

*Graphical User Interface (GUI)
and Object-Oriented Design (OOD)*

A comment on these PPT slides

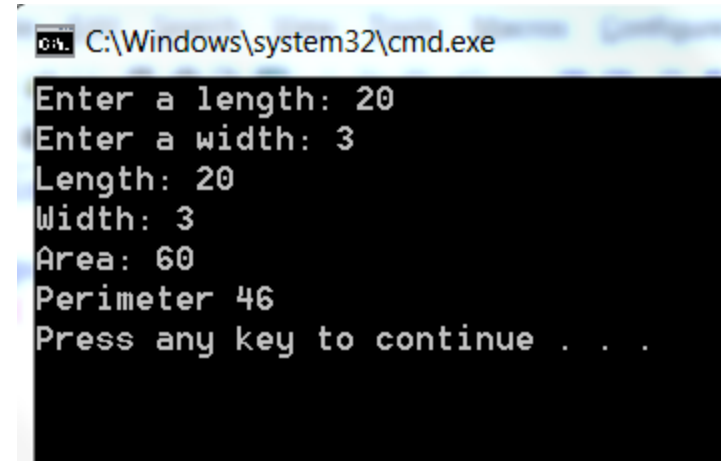
- These slides are being used to teach, not to give a business-style presentation.
- On the whole, there will be less text than in a Word document, but there will also be much more in the way of ‘visuals’.
- Still, I will frequently have “busy” slides in the same way that lecture notes in Word format have lots of information.
- In other words, I’m not going to even try to stick to the “6x6” rule commonly emphasized for giving presentations in Powerpoint format.

Chapter Objectives

- Learn about basic GUI components
- Explore how the GUI components `JFrame`, `JLabel`, `JTextField`, and `JButton` work
- Become familiar with the concept of “event-driven” programming through the use of events and their corresponding event handlers

Rectangle Calculator

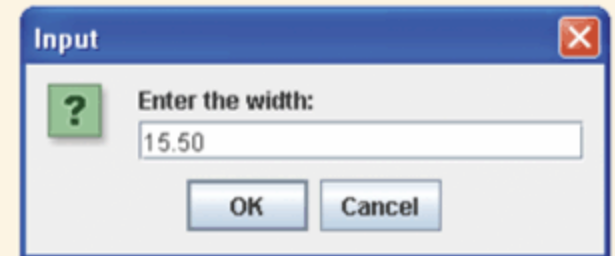
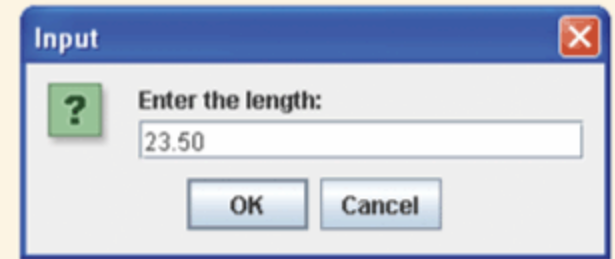
- Uses the console (command window)
- Solution: A better interface
 - The best interfaces are typically graphical in nature (compare DOS v.s. Windows)
 - Some graphical interfaces are better (much!) than others



```
C:\Windows\system32\cmd.exe
Enter a length: 20
Enter a width: 3
Length: 20
Width: 3
Area: 60
Perimeter 46
Press any key to continue . . .
```

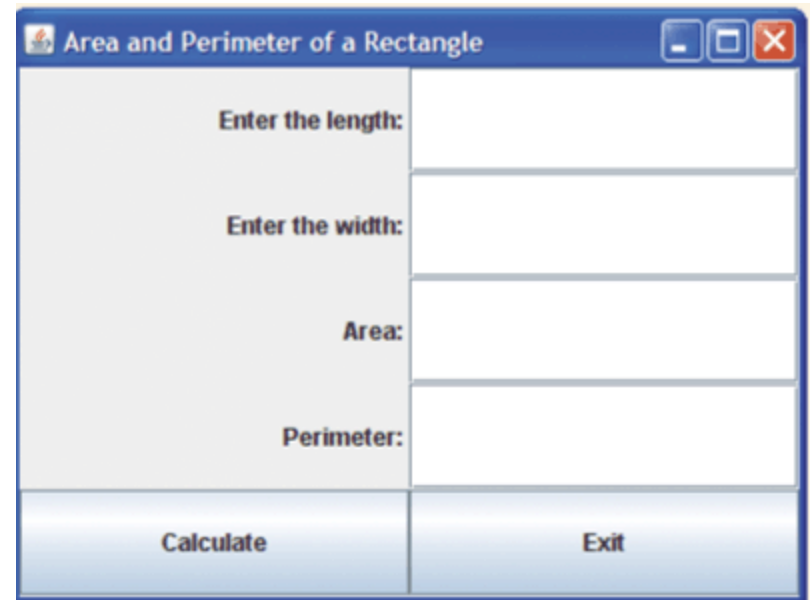
Better Rectangle Calculator

- Uses JOptionPane methods (showInputDialog, showMessageDialog) to read input and output results
 - Could easily do the same with Scanner class
- Limitations:
 - Windows show up one at a time
 - Can't go back to a previous window if you make a mistake
- Solution: Continue to improve the interface



Graphical User Interface (GUI)

- Here is an improved version of the Rectangle calculator using a much better GUI
- View inputs and outputs simultaneously
- One graphical window
- Input values in any order
- Change input values in window
- Click on buttons to get output



Creating a Java GUI

- Java comes prepackaged with many classes that allow you to create GUIs.
- Currently, the favorite way of doing this is by instantiating a series of GUI objects
 - These objects will be instantiated from a series of classes from a package called **javax.swing**
 - You've already seen one of them: The JOptionPane class is from the Swing package
 - So we will now have to get in the habit of importing this package: **import javax.swing.*;**
- There are classes to create buttons (class **JButton**), labels (class **JLabel**), text boxes, windows, scrollbars, frames (class **JFrame**), etc, etc
 - You can instantiate objects of these classes. For each object, you can invoke methods (such as changing the size of a button, change the text of a label, etc)
- We will create our GUIs by instantiating multiple objects from these various classes
- For example, for one single GUI window such as the rectangle calculator above, you might want to instantiate:
 - 1 window (or “frame”)
 - 2 buttons (submit, reset)
 - 2 labels (Enter Height, Enter Width)
 - 2 textfields (space for user to enter the height and width)

Java GUI Components

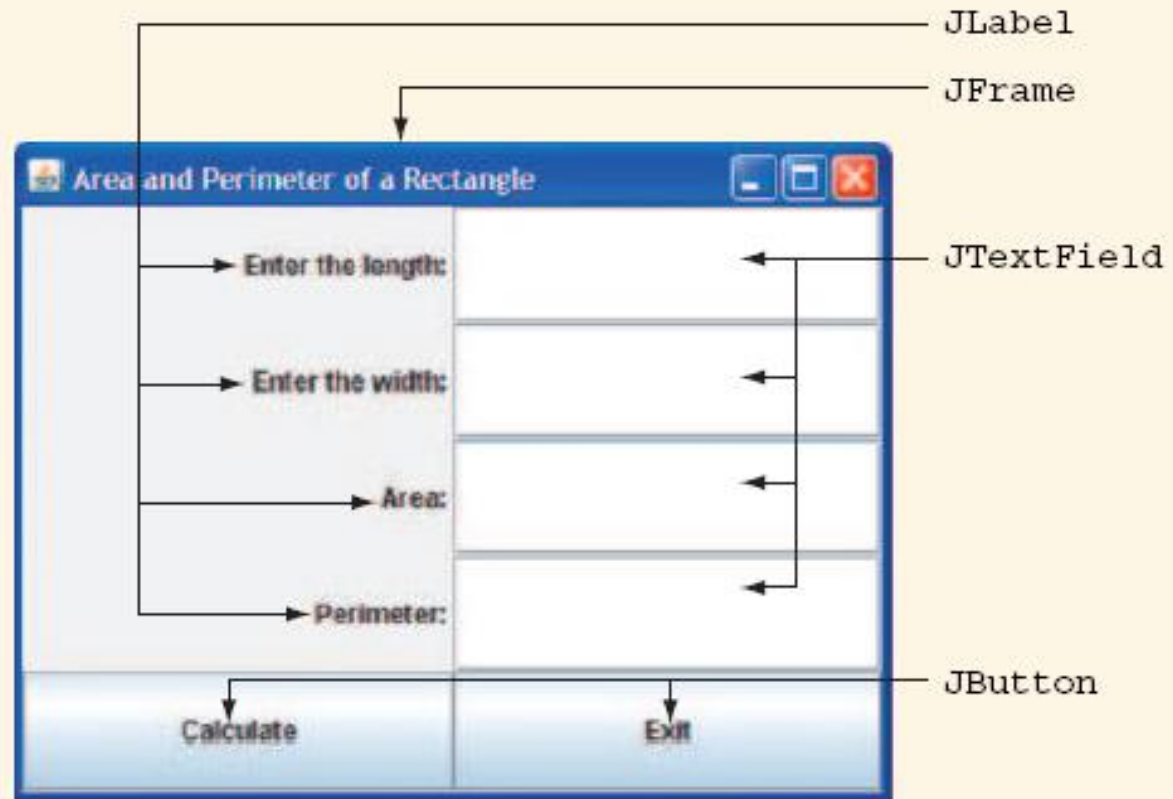
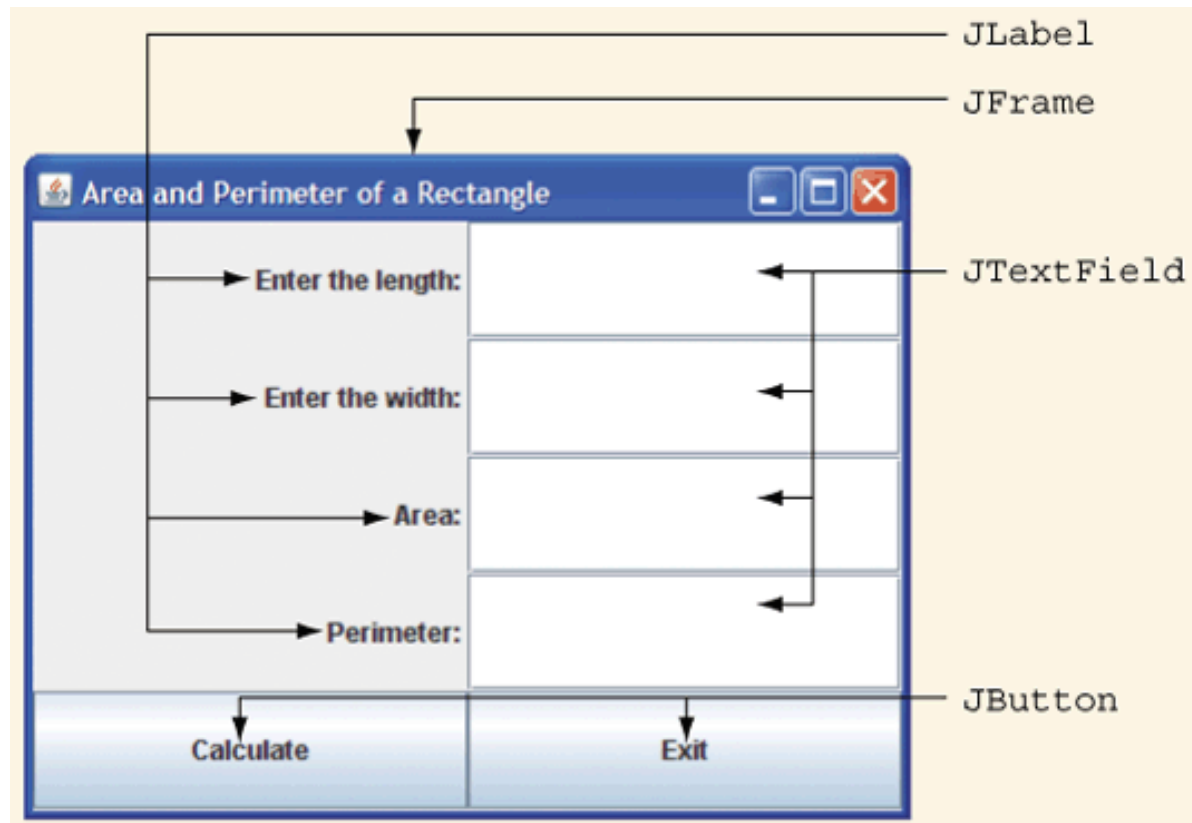


FIGURE 6-3 Java GUI components

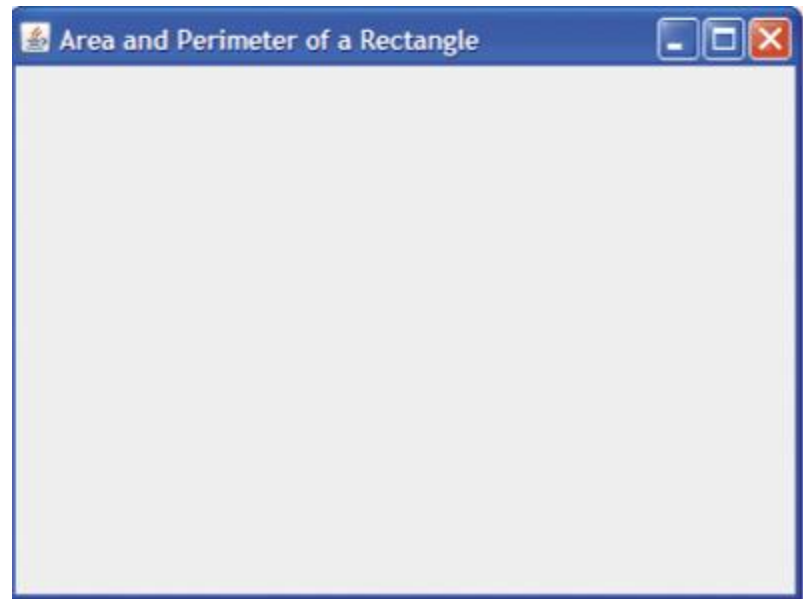
It turns out that creating a user interface such as this one is not terribly difficult in Java. The pre-written classes have done most of the work. All you need is the skill and knowledge to instantiate the various objects and then “paint” them into the window.

We will now turn our attention to creating some of these components, specifically, **Frames** (aka Windows), **Containers**, **Labels**, **Text fields**, and **Buttons**.



The “Frame”: In Java, what we are used to calling a ‘window’ is instead referred to as a **frame**. However, you should not confuse the frame with the components placed *inside* of it. The frame is simply the outer layer. A typical frame is a box and usually has things like open/close/maximize buttons at the top. Frames frequently include a menu bar (File, Edit, View, Help, etc).

Here is a Java frame:



Frame vs Window

- What we are used to calling a Window is, in Java, referred to as a Frame.
- In Java, a window is a very simple frame – one that does not have a title bar or any of the other little buttons we are used to seeing at the top of a gui (minimize, close, menus, etc).
- So your outermost container in your Java apps will typically be a Frame instead of a Window.
- We will create frames by instantiating the class 'JFrame'

Component vs Container

- All GUI objects (buttons, labels, frames, windows, scroll-bars, *containers*, etc, etc) are called **components**.
- A **container** is one type of component. Specifically, it is a component that holds other components.
- In other words, before you can add components such as buttons or labels to your GUI, you must first have a container. You will then add your button/label/etc to that container. A JFrame is one example of a container.

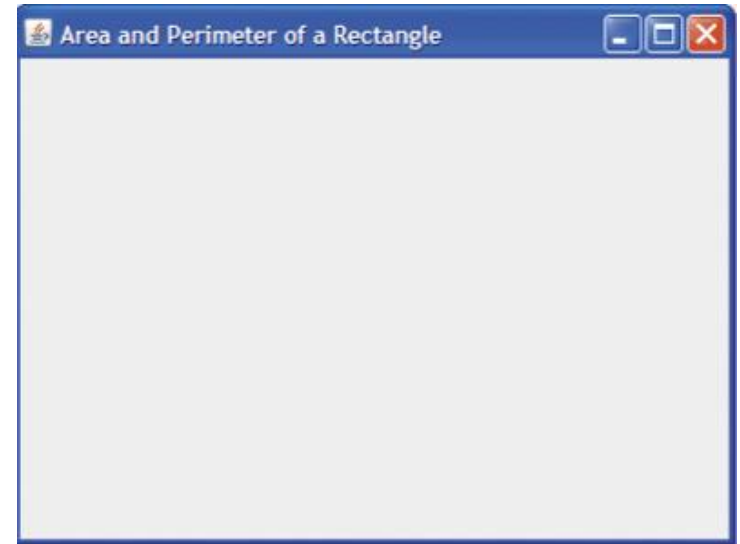
GUI Components

So: To create our GUI (graphical interface), we will:

1. Create a frame
 - Set frame parameters such as exit behavior, size, etc
2. Instantiate the various components for our GUI such as buttons, text fields, labels, etc
3. Add those components to the content pane

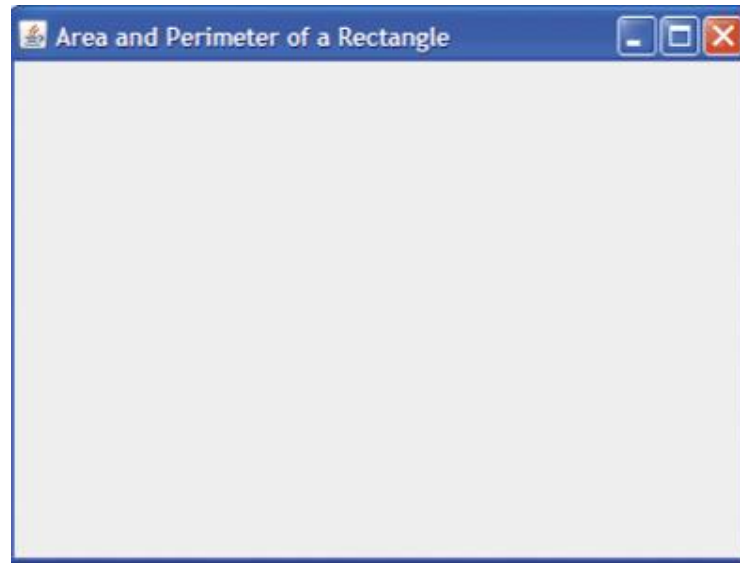
Frames

- Can be created using a `JFrame` object
- The `JFrame` class provides various methods to control attributes (**state**) of a window (e.g. height, width)
 - Size of a window (height and width) is measured in pixels
- Attributes associated with windows
 - Title
 - Width
 - Height
- You can control all of these attributes by invoking various methods of the `JFrame` class ([API](#))
 - Look for methods such as `setTitle()`, `setSize()`, `setVisible()`, `resize()`, etc
 - Many of these are “inherited” from “parent” classes. We will discuss this concept in our Inheritance lecture.



Frames contd

- Create the frame by instantiating the JFrame class
 - Note: Actually *displaying* this frame requires a separate command!!
 - The JFrame class provides several methods that allow us to change the attributes (i.e. state) of the frame object such as title, size, and even color.



```
JFrame mainFr = new JFrame("Area and Perimeter of a Rectangle");
```

TABLE 6-1 Some Methods Provided by the `class` JFrame

Method / Description / Example

```
public JFrame ()
```

```
//This is used when an object of type JFrame is  
//instantiated and the window is created without any title.  
//Example: JFrame myWindow = new JFrame();  
//          myWindow is a window with no title
```

```
public JFrame(String s)
```

```
//This is used when an object of type JFrame is  
//instantiated and the title specified by the string s.  
//Example: JFrame myWindow = new JFrame("Rectangle");  
//          myWindow is a window with the title Rectangle
```

```
public void setSize(int w, int h)
```

```
//Method to set the size of the window.  
//Example: The statement  
//          myWindow.setSize(400, 300);  
//          sets the width of the window to 400 pixels and  
//          the height to 300 pixels.
```


Methods Provided by the class JFrame (continued)

```
public void setTitle(String s)
//Method to set the title of the window.
//Example: myWindow.setTitle("Rectangle");
//          sets the title of the window to Rectangle.
```



An important
method

```
public void setVisible(boolean b)
//Method to display the window in the program. If the value of b is
//true, the window will be displayed on the screen.
//Example: myWindow.setVisible(true);
//  After this statement executes, the window will be shown
//  during program execution.
```



An important
method

```
public void setDefaultCloseOperation(int operation)
//Method to determine the action to be taken when the user clicks
//on the window closing button, ×, to close the window.
//Choices for the parameter operation are the named constants –
//EXIT_ON_CLOSE, HIDE_ON_CLOSE, DISPOSE_ON_CLOSE, and
//DO_NOTHING_ON_CLOSE. The named constant EXIT_ON_CLOSE is defined
//in the class JFrame. The last three named constants are defined in
//javax.swing.WindowConstants.
//Example: The statement
//          setDefaultCloseOperation(EXIT_ON_CLOSE);
//sets the default close option of the window closing to close the
//window and terminate the program when the user clicks the
//window closing button, ×.
```

Two Ways to Create a Frame

- First way (*the way we will use*):
 - Declare and instantiate an object of type `JFrame`
 - Use various methods to manipulate window
- Another way (used by some books):
 - Create class containing application program by extending definition of `class JFrame`
 - Utilizes mechanism of inheritance
 - Not something we're going to worry about now...
 - NOTE: The Malik textbook uses with this second way. However because we haven't discussed inheritance (for that matter, the book hasn't either!) we're going to use a different and simpler version of this code than in your book. So for now you should focus on the techniques we discuss here. If at some later point you decide you prefer to use the inheritance mechanism, that's fine, there are plenty of examples out there to show you how it's done.

The following code will display the window we've been discussing:

```
import javax.swing.*;

public class DrawWindow
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame("Rectangle Calculator");
        frame.setSize(400, 300); //what is the default size?
        frame.setVisible(true);
        frame.setDefaultCloseOperation(3); //Huh?? Why the 3?
    } //end main()


} //end class
```

Try this at home:

1. Comment out all of the lines after the instantiation and see what happens.
2. Then add them back one by one. Compile and run each time.

```
public class DrawWindow
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame("Rectangle Calculator");
        frame.setSize(400, 300);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(frame.EXIT_ON_CLOSE);
    } //end main()
} //end class
```

Compare the highlighted code with the previous slide. Why do we change the perfectly valid ‘3’ into this ugly looking constant?

Answer: Clarity. The method setDefaultCloseOperation accepts one argument of type int. The people who wrote the JFrame class, assigned the value ‘3’ to tell the window to close when the ‘x’ is clicked. 

(By default, the ‘x’ simply hides the frame; it is still sitting there, running in the background).

[See the API for the setDefaultCloseOperation\(\) method here.](#)

So why use a constant? In spite of the relatively lengthy coding required here, it is much easier to determine what the programmer is trying to accomplish.

This is a very common and useful way of using constants. Be sure to get used to it. You should be comfortable if you see constants used in this manner, and you should start using them this way in your own programs when applicable.

```
public class DrawWindow
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame("Rectangle Calculator");
        frame.setSize(400, 300);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(frame.EXIT_ON_CLOSE);
    } //end main()
} //end class
```

Question: Where does this constant come from?

Answer: This constant is a field of the JFrame class.

Question: Why do we include the ‘frame’ object before the constant?

Answer: If we simply said ‘EXIT_ON_CLOSE’ Java would look for the constant in the current class (DrawWindow).

Question: Is there anything else we could use besides ‘frame’?

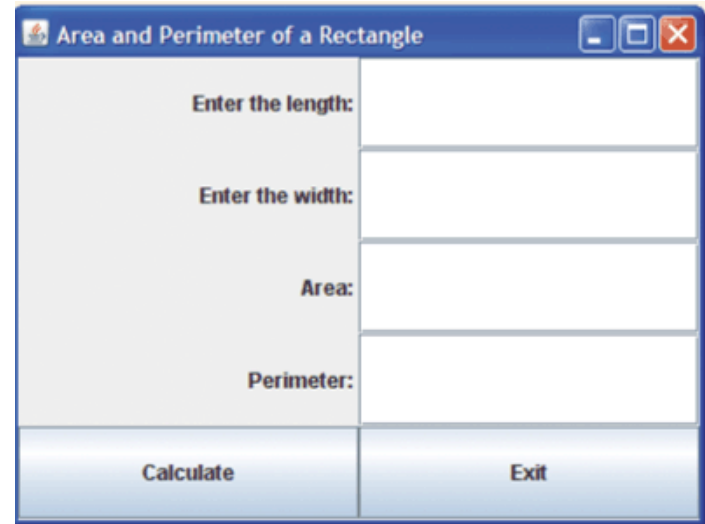
Answer: Yes, we could say ‘JFrame.EXIT_ON_CLOSE’

This works because the constant is a *static* constant. I.e: It is “owned” by the entire class and can therefore be referred to by either the class name, or by any object of that class.

Hmmm, seems these would make for excellent exam questions.

I’d need to ask them in such a way that they were *understood* though – not simply memorized.

Layouts



Note how the window displayed here has a symmetric look to it. There seems to be two columns and five rows. When designing a GUI, the layout is your responsibility.

There are several layouts that are built-in the Swing packages. The particular layout you see here is called a “**grid layout**”. There are several different layouts you can choose from.

You should typically declare your preferred layout for every container (e.g. for every content pane) in your GUI.

If you do not specify a layout, then Java often (though not always) defaults to a layout called ‘Border Layout’.

Some containers default to other kinds of layouts.

Methods Provided by the class Container

So: There are two important methods that you will want to invoke for ALL of your Container objects (such as content panes):

- **'add'** (to add things like buttons, text fields, labels)
- **'setLayout'** (to define the layout)

TABLE 6-2 Some Methods of the `class` Container

Method / Description
<pre>public void add(Object obj) //Method to add an object to the pane</pre>
<pre>public void setLayout(Object obj) //Method to set the layout of the pane</pre>

Layout contd

- As is so common in Java, even abstract concepts such as layouts are represented by classes/objects!
- In other words, to establish a grid layout (mentioned earlier) in your content pane, you will need to instantiate a grid layout object. *Told you there were going to be all kinds of weird objects!*
- So, to set the layout for a Container object (e.g. to set the layout of our frame), the Container class provides the method: ‘setLayout’. We have to pass as an argument to the method, an object of type GridLayout. Here is the code

```
GridLayout gl = new GridLayout(5, 2); //5 rows, 2 columns  
frame.setLayout(gl);
```

You will soon notice, that in most of our programs, we will never need to refer to this GridLayout object again. Therefore, having the reference ‘gl’ is simply not necessary. Yet we still DO need to pass a GridLayout object to the setLayout method. We can do the following:

```
frame.setLayout( new GridLayout(5,2) );
```


Our code so far:

```
import javax.swing.*; //for our GUI components
import java.awt.*;    //for the Container class

public class Test
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame("Rectangle Calculator");
        frame.setSize(400, 300);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.setLayout( new GridLayout(5, 2) );
    } //end main()
} //end class Test
```

Layouts: GridLayout

- Grid layout is just one example of several commonly used layouts. As mentioned, GridLayout organizes a layout into a sort of table with regularly spaced rows and columns. Each cell in the table is given the same size. You might use it if your were creating a calculator (which has a series of identically sized buttons).
- You create a grid layout by instantiating the GridLayout class.
- The example here shows a GridLayout with 5 rows and 4 columns.



Layouts: BorderLayout

- Has five “regions” into which components can be placed: NORTH, SOUTH, EAST, WEST, CENTER

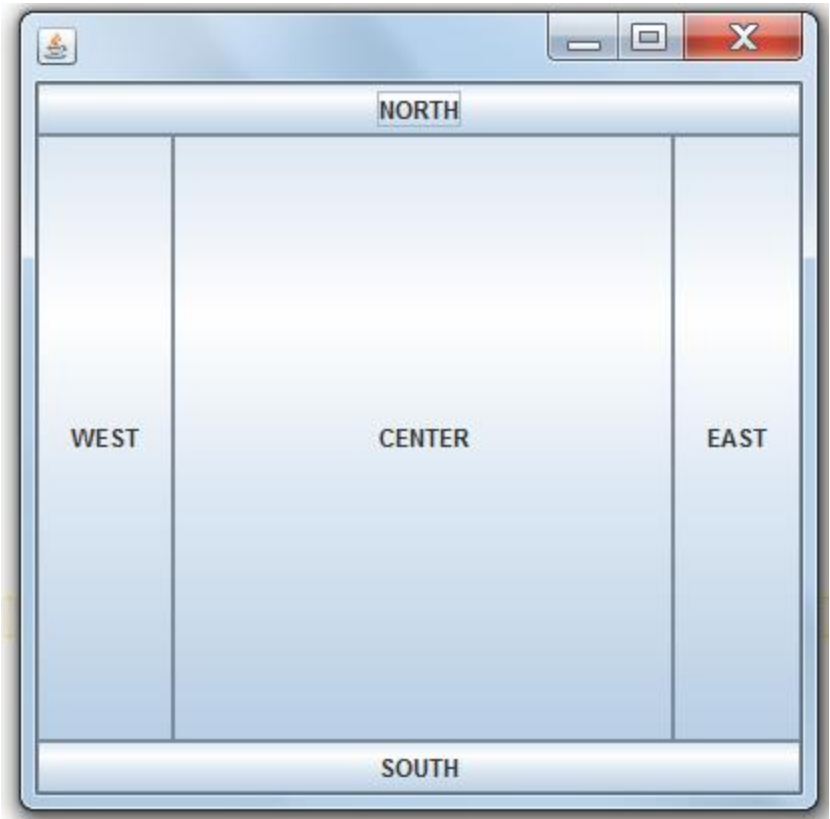
```
frame.setLayout( new BorderLayout() );
```

```
bn = new JButton("NORTH");  
frame.add(BorderLayout.NORTH, bn);  
bs = new JButton("SOUTH");  
frame.add(BorderLayout.SOUTH, bs);
```

```
be = new JButton("EAST");  
frame.add(BorderLayout.EAST, be);
```

```
bw = new JButton("WEST");  
frame.add(BorderLayout.WEST, bw);
```

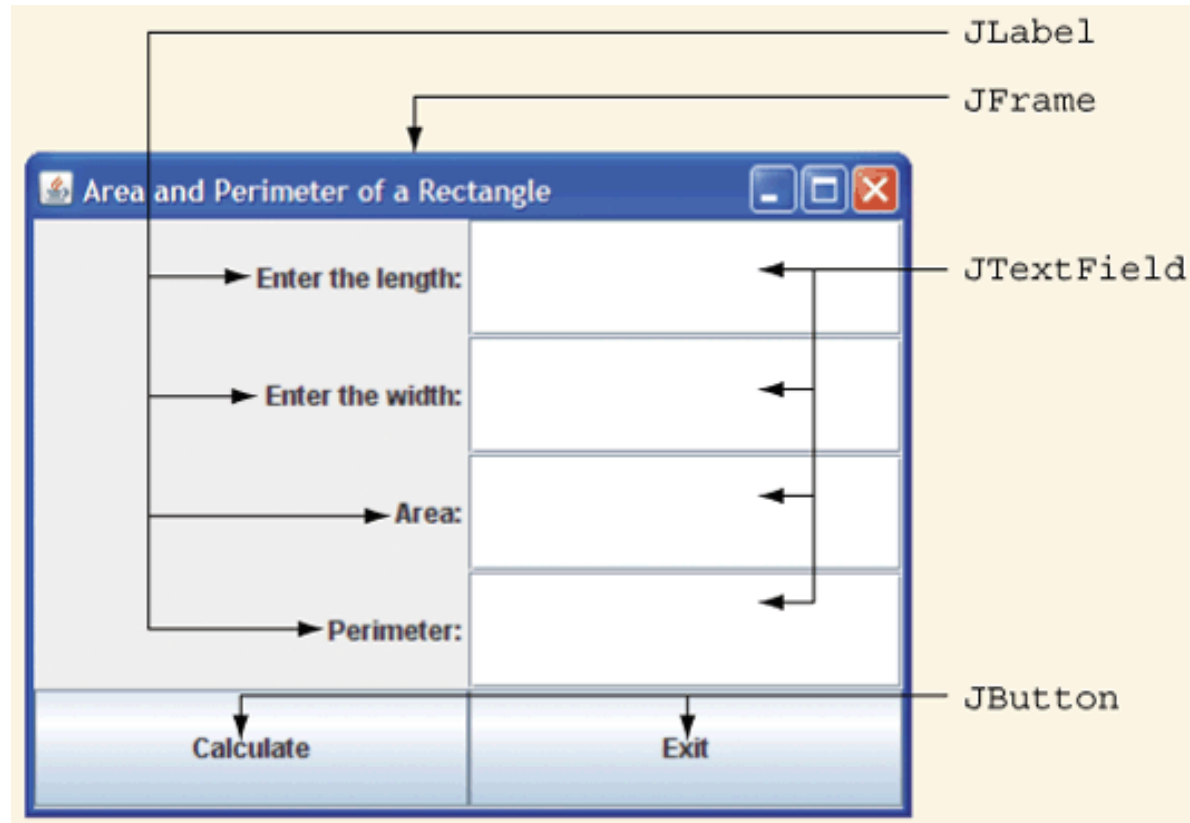
```
bc = new JButton("CENTER");  
frame.add(BorderLayout.CENTER, bc);
```



Several Layout Managers are Available

- In addition to GridLayout and BorderLayout, there are several other layout managers that come packaged with Java.
- Examples are FlowLayout, BoxLayout, GridBagLayout, etc, etc
 - Look up the interface ‘LayoutManager’ in the API. See under ‘All Known Implementing Classes’
- I encourage you to look into them a little bit on your own and experiment.

Labels (using class **JLabel**)



class JLabel (see API [here](#))

Even something as simple as a label is represented by an object. As seen in the previous slide, we need labels to prompt the user to enter the length and the width. We also need labels to refer to the *results*: “Area”, “Perimeter”. So, we will need to instantiate 4 label objects.

At this point, the most important state attribute that interests us for a label is the ‘text’ field (ie. a String)

For our calculator, we’ll instantiate our four labels as follows:

```
JLabel lengthL, widthL, areaL, perimeterL;  
lengthL = new JLabel("Enter the length: ", SwingConstants.RIGHT);  
widthL = new JLabel("Enter the width: ", SwingConstants.RIGHT);  
areaL = new JLabel("The area is: ", SwingConstants.RIGHT);  
perimeterL = new JLabel("Perimeter: ", SwingConstants.RIGHT);  
//Note our naming convention for label objects... USE IT!!!
```

* The second parameter in the constructor is for the alignment. If you look at the API, you’ll see that the second parameter should be an int. The constant RIGHT (a static constant in the SwingConstants class) simply holds the integer ‘4’. However, using the SwingConstants makes things more clear (although admittedly, that may not seem to be the case at the moment).

Adding JLabels: Use the same the 'add()' method:

```
frame.add(lengthL) ;  
frame.add(widthL) ;  
frame.add(areaL) ;  
frame.add(perimeterL) ;
```

FYI: When working with a layout manager such as `GridLayout`, components are added left to right, one row at a time. However, this may not be apparent until you fill all of the cells in the table.

`class JLabel` (partial constructor list from the API)

TABLE 6-3 Some Methods Provided by the `class JLabel`

Method / Description/ Example

```
public JLabel(String str)
    //Constructor to create a label with left-aligned text specified
    //by str.
    //Example: JLabel lengthL;
    //          lengthL = new JLabel("Enter the length:")
    //          Creates the label lengthL with the title Enter the length:
```

```
public JLabel(String str, int align)
    //Constructor to create a label with the text specified by str.
    //    The value of align can be any one of the following:
    //    SwingConstants.LEFT, SwingConstants.RIGHT,
    //    SwingConstants.CENTER
    //Example:
    //    JLabel lengthL;
    //    lengthL = new JLabel("Enter the length:",
    //                          SwingConstants.RIGHT);
    //    The label lengthL is right aligned.
```

```
public JLabel(String t, Icon icon, int align)
    //Constructs a JLabel with both text and an icon.
    //The icon is to the left of the text.
```

```
public JLabel(Icon icon)
    //Constructs a JLabel with an icon.
```


Our code so far:

```
public static void main(String[] args)
{
    JFrame frame = new JFrame("Rectangle Calculator");
    frame.setSize(400, 300);
    frame.setVisible(true);
    frame.setDefaultCloseOperation(frame.EXIT_ON_CLOSE);

    frame.setLayout( new GridLayout(5, 2) );

    JLabel lengthL, widthL, areaL, perimeterL;
    lengthL = new JLabel("Enter the length: ",
                        SwingConstants.RIGHT);
    widthL = new JLabel("Enter the width: ", SwingConstants.RIGHT);
    areaL = new JLabel("The area is: ", SwingConstants.RIGHT);
    perimeterL = new JLabel("Perimeter: ", SwingConstants.RIGHT);

    frame.add(lengthL);
    frame.add(widthL);
    frame.add(areaL);
    frame.add(perimeterL);
} //end main()
```



Rectangle Area & Perimeter



Enter the length:

Enter the width:

The area is:

Perimeter:

class JTextField

A text field is that familiar window such as from web pages where you type in information such as your name and address.

The naming convention we will use **and that I want you to use** is to add the suffix ‘TF’ to the end of your text-field identifiers. e.g. `widthTF`

One important attribute for a text field is the **columns** (how long the field should be). This value is the maximum number of characters that can be entered in the text field.

Other attributes include the ‘**editable**’ state (a boolean) which refers to whether or not the user should be *allowed* to enter information in the textbox. (Sometimes you don’t want them to).

Another attribute is ‘**text**’ which allows you to display some pre-written text in the text field.

TABLE 6-4 Some Methods of the `class` JTextField

Method / Description

<pre>public JTextField(int columns) //Constructor to set the size of the text field.</pre>
--

<pre>public JTextField(String str) //Constructor to initialize the object with the text specified //by str.</pre>

So we might add the following:

```
JTextField lengthTF, widthTF, areaTF, perimeterTF;
lengthTF = new JTextField(10);
widthTF = new JTextField(10);
areaTF = new JTextField(10);
areaTF.setEditable(false); //this TF is to display output
perimeterTF = new JTextField(10);
perimeterTF.setEditable(false); //this TF is to display output
```

- The '10' in the constructor initializes the text field to a maximum of 10 characters.
- The next step is to add these text-field components to our content pane.

TABLE 6-4 Some Methods of the `class` `JTextField` (continued)

Method / Description

<pre>public JTextField(String str, int columns) //Constructor to initialize the object with the text specified //by str and to set the size of the text field.</pre>
--

<pre>public void setText(String str) //Method to set the text of the text field to the string specified //by str.</pre>

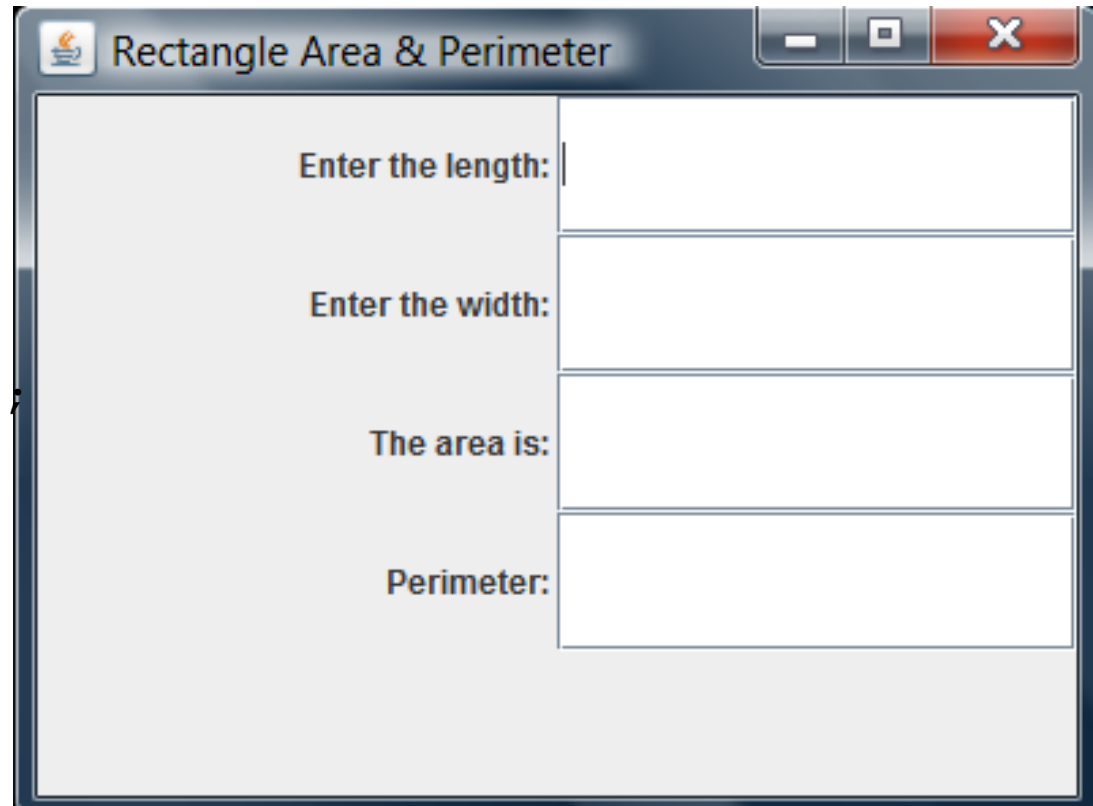
<pre>public String getText() //Method to return the text contained in the text field.</pre>

<pre>public void setEditable(boolean b) //If the value of the boolean variable b is false, the user cannot //type in the text field. //In this case, the text field is used as a tool to display //the result.</pre>
--

<pre>public void addActionListener(ActionListener obj) //Method to register a listener object to a JTextField.</pre>
--

```
JTextField lengthTF, widthTF, areaTF, perimeterTF;  
lengthTF = new JTextField(10);  
widthTF = new JTextField(10);  
areaTF = new JTextField(10);  
perimeterTF = new JTextField(10);
```

```
frame.add(lengthL);  
frame.add(lengthTF);  
frame.add(widthL);  
frame.add(widthTF);  
frame.add(areaL);  
frame.add(areaTF);  
frame.add(perimeterL);  
frame.add(perimeterTF);
```



The image shows a Java Swing window titled "Rectangle Area & Perimeter". The window has a standard Mac OS-style title bar with minimize, maximize, and close buttons. The main content area is light gray and contains four labels and four corresponding text input fields arranged vertically. The labels are "Enter the length:", "Enter the width:", "The area is:", and "Perimeter:". Each label is followed by a white text input field with a thin gray border. The input fields are currently empty.

Note the order in which the components have been added to the layout

class JButton

TABLE 6-5 Commonly Used Methods of the `class` JButton

Method / Description

```
public JButton(Icon ic)
    //Constructor to initialize the button object with the icon
    //specified by ic.
```

```
public JButton(String str)
    //Constructor to initialize the button object to the text specified
    //by str.
```

```
public JButton(String str, Icon ic)
    //Constructor to initialize the button object to the text specified
    //by str and the icon specified by ic.
```

```
public void setText(String str)
    //Method to set the text of the button to the string specified by str.
```

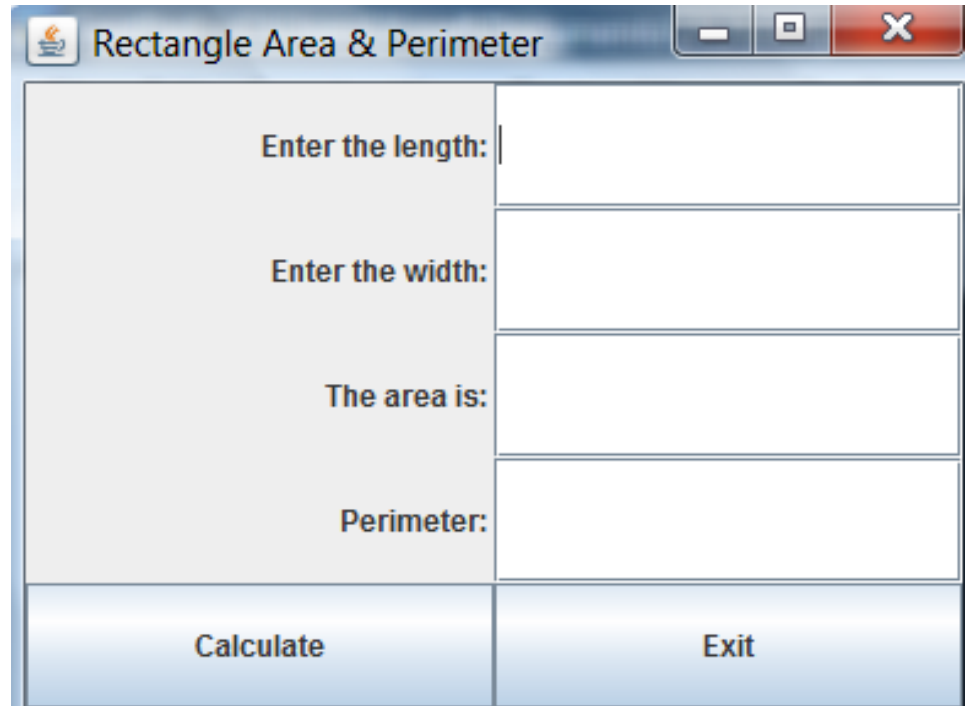
```
public String getText()
    //Method to return the text contained in the button.
```

```
public void addActionListener(ActionListener obj)
    //Method to register a listener object to the button object.
```

Here is the code that generates the buttons, followed by the code to add all the various components to the content pane. Note the order in which the components are added. Recall that when adding components to a **GridLayout**, the components are added from left to right, row by row.

```
JButton calculateB, exitB;  
calculateB = new JButton("Calculate");  
exitB = new JButton("Exit");
```

```
frame.add(lengthL);  
frame.add(lengthTF);  
frame.add(widthL);  
frame.add(widthTF);  
frame.add(areaL);  
frame.add(areaTF);  
frame.add(perimeterL);  
frame.add(perimeterTF);  
frame.add(calculateB);  
frame.add(exitB);
```



Recap

- What we've done:
 - We created a **Window/Frame** (by instantiating the JFrame class)
 - We set up a **layout manager** (in this case, a GridLayout) for our frame by invoking the `setLayout()` from the Container class
 - We created several components for our GUI such as **labels, text fields, buttons**
 - We added the various components to our container
- The summary of our code is on the next slide

Here is the code to generate a fairly nice GUI for our rectangle calculator.

Feel free to copy it into your compiler so that you can actually *see* it! Then compile and execute the program.

Then click on the buttons and observe what takes place.

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

public class RectangleCalculator
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame("Rectangle Area & Perimeter");
        frame.setSize(400, 300);
        frame.setDefaultCloseOperation(frame.EXIT_ON_CLOSE);

        frame.setLayout( new GridLayout(5, 2) );

        JLabel lengthL, widthL, areaL, perimeterL;
        lengthL = new JLabel("Enter the length: ", SwingConstants.RIGHT);
        widthL = new JLabel("Enter the width: ", SwingConstants.RIGHT);
        areaL = new JLabel("The area is: ", SwingConstants.RIGHT);
        perimeterL = new JLabel("Perimeter: ", SwingConstants.RIGHT);

        JTextField lengthTF, widthTF, areaTF, perimeterTF;
        lengthTF = new JTextField(10);
        widthTF = new JTextField(10);
        areaTF = new JTextField(10);
        perimeterTF = new JTextField(10);

        JButton calculateB, exitB;
        calculateB = new JButton("Calculate");
        exitB = new JButton("Exit");

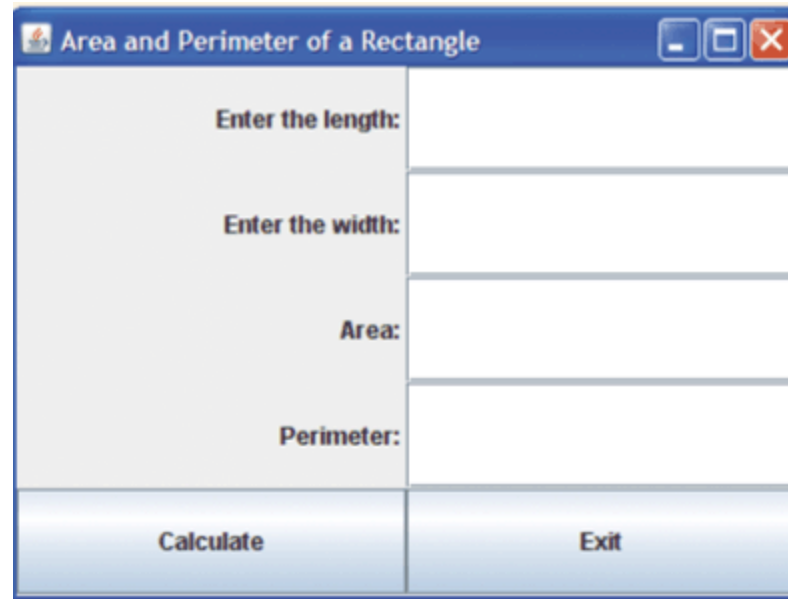
        frame.add(lengthL);
        frame.add(lengthTF);
        frame.add(widthL);
        frame.add(widthTF);
        frame.add(areaL);
        frame.add(areaTF);
        frame.add(perimeterL);
        frame.add(perimeterTF);
        frame.add(calculateB);
        frame.add(exitB);

        frame.setVisible(true);

    } //end main()
} //end class RectangleCalculator
```

So we have a pretty picture....

Big Deal – it's not like it *does* anything!

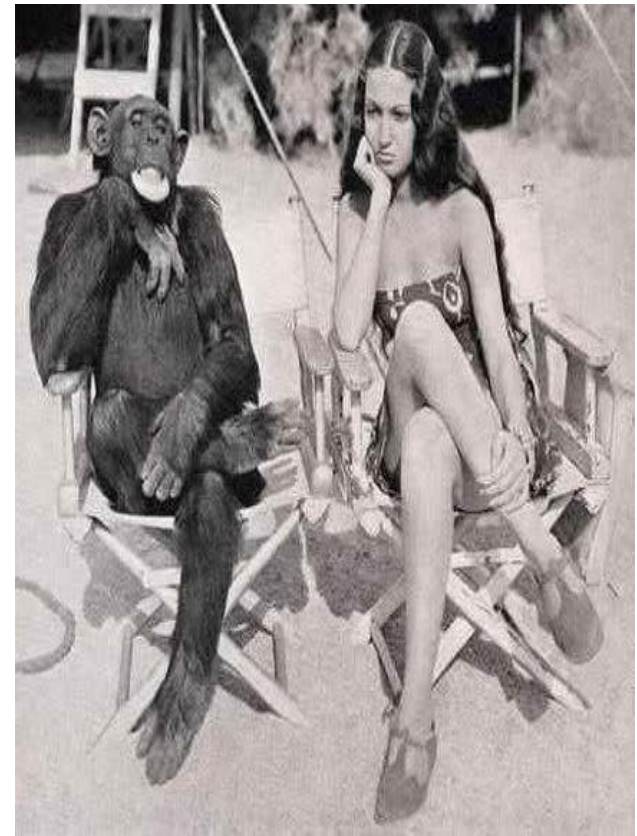


A Java Swing window titled "Area and Perimeter of a Rectangle" with a blue title bar and standard window controls (minimize, maximize, close). The window is divided into two main sections. The left section has a light gray background and contains four labels: "Enter the length:", "Enter the width:", "Area:", and "Perimeter:". The right section has a white background and contains four empty text input fields corresponding to the labels on the left. At the bottom of the window, there are two buttons: "Calculate" on the left and "Exit" on the right, both with a blue gradient background.

What Now???

The “Monkey See, Monkey Do” Teaching Model

- Sometimes (and *only* sometimes!) it’s easier to learn to code by NOT overthinking things too much. Instead, begin by simply studying, copying, and experimenting with code written by others.
- In this lecture (and in a couple of others), I am going to use certain Java techniques that we have not yet covered. For some of them you’ll simply have to “ape” what I do and trust that there is a good reason for it.
- However, as you practice with the code, and as we progress through the course, you’ll begin to get a sense of how and why these techniques were employed.
- For learning GUIs at this stage of the game, you should practice creating and modifying the GUIs we’ve covered here.
- Once you can pound out GUIs fairly easily (i.e. without having to ‘think’ about them a lot), you can then continue on and **do the same thing with events**.



Events

Events: Going back to our Rectangle program, when the calculate button is clicked, it should (hopefully) be clear that some kind of action must be triggered to perform the calculations and then output the results.

GUI components can generate all kinds of different events. For example, a button can generate an event by being clicked. A text field can generate an event when the ‘Enter’ key is pressed. More on this later.

We will spend the remainder of this lecture learning about the steps involved in writing the code that ‘handles’ the events.

**** Overview of steps in Event Handling ****

1. Create a class that contains the code to handle your events
 - This step involves creating a special kind of class, one that “implements an interface”. We will discuss interfaces in a later lecture.
2. Write code that connects the event you want to react to (e.g. the ‘Calculate’ button in the rectangle calculator) with the class from step #1.
 - This step is referred to as ‘registering the event handler’.

1. Creating the Handler Class

We begin by writing a particular kind of class called an ActionListener. This class has two noteworthy features:

1. It “implements an interface”. For now, all you need to know is that you must add the words ‘implements ActionListener’ after the class declaration.
2. It contains a method called ‘actionPerformed’. The code that responds to the event will be contained inside this method.

Here is a basic example of a handler class that could be connected to a button somewhere in our GUI:

```
private class CalculateButtonHandler implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {
        System.out.println("The button was clicked!");
    }
} //end class CalculateButtonHandler
```

When naming a handler class, we usually append the word ‘**Handler**’ at the end.

2. Registering the EventHandler:

GUI components such as buttons, text fields, and others, all have a method called ‘addActionListener’. This method takes one argument of type `ActionListener`. The class we created just previously definitely qualifies as an `ActionListener` class.

Suppose we had a button called `calculateB`. We will register the event handler class from step #1 with this button by invoking the ‘addActionListener’ method. The argument to this method will be an instance of our handler class:

```
calculateB.addActionListener( new HandlerClassExample() );
```

Again, the ‘addActionListener’ method is available to all GUI components:

```
emailTF.addActionListener( new AddToDBHandler() );  
exitB.addActionListener( new ExitHandler() );
```


Rinse, Lather, Repeat

I realize this may be a bit confusing right now, but reread the slides a few times, then look at some examples. Then come back to these descriptions. Repeat as needed and it will begin making more sense.

Where do we put the handler class?

The handler class can and often does exist as its own file. Even if you have multiple different handler classes, say, for multiple different buttons, you could have separate classes, each of which exists as a separate source code file.

However, our handler code frequently is tied to a specific GUI and will probably not be used for other separate GUIs. For this reason, we frequently put our handler class (or classes) inside the *same* class as our GUI. This is known as an “**nested class**” or, sometimes, as an “inner” class.

It is important that this class NOT be declared as static.

ActionListener is in `java.awt.event`

- The ActionListener class (note the argument to the actionPerformed method) is from the package java.awt.event. Therefore this must be imported.
 - This package is *not* imported by saying `import java.awt.*;`
 - Need: `import java.awt.event.*;`

One final step

- You will notice that there are a few changes that are admittedly confusing at this point. For example
 - I made all of the components class-level fields (they are not located inside any method).
 - I put most of the code for our program in a method that was NOT main. **One important point is that the method not be declared as static.** I called this method `go()` or sometimes `driver()` or similar.
 - To actually get the application to run, I created a main method, and instantiate the current class. I then invoked my `driver()` method.
- This may be slightly confusing for now, but it is necessary, and with practice you will quickly get used to it.

```

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

public class SimpleGui {
    JFrame frame;
    JButton button;

    public static void main(String[] args) {
        SimpleGui g = new SimpleGui();
        g.go();
    } //end method main()

    public void go() {
        frame = new JFrame("Practice GUI");
        frame.setSize(300, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        button = new JButton("Click Me");
        frame.add(BorderLayout.CENTER, button);

        button.addActionListener( new ButtonHandler() );

        frame.setVisible(true);
    } //end method go()

    private class ButtonHandler implements ActionListener {
        public void actionPerformed(ActionEvent ae) {
            System.out.println("The button was clicked!");
        } //end method actionPerformed()
    } //end ButtonHandler nested (inner) class

} //end class

```

A version of this code SimpleGui.java with several additional comments can be downloaded off the course web page, or directly by [clicking here](#).

STUDY THIS EXAMPLE!

Modify and experiment with it until you are comfortable with what is going on.

```

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

public class SimpleGui {
    JFrame frame;
    Container cp;
    JButton button;

    public static void main(String[] args) {
        SimpleGui g = new SimpleGui();
        g.go();
    } //end method main()

    public void go() {
        frame = new JFrame("Practice GUI");
        frame.setSize(300, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        button = new JButton("Click Me");
        frame.add(BorderLayout.CENTER, button);

        ButtonHandler b = new ButtonHandler();
        button.addActionListener( b );

        frame.setVisible(true);
    } //end method go()

    private class ButtonHandler implements ActionListener {
        public void actionPerformed(ActionEvent ae) {
            System.out.println("The button was clicked!");
        } //end method actionPerformed()
    } //end ButtonHandler nested (inner) class

} //end class

```

In this version, the ButtonHandler class is explicitly instantiated. The reference 'b' is then passed to the addActionListener method. However, doing so is probably not necessary since we don't anticipate any need to use the ButtonHandler reference in the future. We might as well simply pass the instantiation to the addActionListener method and not bother with declaring a reference as we did in the previous slide.

Another Example:

The class *TwoButtonWithHandlers.java* is an example of a GUI with two buttons and the handler class for each. Again, the handler classes are written inside the GUI class, so they are ‘nested’ classes.

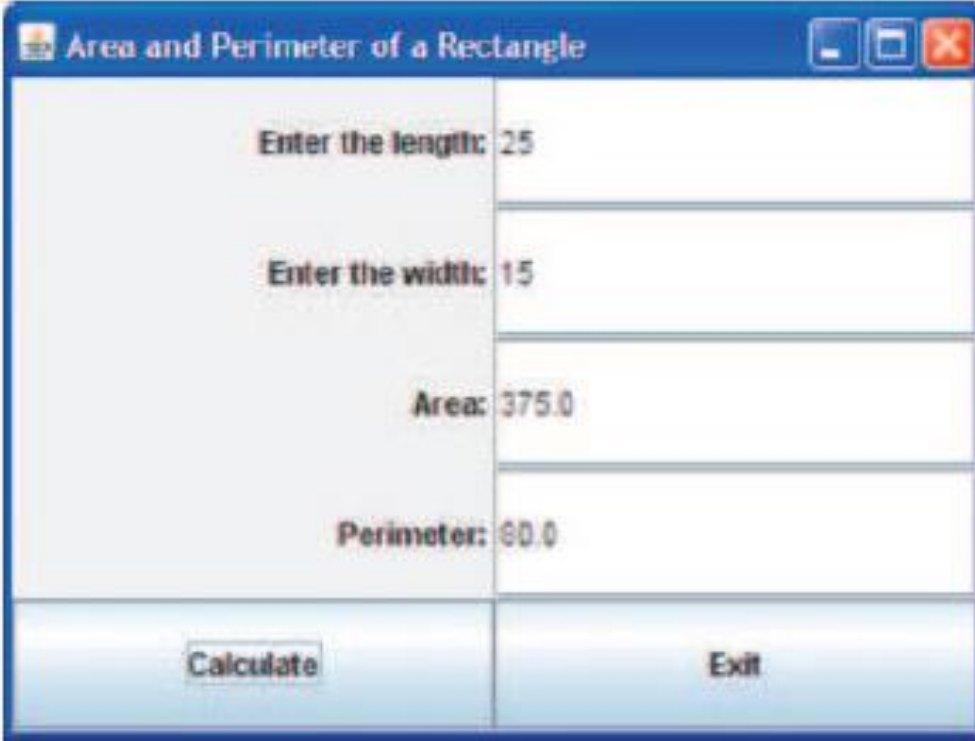
Using a separate class

- You can also place your ActionListener classes in separate files distinct from your GUI files.
- In other words, you do not have to create your handler classes as inner classes.
 - If you anticipate using the handler class in various different GUIs, then you will want to declare it in a separate, external.
 - If you only anticipate using the handler in the current GUI program, then create it as an inner class.
- You should know how to do both versions. As we have been doing throughout our Java work to date, maintain simplicity by keeping all of your files in the same folder.

Putting it all together

- My version of the code for this program can be found on the class webpage: [RectangleCalculator.java](#)
- As was discussed earlier, recall:
 - I made all of the fields class-level fields
 - I put most of the code for our program in a method that was NOT main. The key point is that it is not a static method. I called it `driver()`
 - To actually get the application to run, I created a main method, and instantiated the RectangleCalculator class. I then invoked my driver method.

Rectangle Program: Sample Run



Enter the length:	25
Enter the width:	15
Area:	375.0
Perimeter:	60.0
Calculate	Exit

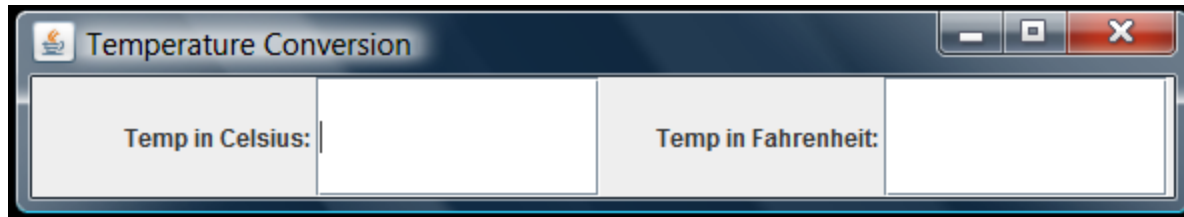
FIGURE 6-8 Sample run for the final RectangleProgram

Doing it on your own

1. Create your class
2. Make all fields of your GUI class-level fields (i.e. don't put them inside a method)
3. Put the bulk of your program's code inside a method (e.g. `driver()`) that is NOT main. This method must not be static.
 - I hope this is obvious, but as a reminder, any inner classes (e.g. a class to implement `ActionListener`) can not be inside a method.
4. Create a `main()` method and instantiate your class.
5. Use this object to invoke your driver method

Temperature Conversion

- See `TempConversion.java` on the class page.



- The requirements:
 - Draw the window shown here
 - When the user enters a number in either text field and then presses enter, the corresponding conversion should show up in the opposite text field.

TempConversion: What do we need?

The Display:

- A window
- A container (so we can access the content pane)
- 2 labels
- 2 text fields
- 2 handlers (one for the celcius field, one for the farenheit field)
- A layout manager (maybe not since the GUI has only one row. If no layout is specified, adding components simply places them one next to the other.)

The Event:

- When the user types something in a field and presses enter, the event should retrieve that value, do the conversion, then display the answer in the other text field.

Using the ActionEvent reference (optional)

- The ActionEvent reference that you see as a parameter in the actionPerformed method is instantiated by Java when an event is triggered. This object is then passed by Java to the actionPerformed method.
- This reference encapsulates all kinds of information about the event.
- For now, the piece of information you may find most useful comes from the method getSource(). This method tells you exactly which component generated the event.
- getSource() returns a reference to the object that generated the event. Example: Let's say a button called calculateB generated the event. In that case the statement

`e.getSource() == calculateB`

would return 'true'.

See TwoButtonsActionEvent.java on class page. Note how we only use one handler class, yet inside that class we can use the ActionEvent reference to determine which object generated the event.