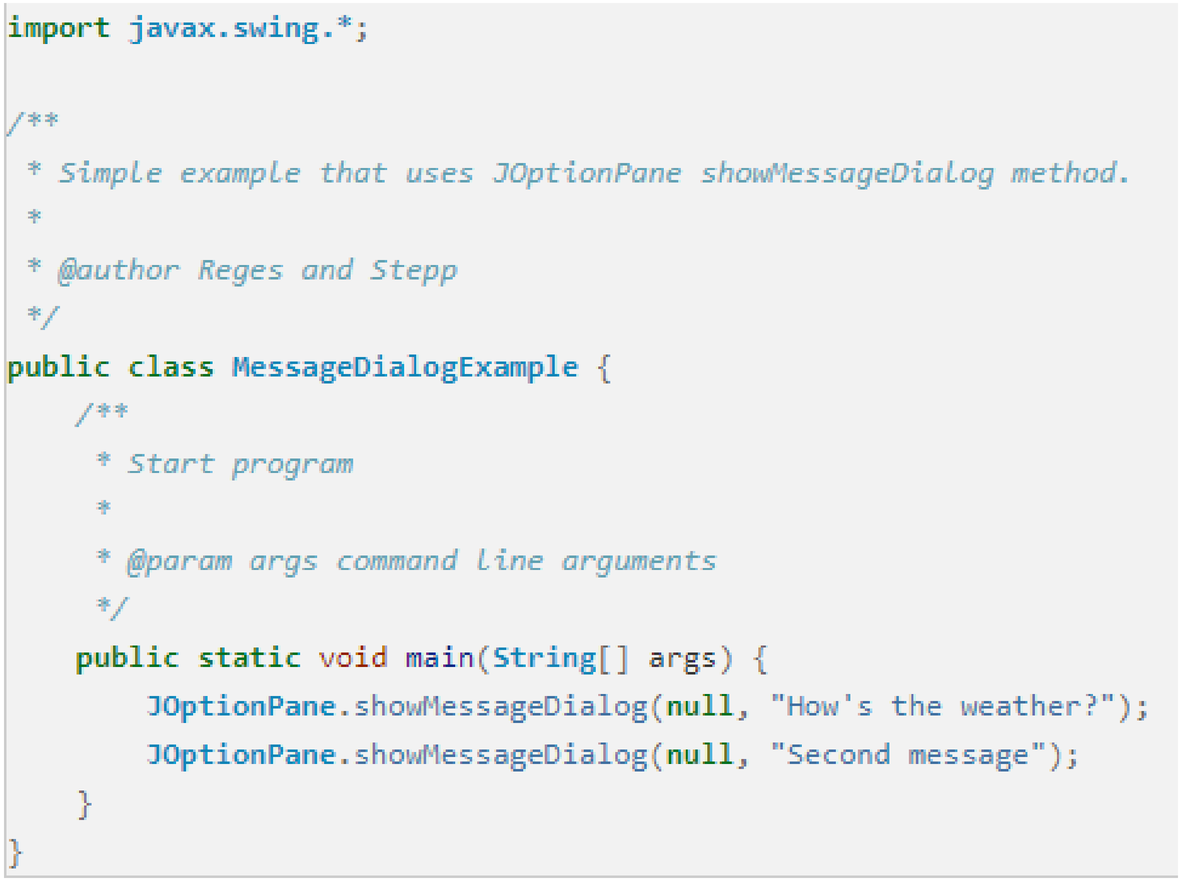
Option Pane

- The first object we will examine with GUIs is the option pane.
- Reges and Stepp[1] describe the option pane: “The simplest way to create a graphical window in Java is to have an option pane pop up. An option pane is a simple message box that appears on the screen and presents a message or a request for input to the user.”
- The option pane that we will be using is JOptionPane, which is part of the javax.swing package.
  - In order to use JOptionPane, we will have to import the javax.swing package.
- We will discuss three types of JOptionPane:
  - showMessageDialog(parent, message)
  - showConfirmDialog(parent, message)
  - showInputDialog(parent, message)

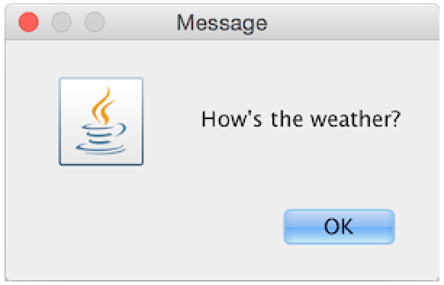
1 Reges, S., & Stepp, M. (2016). Building Java Programs: A Back to Basics Approach (4th ed.). Addison-Wesley.

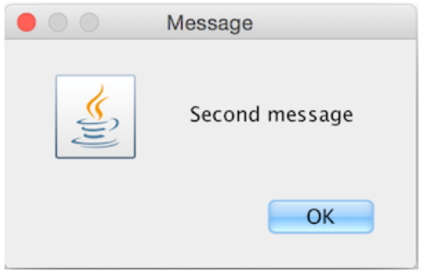
JOptionPane: showMessageDialog

- If we want to simply display a message with OK button, we can use the showMessageDialog method
- `showMessageDialog` is analogous to `System.out.println` to display a message



```
import javax.swing.*;

/**
 * Simple example that uses JOptionPane showMessageDialog method.
 *
 * @author Reges and Stepp
 */
public class MessageDialogExample {
    /**
     * Start program
     *
     * @param args command line arguments
     */
    public static void main(String[] args) {
        JOptionPane.showMessageDialog(null, "How's the weather?");
        JOptionPane.showMessageDialog(null, "Second message");
    }
}
```

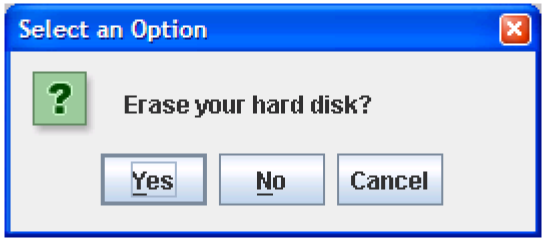




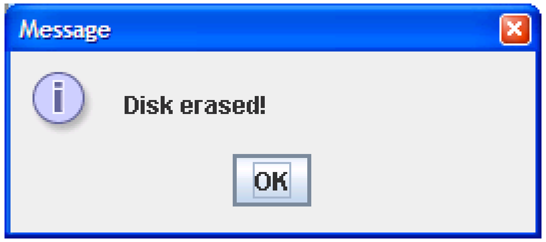
JOptionPane: showConfirmDialog

- If we want to simply display a message and a list of choices (Yes, No, or Cancel), we can use the showConfirmDialog method. This method returns the user's choice and an integer with one of the following values:
  - JOptionPane.YES_OPTION
  - JOptionPane.NO_OPTION
  - JOptionPane.CANCEL_OPTION
- `showConfirmDialog` is analogous to `System.out.println` analogous to a `System.out.print` that prints a question, then reads an input value from the user (one of provided choices)

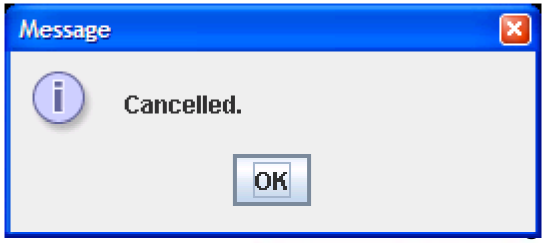
```
import javax.swing.*;

public class ConfirmDialogExample {
    public static void main(String[] args) {
        int choice = JOptionPane.showConfirmDialog(null, "Erase your hard
   disk?");
        if (choice == JOptionPane.YES_OPTION) {
            JOptionPane.showMessageDialog(null, "Disk erased!");
        } else {
            JOptionPane.showMessageDialog(null, "Cancelled.");
        }
    }
}
```


Select an Option

Erase your hard disk?


Message

Disk erased!

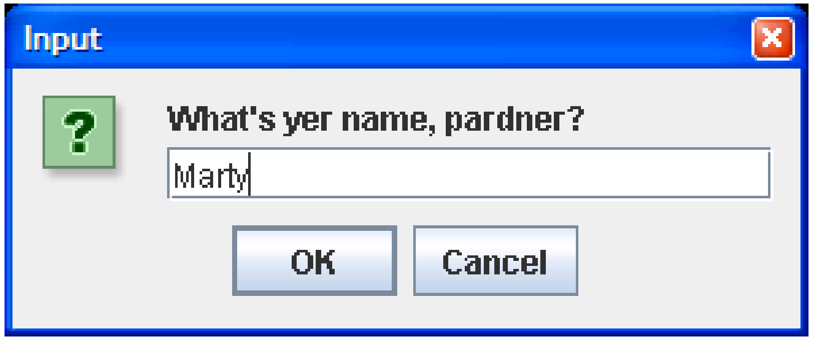

Message

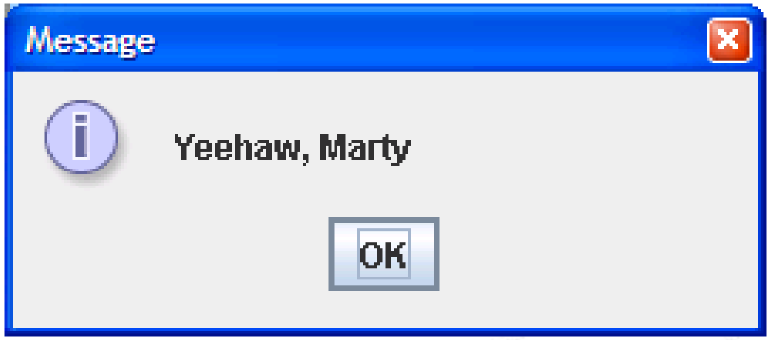
Cancelled.

JOptionPane: showInputDialog

- If we want to simply display a message and text field for input, we can use the showInputDialog method. The method returns the user's value entered as a String.
- `showInputDialog` analogous to a `System.out.print` that prints a question, then reads an input value from the user (can be any value)

```
import javax.swing.*;

public class InputDialogExample {
    public static void main(String[] args) {
        String name = JOptionPane.showInputDialog(null,
                                 "What's yer name, pardner?");
        JOptionPane.showMessageDialog(null, "Yeehaw, " + name);
    }
}
```

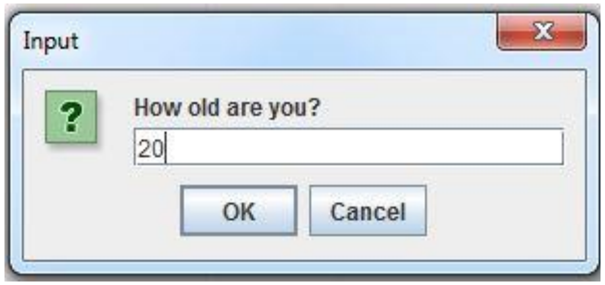




showInputDialog: input other than String

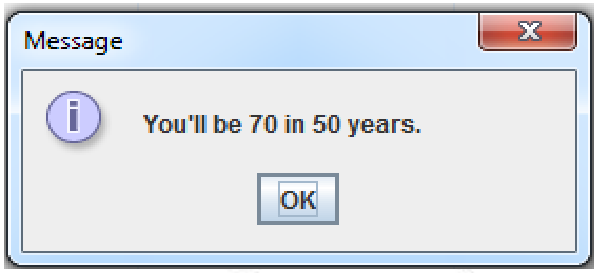
- We can use the following methods to parse a String to the needed type:
  - Integer.parseInt(str): Returns the integer represented by the given String as an int.
  - Double.parseDouble(str): Returns the real number represented by the given String as a double.
  - Boolean.parseBoolean(str): Returns the boolean values represented by the given String (if the text is "true", returns true; otherwise, returns false).
- Each of these method calls could potentially throw a NumberFormatException. To protect against invalid input we can use the following try/catch structure. Let's examine how our Age.java from an earlier lectures would use JOptionPane:

```
import javax.swing.*;

/**
 * Calculates number of years until 70
 *
 * @author Jessica Young Schmidt
 */
public class AgeWithOptionPane {
    /**
     * Starts the program
     *
     * @param args array of command line arguments
     */
    public static void main(String[] args) {
        String ageText = JOptionPane.showInputDialog(null, "How old are you?");
        int age = -1;
        try {
            age = Integer.parseInt(ageText);
        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(null, "Invalid integer.");
            System.exit(1);
        }
        JOptionPane.showMessageDialog(null, "You'll be 70 in " + (70 - age) + " years.");
    }

}
```


Input

How old are you?

20


Message

You'll be 70 in 50 years.

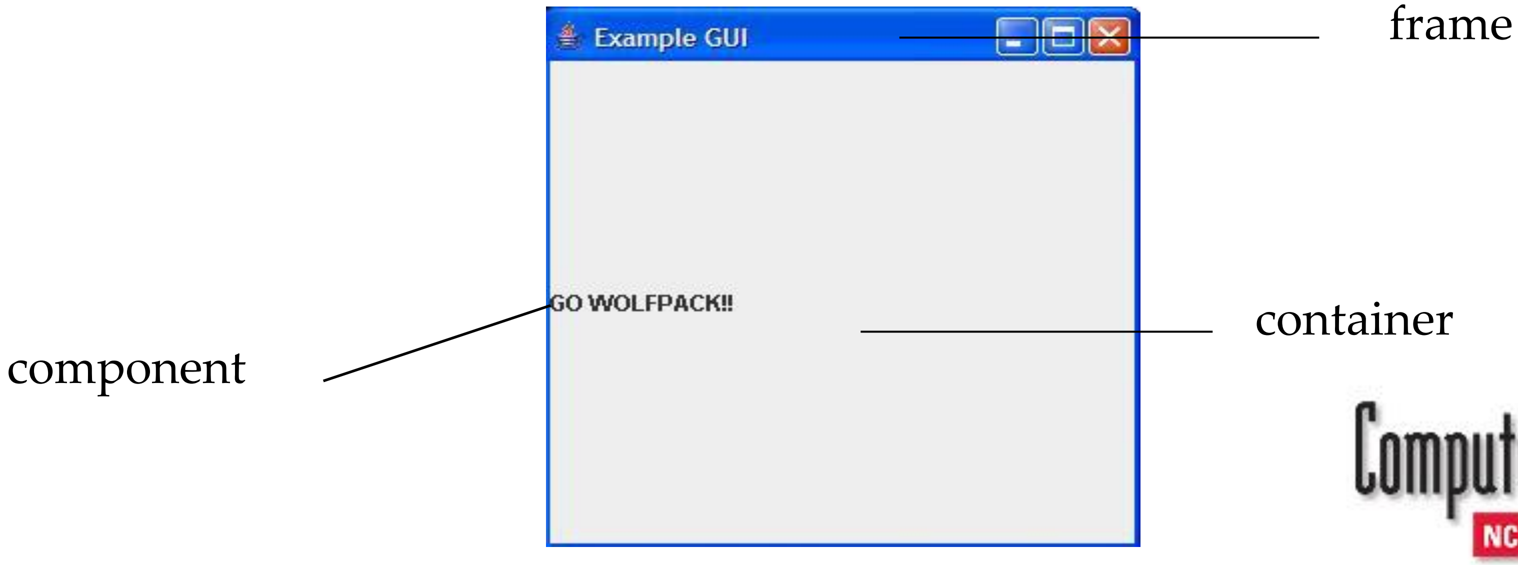
`JOptionPane`

- Advantages:
  - simple
  - flexible (in some ways – three types of panes)
  - looks better than the terminal window

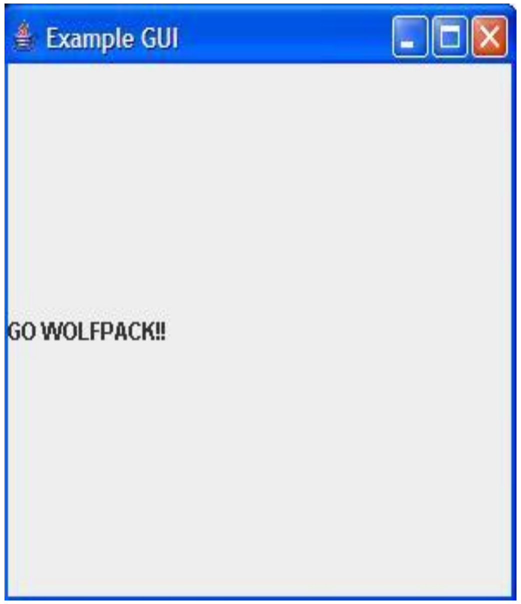
- Disadvantages:
  - created with static methods, which is not a very object-oriented approach
  - not very powerful (just simple dialog boxes)

GUI elements

- Unfortunately, JOptionPane is not flexible enough to create rich GUIs.
- Instead, complex GUIs contain the following elements:
  - **frame**: A graphical window on the screen.
  - **components**: GUI widgets such as buttons or text fields.
  - **containers**: Logical groups of components.



The Frame

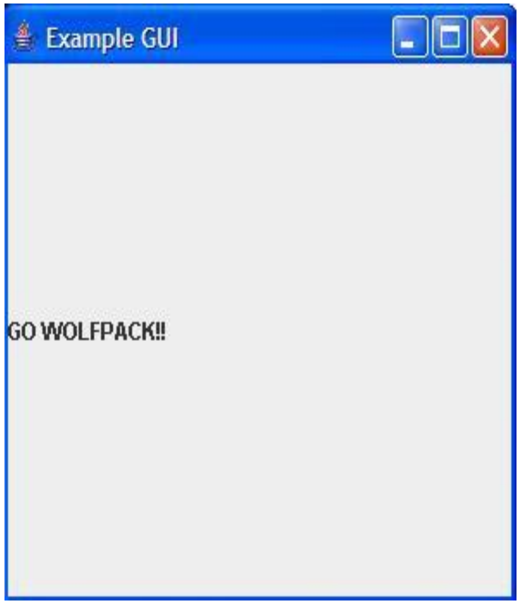


- The “Frame” portion of a GUI is the title bar (which includes the min/max and close window buttons) along with the edges of the GUI

- Think of it in terms of a white board, where the silver metal border would be the frame

Computer Science
NC STATE UNIVERSITY

The Container



- The Container is the part of the GUI that is located inside the frame. It is just a place for us to place individual components.

- This would be the white area of the white board

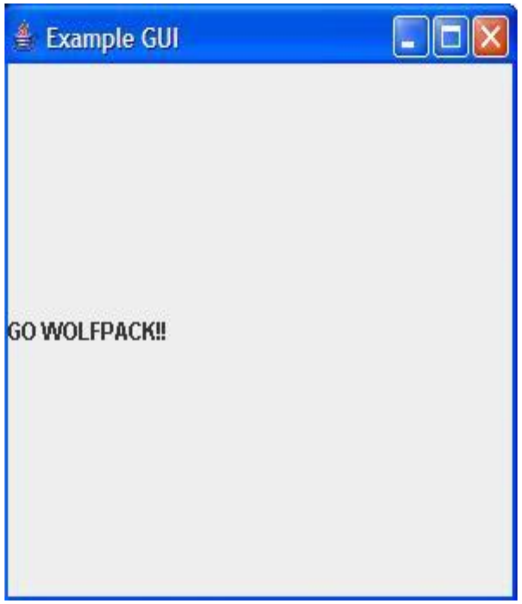
- Without it, you would not be able to write on the white board

- Likewise, without a container, you could not place any components in your GUI

Components

- Components are the individual items you see in the GUI, including:
  - Text
  - Buttons
  - Check boxes
  - Combo boxes
  - Radio buttons
  - MANY MORE!

- Components allow the user to interact with the GUI



Computer Science
NC STATE UNIVERSITY

Java GUI: AWT and Swing

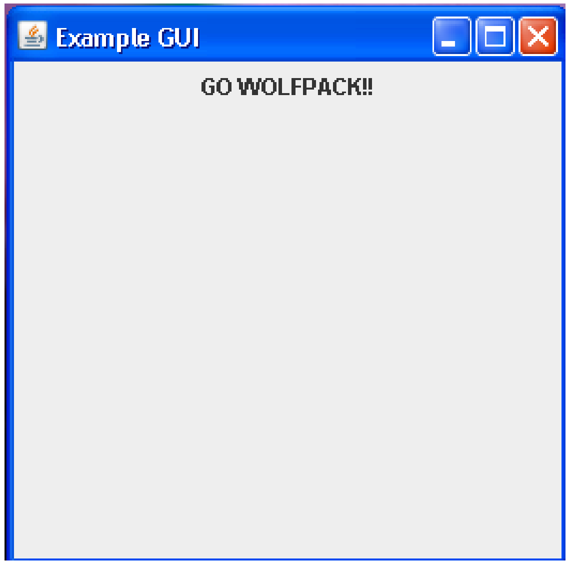
- Sun's initial idea: create a set of classes/methods that can be used to write a multi-platform GUI (Abstract Windowing Toolkit, or **AWT**)
  - problem: not powerful enough; limited; a bit clunky to use
- Second edition (JDK v1.2): **Swing**
  - a newer library written from the ground up that allows much more powerful graphics and GUI construction
  - Drawback: Both exist in Java now; easy to get them mixed up; still have to use both sometimes!

The Code from the Example

- Download the ExampleGUI.java file from the moodle page and open it in an editor.
- Note that we need to **import java.awt.* and javax.swing.*** in order for our program to find the JFrame, Container, and all the component classes.
- Compile and execute!

```
import java.awt.*; // imports the Container class
import javax.swing.*; // imports JFrame and component classes

public class ExampleGUI extends JFrame  { // tells Java this
                                          // is a GUI application
   public ExampleGUI() {
     setTitle("Example GUI");  // sets the title of the frame
     setSize(300,300);  //frame height and width (in pixels)
     setLocation(100,100); // (x,y) of the frame in the screen
     setDefaultCloseOperation(EXIT_ON_CLOSE);  // tells the
                         // program to exit when the x is clicked
     Container c = getContentPane();  // creates a container
                                     // for our frame
     c.setLayout(new FlowLayout());  // container layout
     JLabel label = new JLabel("GO WOLFPACK!!");  //creates a
                                                  //label
     c.add(label);  // adds the label to the container
     setVisible(true);  // makes the frame visible
   }

   public static void main(String args[]) {// creates a gui
      ExampleGUI gui = new ExampleGUI();
  }
}
```

The Example


Example GUI

GO WOLFPACK!!

- The example is a very basic GUI
- It creates a Frame, a container, and a single component (a label), and shows the GUI to the user
- Let's see more components!

Text Fields

- A text field is an area in a GUI where a user may enter a line of text.
- A very common example is the address bar in a web browser for a user to enter a URL
- The class that we will use to create text fields is called **JTextField**

Making a JTextField

- To make a JTextField, we simply create an object of the JTextField class.   When we create the object, we can provide the starting text for the field to contain, and the maximum number of characters that can be shown in it (the size of the text field) :

- JTextField field = new JTextField(“Hello”, 10);

Buttons

- Another common component is the button, which allows a user to click it to perform some action
- In Java, we will use the **JButton** class to create buttons for our GUIs

Making a JButton

- To make a JButton, we simply create a JButton object and supply the text we want written on the button as a parameter.
- JButton button = new JButton(“CLICK ME”);

Adding The Components to the Container

- Let’s take a moment to add the last two components to our code.
- Creating the text field and button DOES NOT automatically put them in the GUI
- Instead, we have to explicitly add them to the container.
- We do this by writing **c.add(   )**, and write the variable name of the component in the parenthesis
  - – c.add(field);
  - – c.add(button);

Text Areas

- The next component we will learn about is a text area. A text area is place in your GUI where a user can type in several lines of text.
- The area where you type the body of your email is a text area.
- In Java, the class we will use to create text areas is called **JTextArea**

Making a JTextArea

- To create a JTextArea, you simply create an object of the JTextArea class, and give it the text you want it to start with, followed by the number of lines, followed by the size of each line:

- JTextArea area = new JTextArea("This\nIs\nText", 5, 10);

Color

- Now that we have seen the basic components, we can learn about changing color.

- You can change the background and foreground color of all your components, as well as the background color of the container

Changing the Background Color of the Container

- We can change the background of the container by calling the **setBackground( )** method, which accepts a Color object as a parameter which specifies the color you want the container background to be

- To tell the method which color you want to use, you simply write “Color”, followed by a period, followed by the color you wish to use

Examples

- c.setBackground(Color.blue);
- c.setBackground(Color.green);
- c.setBackground(Color.red);
- Try one and recompile!

Foreground vs Background

- We can change the color of components and also the color of the area around the components

- Background color is the color “behind” the component

- Foreground color is the color "of " the component, such as the color of text

Changing the Background Color of a Component

- Some components, such as text areas, buttons, and text fields, can have their backgrounds changed. Others, such as labels, cannot.

- To do this, simply write the variable name of the component, followed by a period, followed by the setBackground( ) method call that contains the color you want as a parameter.

Examples

- field.setBackground(Color.yellow);
- button.setBackground(Color.blue);

Changing the Foreground Color of Components

- We can change the foreground color of components almost the same way we set the background

- Simply write the variable name of the component, followed by a period, followed by **setForeground( )**, where the color is specified as a parameter

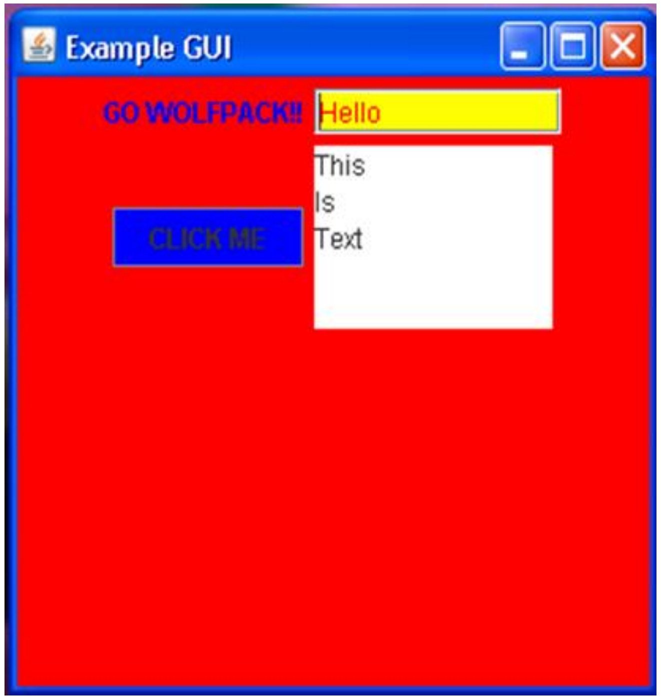
Computer Science
NC STATE UNIVERSITY

Examples

- label.setForeground(Color.blue);
- field.setForeground(Color.red);
- Add the lines and recompile!

Computer Science
NC STATE UNIVERSITY

```
import java.awt.*; // imports the Container class
import javax.swing.*; // imports JFrame and component classes

public class ExampleGUI extends JFrame {

    public ExampleGUI() {
        setTitle("Example GUI"); // sets the title of the frame
        setSize(300,300); //height and width of the frame (in pixels)
        setLocation(100,100); // sceen location (x,y) of the frame
        setDefaultCloseOperation(EXIT_ON_CLOSE); // tells the program
                                                   // to exit when the x is clicked
        Container c = getContentPane();
        c.setLayout(new FlowLayout());
        c.setBackground(Color.red);

        Jlabel label = new JLabel("GO WOLFPACK!!"); // creates a label
        JTextField field = new JTextField("Hello", 10);
        Jbutton button = new JButton("CLICK ME");
        JTextArea area = new JTextArea("This\nIs\nText", 5, 10);
        field.setBackground(Color.yellow);
        button.setBackground(Color.blue);
        label.setForeground(Color.blue);
        field.setForeground(Color.red);

        c.add(label); // adds the label to the container
        c.add(field); // adds the text field to the container
        c.add(button); // adds the button to the container
        c.add(area); // adds the text area to the
                        container
        setVisible(true); // makes the frame visible
    }
}
```



Computer Science
NC STATE UNIVERSITY

Component Arrangement

- The ability to place components exactly where you want them is based on the layout of the container

- Notice in our code we have the line:

  c.setLayout(new FlowLayout());

- A flow layout allows us to add components to the container the same way you add words to a sheet of paper:
  - Start in the upper left corner, and add components across the GUI until you run out of room. Then move to the next line and continue

Flow Layout

- In the flow layout, the components appear in the order in which they are added to the container.
- This is not the greatest layout, but we will stick with it for now

Interacting with a GUI

- Now that we can put basic components in our GUI, we can learn how to program it so a user can interact with it

- The most basic form of interaction is having a user click a button, and the program reacting to it

ActionListener

- When we want our GUIs to listen for actions from the user, such as button clicks, we implement the **ActionListener** interface

- To do this, on the same line where we declare our class, we add the words “**implements ActionListener**”

- To use it, we must also **import java.awt.event.***, which is the location of ActionListener

Example

```
public class ExampleGUI extends JFrame
        implements ActionListener {
….


}
```

The actionPerformed Method

- Any time you implement the ActionListener interface, you are **REQUIRED** to add the following method to your class:

```
public void actionPerformed(ActionEvent e) {

}
```

- Your program will not compile if you do not have this method!

How it Works

- Any time a user interacts with a component that is “listening” for user input, the actionPerformed method gets called

- Thus, if a button is listening, and a user clicks it, any code written in the actionPerformed method will be executed

Telling a Component to Listen For User Interaction

- Components do not automatically listen for user input; we must tell them to
- To tell a component to listen, simply write the variable name of the component, followed by a period, followed by **addActionListener(this);**
- **Example:**
  - **button.addActionListener(this);**
- Add this before adding button to the container (after button creation)

Add The Code!

- Add the code to your file, and add a single println statement in the actionPerformed method:

```
System.out.println("Button clicked!");
```

- Recompile, and click the button a few times

More On actionPerformed

- For the moment, we are just printing some text to the console when the button is clicked

- We can do anything in actionPerformed, such as make a calculation based on user input in other components

- First, we need to learn how to read information from text fields and text areas!

Reading from a text field/area

- When you want to read from a text field or area, you simply write the variable name of the text field or text area, followed by a period, followed by **getText();**

- This will read whatever is in the field and return it as a String. For example:

- **String s = field.getText();**

A Problem!

- Notice that all of our components were declared inside the constructor.
- This means if we want to use them in actionPerformed, we can’t!
- Thus, we need to declare all of our components as instance variables (and remove type from initialization).
- Let’s make that change right now!

```
    private JLabel label;
    private JTextField field;
    private JButton button;
    private JTextArea area;
```

Updating actionPerformed

- Let’s change actionPerformed so that when the button is clicked, it will print to the screen the contents of both the text field and the text area.
- Let’s see the code…

Updated Code

```
public void actionPerformed(ActionEvent e)  {
    System.out.println("Button clicked!");
    String fieldText = field.getText();
    String areaText = area.getText();

    System.out.println("The textfield says: " + fieldText);
    System.out.println("The textarea says: \n " + areaText);

}
```

- Recompile and see it work!

Setting The Text

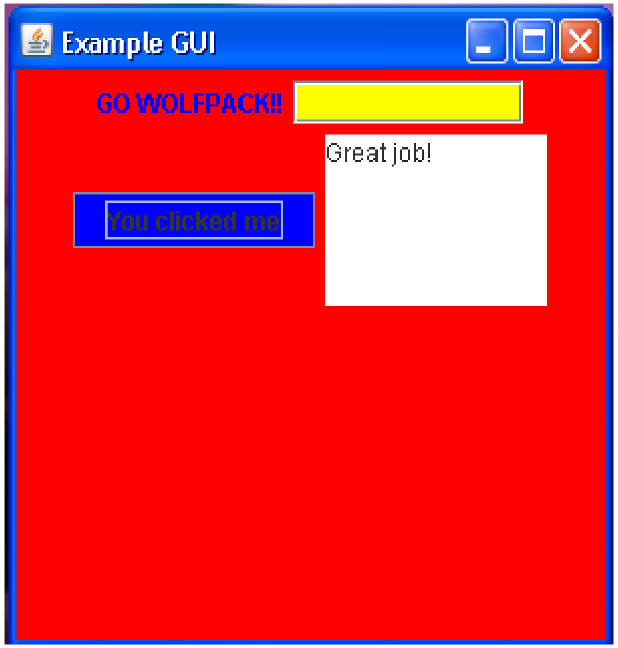
- Sometimes we want to change the text written in a text area, or of a label or button

- To do this, we simply write the name of the component, a period, and **setText( )**

- We put a String inside the parenthesis that contains the text we want to change the component to

Code

- button.setText(“You clicked me”);
- field.setText(“”);
- String s = “Great job!”;
- area.setText(s);
- Add these lines to the bottom of actionPerformed and recompile!
- Then submit ExampleGUI.java



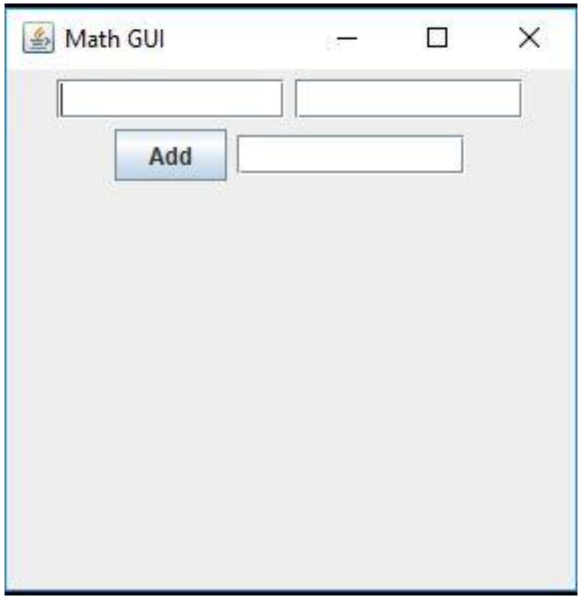
```
public void actionPerformed(ActionEvent e) {
  System.out.println("Button clicked!");
  String fieldText = field.getText();
  String areaText = area.getText();
  System.out.println("The textfield says: " + fieldText);
  System.out.println("The textarea says: \n" + areaText);

  button.setText("You clicked me");
  field.setText("");
  String s = "Great job!";
  area.setText(s);
}
```

Computer Science
NC STATE UNIVERSITY

Example 2

- Download ExampleGUI2.java from the moodle page.

- Notice that we have three text fields – two fields that will contain numbers to be added together, and the third field will contain the sum.



The actionPerformed Method

- Check out the actionPerformed Method:

```
public void actionPerformed(ActionEvent e) {

    String s1 = txtNum1.getText();
    int x1 = Integer.parseInt( s1 );

    String s2 = txtNum2.getText();
    int x2 = Integer.parseInt( s2 );

    int sum = x1 + x2;

    txtNum3.setText("" + sum);
}
```

Building On The Example

- Suppose we want to add another button to multiply the two values.
- Now there will be two buttons that will both trigger the actionPerformed method.
- How will we know which one triggered it so we can take the appropriate action?

The ActionEvent Parameter

- The ActionEvent Parameter holds the source that triggered the actionPerformed method to execute.
- We can ask it who triggered the method in an if statement, and take the appropriate action.
- To ask the Parameter who triggered the method call, we simply call the **getSource()** method of the ActionEvent variable, and compare it to each button to see if they are equal.

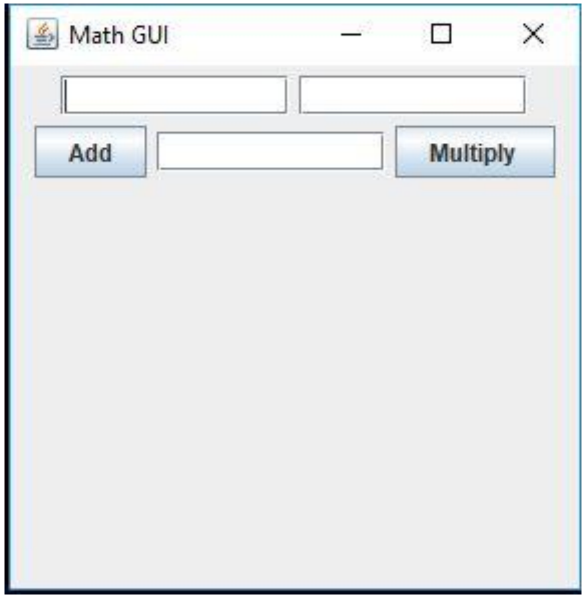
Example

```
public void
    actionPerformed(ActionEvent e) {

    String s1 = txtNum1.getText();
    int x1 = Integer.parseInt( s1 );

    String s2 = txtNum2.getText();
    int x2 = Integer.parseInt( s2 );

    int result = 0;

    if( e.getSource() == add ) {

         result = x1 + x2;

    } else if ( e.getSource() ==
      multiply){
         result = x1 * x2;
    }

    txtNum3.setText("" + result);
}
```

- Add the JButton “multiply” to the GUI.
  - private JButton multiply;
  - multiply = new JButton ("Multiply ");
  - multiply.addActionListener(this);
  - c.add(multiply);

Compile and execute.

Submit ExampleGUI2.java

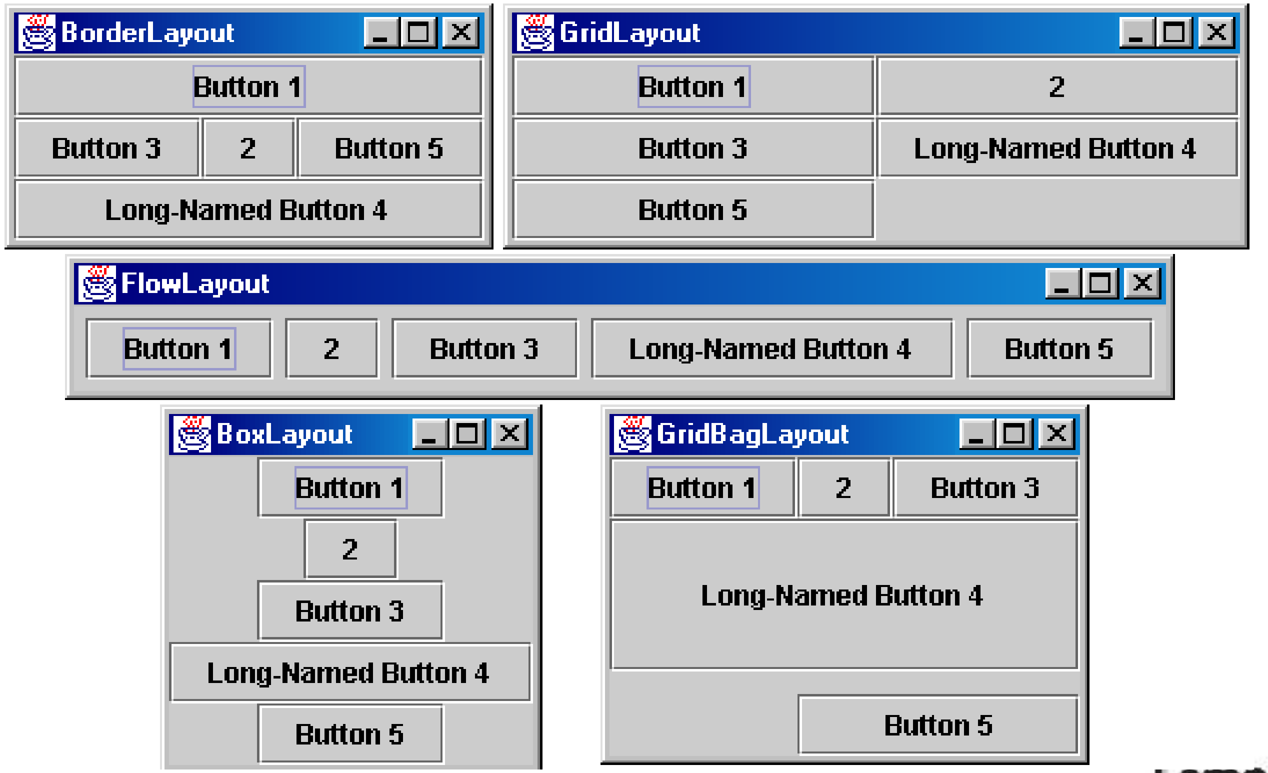


Messy GUIs

- Now that we have several components in our GUI, it is beginning to look a little cluttered.
- It is ideal to place components in the GUI in a way such that things do not appear cluttered.
- We can alter the way components are added to the GUI through the use of Layouts.

Layout managers

- Here are several common Java layout managers:



FlowLayout

- The only layout for the container we have talked about thus far is FlowLayout

- It allows us to add components to the container the same way we add words to a piece of paper:
  - Start at the upper left corner
  - Add components from left to right
  - When we run out of space, we move down to the next line and repeat
  - The order in which we add components is the order they appear

Pros and Cons

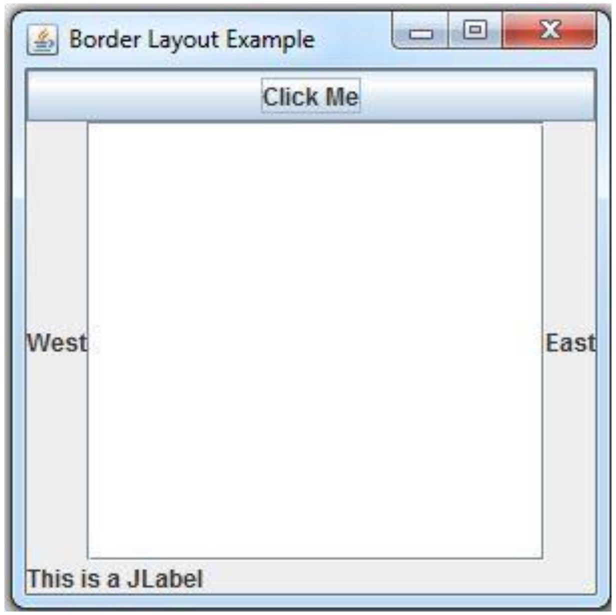
- Pros:
  - Extremely easy to use
  - … that's really it!
- Cons:
  - Very hard to get the GUI to look how you want it to
  - Changing the size of the GUI changes the appearance of the components

BorderLayout

- To place components where you want them to appear, it is better to use a BorderLayout
- BorderLayout is the default layout of a container
- It allows you to place a component in each of 5 areas: NORTH, SOUTH, EAST, WEST, CENTER
- Take a look at BorderLayoutGUI.java (on the moodle page)

Border Layout Example

```
public class BorderLayoutGUI extends JFrame {
   public BorderLayoutGUI()  {
          setSize(300,300);
          setTitle("Border Layout Example");
          setDefaultCloseOperation(EXIT_ON_CLOSE);

          Container c = getContentPane();
          JButton b = new JButton("Click Me");
          JLabel l = new JLabel("This is a JLabel");
          JTextField f = new JTextField("", 10);

          JLabel e = new JLabel("East");
          JLabel w = new JLabel("West");

          c.add(b, BorderLayout.NORTH);
          c.add(l, BorderLayout.SOUTH);
          c.add(f, BorderLayout.CENTER);
          c.add(e, BorderLayout.EAST);
          c.add(w, BorderLayout.WEST);

          setVisible(true);
}

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



Pros and Cons

- Pros:
  - You can tell a component where to appear
  - The order you add a component does not change anything
  - Easy to implement
- Cons:
  - One component per area
  - Changing the size of the GUI changes the size of the components (though it does not alter their arrangement)

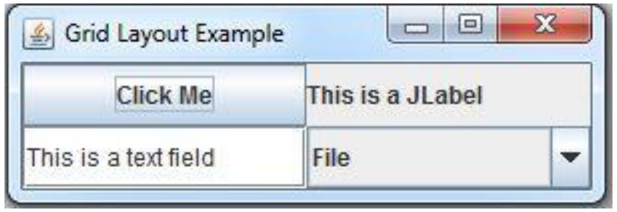
GridLayout

- The GridLayout allows us to specify how many rows and columns of components we want to have, and each component added is placed into the "grid"

- Let's look at the GridLayoutGUI.java code (on moodle page)

Grid Layout Example

```
public class GridLayoutGUI extends JFrame {
     public GridLayoutGUI() {
                         setSize(300,100);
                         setTitle(“Grid Layout Example");
                         setDefaultCloseOperation(EXIT_ON_CLOSE);
                         Container c = getContentPane();
                         c.setLayout(new GridLayout(2,2));
                         JButton b = new JButton("Click Me");
                         JLabel l = new JLabel("This is a JLabel");
                         JTextField f = new JTextField("This is a text field", 10);
                         JComboBox cb = new JComboBox();
                         cb.addItem("File");
                         cb.addItem("Edit");
                         cb.addItem("Save");
                         c.add(b);
                         c.add(l);
                         c.add(f);
                         c.add(cb);

                         setVisible(true);
    }
    public static void main(String[] args)    {
        new GridLayoutGUI();
    }
}
```



Pros and Cons

- Pros:
  - Can be extremely useful in laying out components in an organize manner
  - Very easy to use
- Cons:
  - Components grow in size with the GUI

The JPanel Component

- In Layouts other than FlowLayout, we can only put one component in each area, which may not be enough to show everything we want to show
- The JPanel component solves this problem. JPanel is essentially another container that we can add several components to and add it to the main container as a single component
- The JPanel container's default layout is FlowLayout, though you can change that the same way you do the GUI's container

Using JPanel

- Suppose I want to put both a text field and a button in the NORTH area of a BorderLayout

- JTextField field = new JTextField(“”,10);
- JButton button = new JButton(“Click”);
- JPanel panel = new JPanel();
- panel.add(field);
- panel.add(button);
- c.add(panel, BorderLayout.NORTH);

GUI Layout Information

- For more information on GUI Layouts and more sample code, visit the Java tutorial at:

- http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html

Lab Team Exercise

- Go to the moodle page and work on the RationalNumberGUI assignment.
- Start with the given RationalNumberGUI.java program.
  Add the following labels, text fields, and buttons to the panel:

  – num1Label (Numerator 1)
  – num1Text
  – den1Label (Denominator 1)
  – den1Text
  – num2Label (Numerator 2)
  – num2Text
  – den2Label (Denominator 2)
  – den2Text
  – subtractButton (Substract)
  – multiplyButton (Multiply)
  – divideButton (Divide)

Hint: The order that components are added to the panel is important!

Starter Code

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

public class RationalNumberGUI extends JFrame implements ActionListener {
   private JLabel ansLabel;
   private JTextField ansText;
   private JButton addButton;

public RationalNumberGUI() {
      super("Rational Number GUI");
      setSize(500, 500);
      setLocation(100, 100);
      setDefaultCloseOperation(EXIT_ON_CLOSE);
      // Create a JPanel
      JPanel panel = new JPanel(new GridLayout(7, 2));
      // Add JPanel to the frame
      add(panel);
      ansLabel = new JLabel("Answer");
      panel.add(ansLabel);
      ansText = new JTextField(5);
      panel.add(ansText);
      addButton = new JButton("Add");
      panel.add(addButton);
      setVisible(true);
   }

public void actionPerformed(ActionEvent e) {
   }

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

Update with code from GUI Tutorial

Action events with ActionListener

Let's update our GUI for the Add operation:

Add to Constructer above the setvisible(true); statement.

```
addButton.addActionListener(this);
subtractButton.addActionListener(this);
multiplyButton.addActionListener(this);
divideButton.addActionListener(this);
```

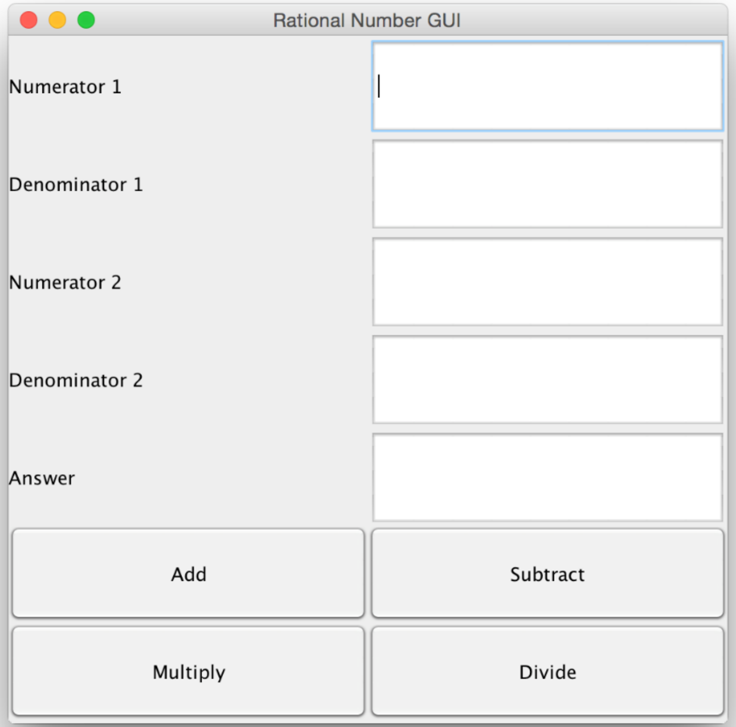
Update actionPerformed method:

```
void actionPerformed(ActionEvent e) {
    if (e.getSource() == addButton) {
        try {
            int num1 = Integer.parseInt(num1Text.getText());
            int num2 = Integer.parseInt(num2Text.getText());
            int den1 = Integer.parseInt(den1Text.getText());
            int den2 = Integer.parseInt(den2Text.getText());
            RationalNumber first = new RationalNumber(num1, den1);
            RationalNumber second = new RationalNumber(num2, den2);
            ansText.setText(first.add(second).toString());
        } catch (NumberFormatException nfe) {
             JOptionPane.showMessageDialog(null, "Invalid integer.");
        } catch (IllegalArgumentException iae) {
            JOptionPane.showMessageDialog(null, "Denominator of zero in given number or result of operation.");
        }
   } if (e.getSource() == subtractButton) {
         System.out.println("Subtract clicked");
   } if (e.getSource() == multiplyButton) {
        System.out.println("Multiply clicked");
   } if (e.getSource() == divideButton) {
        System.out.println("Divide clicked"); }
   }
```

Computer Science
NC STATE UNIVERSITY

Lab Exercise (cont.)

- Add the subtract, multiply, and divide operations on your own.
- Prevent the user from editing the answer text field by adding the following to the constructor (before setVisible(true)): `ansText.setEditable(false);`
- Submit your final version of RationalNumberGUI.java
- Your final functioning GUI should look like:


Rational Number GUI
Numerator 1
Denominator 1
Numerator 2
Denominator 2
Answer
Add
Subtract
Multiply
Divide