GUI

Graphical User Interface

- GUI Introduction
- Creating Windows
- Equipping GUI classes with a main method
- Layout managers
- Radio buttons and Check Boxes
- Borders

GUIs - Introduction

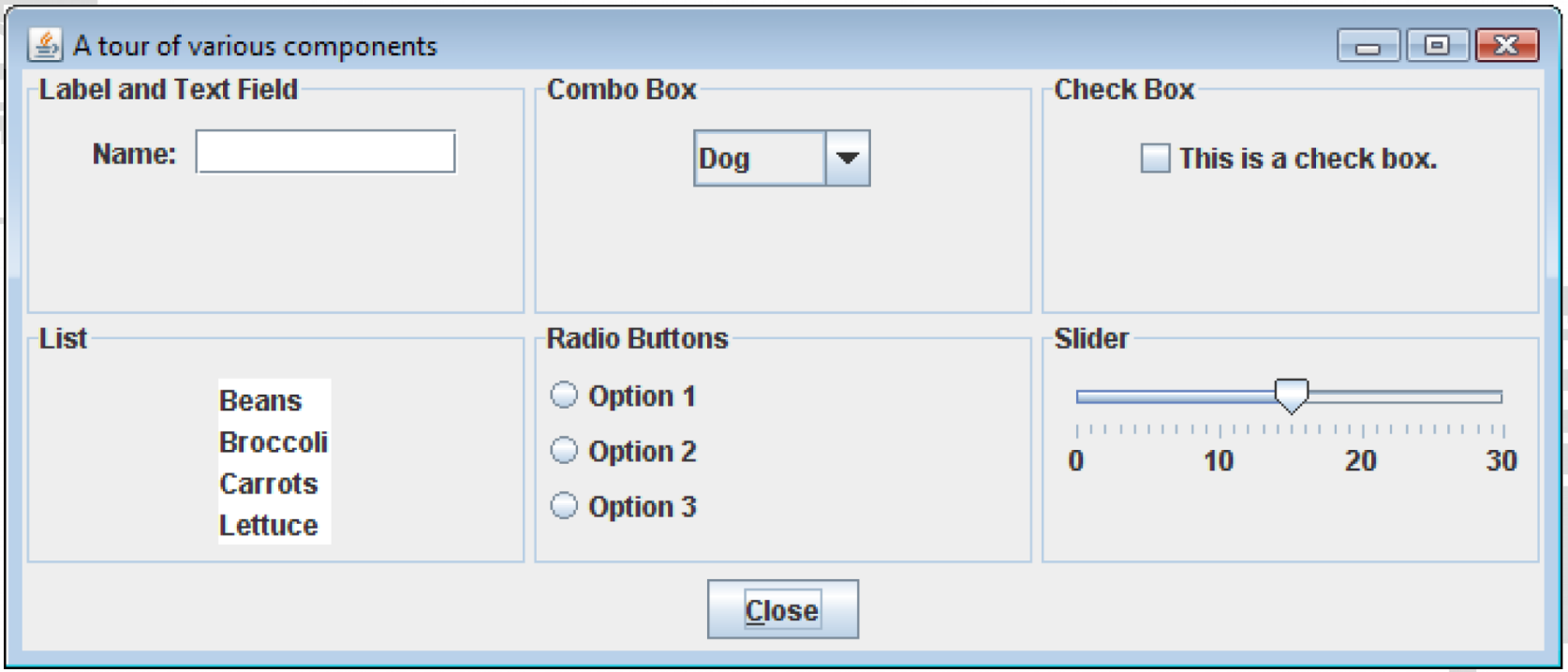
- Many Java application use a *graphical user interface* or *GUI* (pronounced "gooey").
- A GUI is a graphical window or windows that provide interaction with the user.
- GUI's accept input from:
  - the keyboard
  - a mouse.
- A window in a GUI consists of *components* that:
  - present data to the user
  - allow interaction with the application.

GUIs - Introduction

- A GUI is just a graphical front end for the program.
- The only difference between a GUI application and a command-line application is the way the user interacts with it.
- There is nothing special about a GUI application, it just looks different!

Introduction

- Some common GUI components are:
  - buttons, labels, text fields, check boxes, radio buttons, combo boxes, and sliders.



JFC, AWT, Swing

- Java programmers use the *Java Foundation Classes (JFC)* to create GUI applications.
- The JFC consists of several sets of classes, many of which are beyond the scope of this course
- The two sets of JFC classes that we focus on are AWT and Swing classes.
  - AWT and Swing are different ways of creating GUIs in Java
  - We will be focusing on Swing for this Module

AWT

- AWT stands fro Abstract Windowing Toolkit
- AWT uses the underlying Operating System to draw the GUI.
- This has the advantage of making the program look like other programs written for the operating system
- The downside is, not all operating systems support the same features or implement features in different ways
- Because of this, creating a cross-platform application in AWT is difficult.

Swing

- Swing doesn't rely on the operating system to draw most of its components
- Swing GUIs look consistent across all operating systems
  - The downside of this is that they often look out of place on all operating systems!
- Because the components from Swing are not reliant on the underlying operating system it is easy to create your own
- However, because Swing does not use the operating system's internal components, Swing applications can be slower than ones that use the native operating system components

Event-Driven programming

- Programs that operate in a GUI environment are "Event driven"
- An Event is an action that takes place in the program such as a the clicking of a button or press of a key
- Part of writing a GUI application is creating code that runs when an event is *triggered*
- This is known as an "Event Listener"
- An Event Listener is a piece of code that is called when the specified event is triggered

javax.swing and java.awt

- The classes for Swing components all exist in the package javax.swing
- To use them in your project you must import them, e.g.

```
import javax.swing.*;
```

  - Note the x in javax! It's not java.swing!

Swing

- As Swing is partly built on top of AWT, some AWT classes are required in Swing applications
- You can import AWT classes from java.awt e.g.

```
import java.awt.*;
```

- Note that there is no x after java in this package name!

Creating Windows

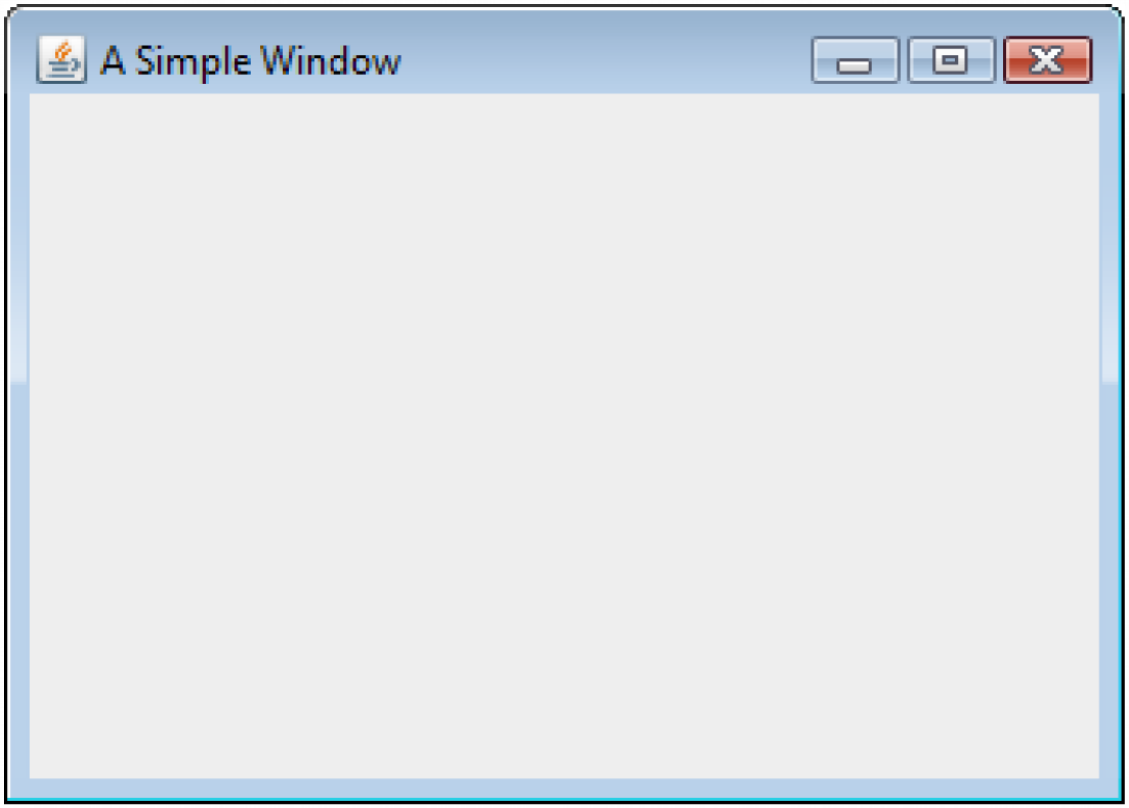
- GUI applications need at least one *window*
- A window is a *container*, which is simply a component that holds other componetns
- In Java a container that can be displayed as a window is a *frame*
- In a Swing application you create a frame from the JFrame class.

Windows

- A frame is a basic window that has:
  - A border around it
  - A title bar
  - A set of buttons for
    - Minimizing
    - Maximizing
    - Closing the windows
- These are standard features provided by the Operating System
- These are called *window decorations*

Jframe Window

- A basic window may look like this:



Creating windows

```
public static void main(String[] args) {

        JFrame window = new JFrame();

        window.setTitle("A Simple Window");

        window.setSize(350, 250);

        window.setVisible(true);

    }
```

Create the Frame

Set the window title

Set the height of the window (in pixels)

Set the width of the window (in pixels)

Make the window visible

Best practices

- Many old fashioned Java Tutorials tell you to create a class that *extends* JFrame like this:
-

```
public class Window2 extends JFrame {
    public Window2() {
        setTitle("Window Title");
        setVisible(true);
        setSize(350, 250);
    }

    public static void main(String[] args) {
        new Window2();
    }
}
```

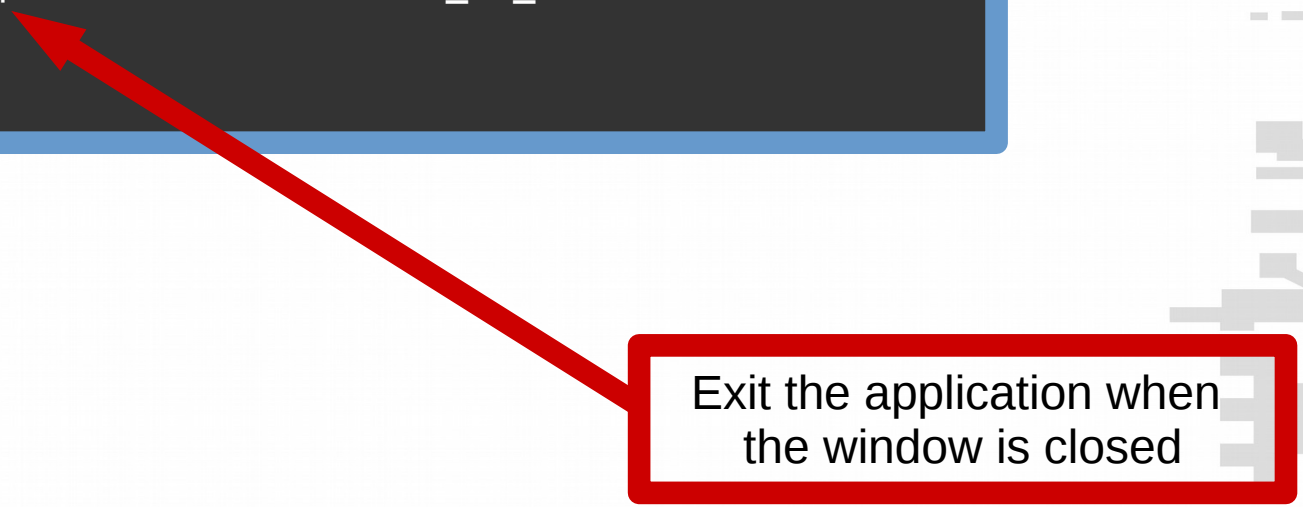
- This is considered *bad practice*. Instead you should create a new instance of the frame in the main method and add things to it.

Creating Windows

- By default, closing the window will just hide it
- Most of the time if you only have one window you want the program to exit when the window is closed.
- The Jframe provides a method called setDefaultCloseOperation
- You can use this to exit the application when the window is closed

Creating windows

```
JFrame window = new JFrame();

window.setTitle("A Simple Window");

window.setSize(350, 250);

window.setVisible(true);

window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```



Components

- Swing provides components for all the standard user interface features you see in programs:
  - Buttons
  - Menus
  - Text Fields
  - Labels

Labels

- A label is just a piece of text which is displayed in the program. It is not editable by the user and is used to tell the user to do something
- Swing provides a class **JLabel** for this purpose

Buttons

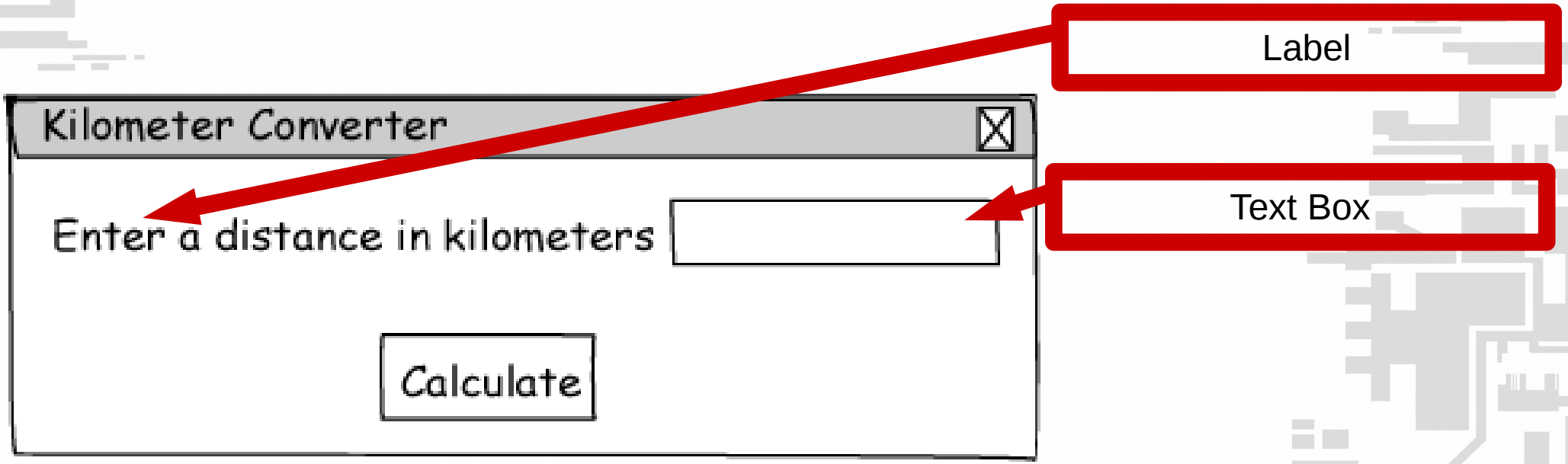
- Buttons are interactive components. They contain some text and allows the user to click on them
- When the button is clicked, some code is executed
- Swing provides the class **JButton** for this purpose.

Text Fields

- Text fields allow the user to enter a line of text
- The user can select the box by clicking it and then type into it
- Swing provides a **JTextField** class for this purpose.

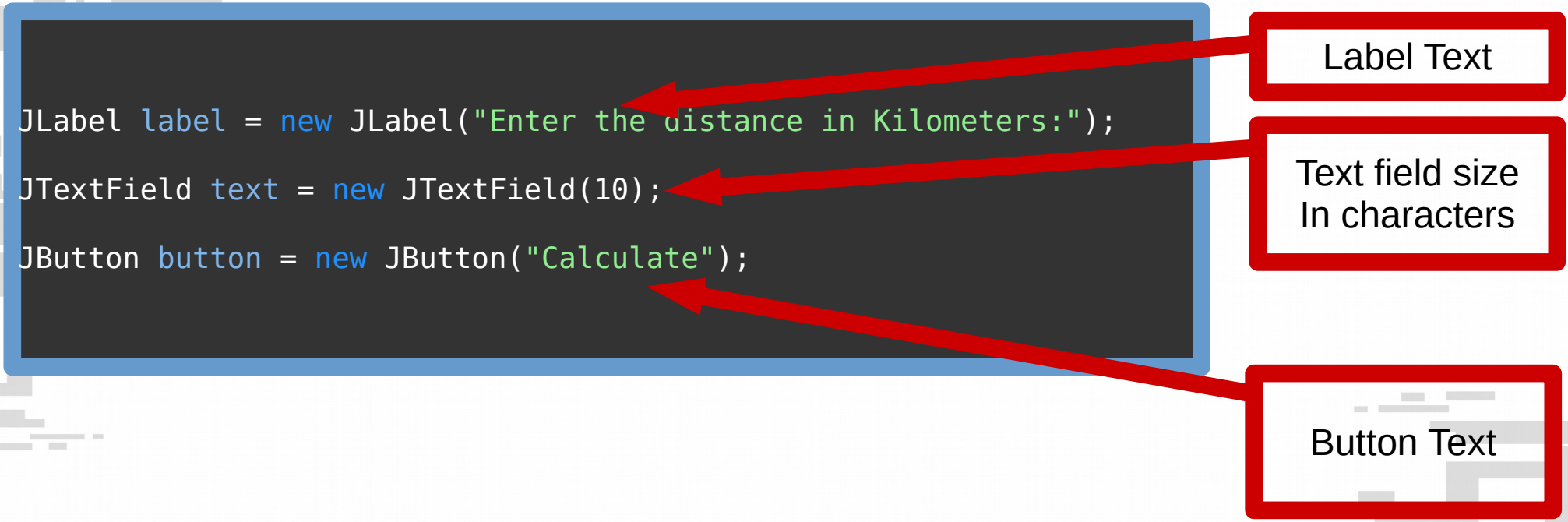
Example Program

- A button, Text box and label can be combined to make a simple program for building a Kilometre to Mile converter
- The program could look something like this



Adding components

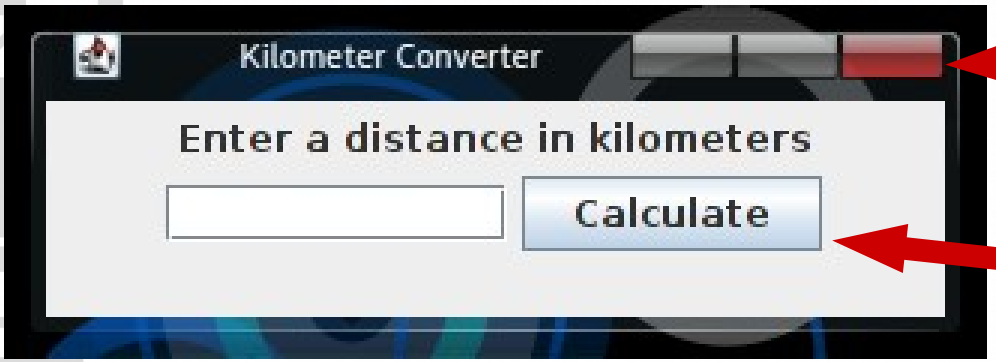
```
JLabel label = new JLabel("Enter the distance in Kilometers:");

JTextField text = new JTextField(10);

JButton button = new JButton("Calculate");
```



Adding components to the Window

- A Window is a container that can hold one other components. You cannot add all three components to a window
- Another component is a **Panel** a panel can hold any number of other components
- In most applications you will need to add a panel to the window
- Then add your components to the panel

```
JFrame window = new JFrame();

window.setTitle("Kilometer Converter");

window.setSize(350, 250);

window.setVisible(true);

window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


JLabel label = new JLabel("Enter the distance in kilometers:");

JTextField text = new JTextField(10);

JButton button = new JButton("Calculate");

JPanel panel = new JPanel();

panel.add(label);
panel.add(text);
panel.add(button);

window.add(panel);
```



Operating system
Window decorations

Swing colour scheme
and graphics that
do not match
the operating system!

Handling the events

- Now the GUI looks the way we intended, we need to make the calculate button do something when clicked
- An event is an action that takes place within a program such as clicking the button
- When an event takes place a specified piece of code is executed

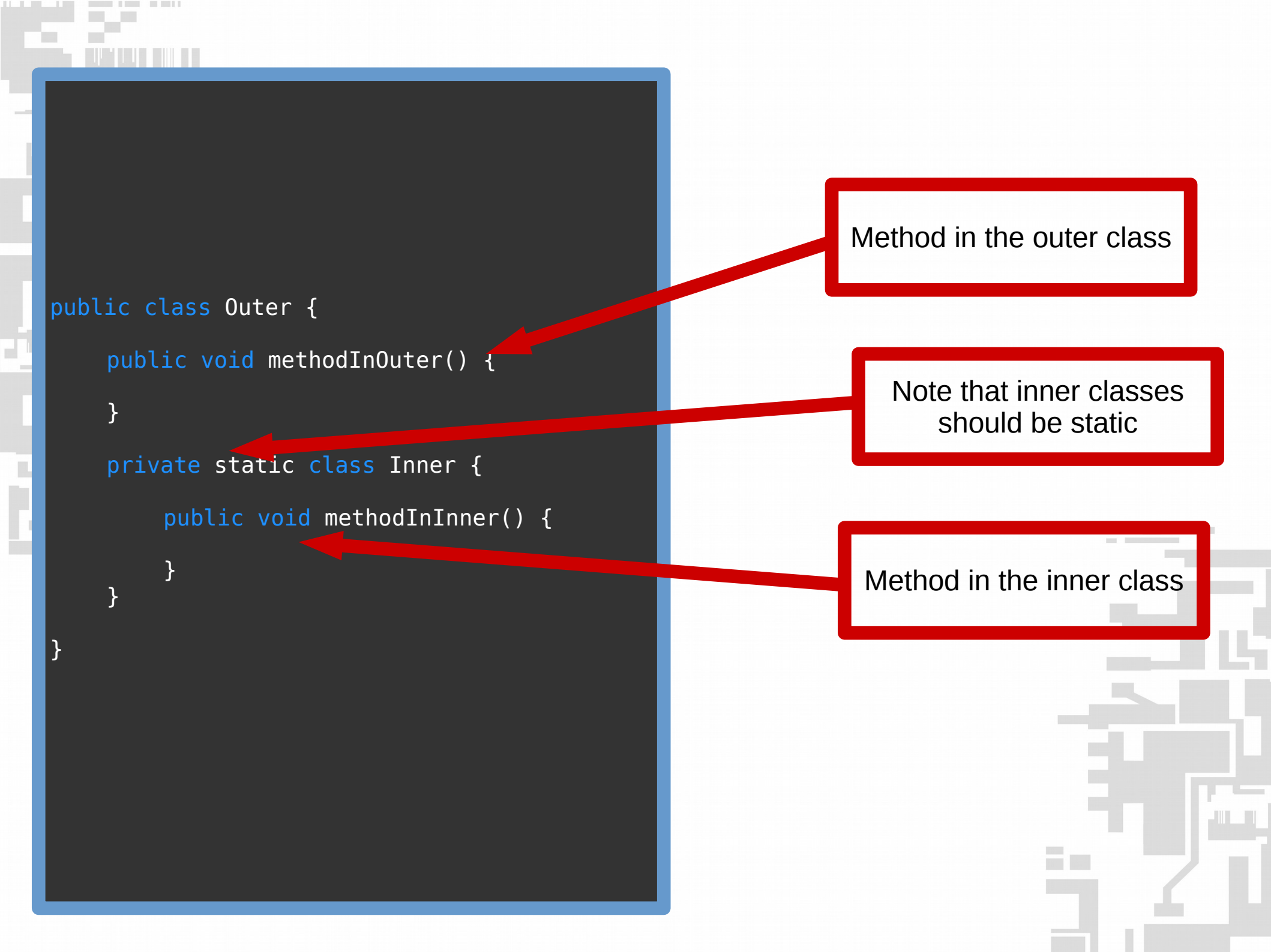
Handling Events

- An Event listener is an object that responds to events
- The component being interacted with *fires* an event which is then called in the event listener
- Event listeners are created by the programmer
- An event listener is a class

Inner Classes

- If you have to write a class in a new file for every action in the program you end up with a lot of classes in any non-trivial application!
- Java provides functionality for storing multiple classes in the same file. This is often used for events as it makes the code more manageable.
- These are known as "inner" and "outer" classes.
- An inner class exists inside the outer class.

```java
public class Outer {

    public void methodInOuter() {

    }

    private static class Inner {

        public void methodInInner() {

        }
    }

}
```



Event listener

- All event listener classes must implement a specified interface depending on the even type
- When you write a class that implements an interface you are agreeing that the class will have all of the methods that are specified in the interface

Currency Converter Button

- JButton components generate *action events* which require an *action listener class*
- An action listener class must:
  - Implement the interface **ActionListener**
  - Have a method named **ActionPerformed**
- The ActionPerformed method takes an argumen of the ActionEvent type

```
private class ButtonActionListener implements ActionListener {

    public void actionPerformed(ActionEvent arg0) {
        //Code to run when the button is clicked

    }
}
```

Handling Events

- The process of connecting an event listener object to a component is called *registering* the event listener.
- `JButton` components have a method named `addActionListener`.
- The addActionListener method takes one argument, an instance of the Action Listener class.

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

public class Example2 {

   private static class MyActionListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            System.out.println("Button clicked");
        }
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setSize(200,30);
        JButton button = new JButton();
        frame.add(button);
        frame.setVisible(true);

        button.setText("Press me");

        button.addActionListener(new MyActionListener());
    }
}
```

Import java.awt.event.*

Code to run when button Is clicked

Create the window and add a button

Assign the event

Output (in debug console) when button pressed:
“Button clicked”

Distance converter

- The distance converter can be set up in the same way.
- However, the distance converter needs to be able to read data from the text box
- This means the text box instance must be accessible to the ActionListener class
- To do this, you can define constructor arguments for the ActionListener:

```
public class Window {
        private static class ButtonActionListener implements ActionListener {

        public void actionPerformed(ActionEvent arg0) {
            //Code to run when the button is clicked
        }
    }

    public static void main(String[] args) {
        JFrame window = new JFrame();
        window.setTitle("Kilometre Converter");
        window.setSize(350, 250);
        window.setVisible(true);
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JLabel label = new JLabel("Enter the distance in Kilometers:");
        JTextField text = new JTextField(10);
        JButton button = new JButton("Calculate");

        button.addActionListener(new ButtonActionListener());

        JPanel panel = new JPanel();

        panel.add(label);
        panel.add(text);
        panel.add(button);
        window.add(panel);
    }

}
```

Here, there is no way for the actionListener to access the JTextField Instance (text)

```
public class Window {
    private static class ButtonActionListener implements ActionListener {
        private JTextField text;

        public ButtonActionListener(JTextField text) {
            this.text = text;
        }

        public void actionPerformed(ActionEvent arg0) {
            //Code to run when the button is clicked
        }
    }

    public static void main(String[] args) {
        JFrame window = new JFrame();
        window.setTitle("Kilometre Converter");
        window.setSize(350, 250);
        window.setVisible(true);
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


        JLabel label = new JLabel("Enter the distance in Kilometers:");
        JTextField text = new JTextField(10);
        JButton button = new JButton("Calculate");

        button.addActionListener(new ButtonActionListener(text));
        JPanel panel = new JPanel();

        panel.add(label);
        panel.add(text);
        panel.add(button);
        window.add(panel);
```

The text field can be Passed in as a Constructor argument

Distance converter demo

- Now that the ActionListener can access the text box, it's possible to read the text that is contained in the text box
- This can be done using the JTextField's **getText()** method

```
public class Window {
    private static class ButtonActionListener implements ActionListener {
        private JTextField text;

        public ButtonActionListener(JTextField text) {
            this.text = text;
        }

        public void actionPerformed(ActionEvent arg0) {
            System.out.println(text.getText());
        }
    }

    public static void main(String[] args) {
        JFrame window = new JFrame();
        window.setTitle("Kilometre Converter");
        window.setSize(350, 250);
        window.setVisible(true);
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


        JLabel label = new JLabel("Enter the distance in Kilometers:");
        JTextField text = new JTextField(10);
        JButton button = new JButton("Calculate");

        button.addActionListener(new ButtonActionListener(text));
        JPanel panel = new JPanel();

        panel.add(label);
        panel.add(text);
        panel.add(button);
        window.add(panel);
```

Read value from the text field when the button is clicked

```
public class Window {
    private static class ButtonActionListener implements ActionListener {
        private JTextField text;

        public ButtonActionListener(JTextField text) {
            this.text = text;
        }

        public void actionPerformed(ActionEvent arg0) {
            System.out.println(text.getText() + "km is " +
            Double.parseDouble(text.getText()) * 0.621 + " miles");
        }
    }

    public static void main(String[] args) {
        JFrame window = new JFrame();
        window.setTitle("Kilometre Converter");
        window.setSize(350, 250);
        window.setVisible(true);
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


        JLabel label = new JLabel("Enter the distance in Kilometers:");
        JTextField text = new JTextField(10);
        JButton button = new JButton("Calculate");

        button.addActionListener(new ButtonActionListener(text));
        JPanel panel = new Jpanel();

        window.add(panel);
        panel.add(label);
        panel.add(text);
        panel.add(button);
```

Do the KM to Miles Conversion

Layouts

- An important part of designing a GUI application is determining the layout of the components
- The term *layout* refers to the positioning and sizing of components.
- In Java, you do not normally specify the exact location of a component within a window.
- A *layout manager* is an object that:
  - controls the positions and sizes of components, and
  - makes adjustments when necessary.

Layout Managers

- Panels and Frames use Layout Managers to define how components are displayed
- Swing provides several Layout Managers
  - **FlowLayout** – Arranges components in Rows. This is the default for panels
  - **BorderLayout** – Arranges components in five regions:
    - North, south, east, west and Center
    - This is the default for the JFRame
  - **GridLayout** – Arranges components in a table with rows and columns

Layout Managers

- Frames and Panels have a **setLayout()** method
- This takes an instance of a layout object such as:

```
JPanel panel = new JPanel();

panel.setLayout(new BorderLayout());
panel.setLayout(new FlowLayout());
```

FlowLayout

- `FlowLayout` is the default layout manager for `JPanel` objects.
- Components appear horizontally, from left to right, in the order that they were added. When there is no more room in a row, the next components "flow" to the next row.

FlowLayout

- As the window is resized and the buttons no longer fit, the buttons wrap onto the next line

```
JFrame window = new JFrame();
// Set the title bar text.
window.setTitle("Flow Layout");

// Set the size of the window.
window.setSize(200, 200);

// Specify an action for the close button.
window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

// Add a FlowLayout manager to the content pane.
window.setLayout(new FlowLayout());

// Create three buttons.
JButton button1 = new JButton("Button 1");
JButton button2 = new JButton("Button 2");
JButton button3 = new JButton("Button 3");

// Add the three buttons to the content pane.
window.add(button1);
window.add(button2);
window.add(button3);

  // Display the window.
window.setVisible(true);
```

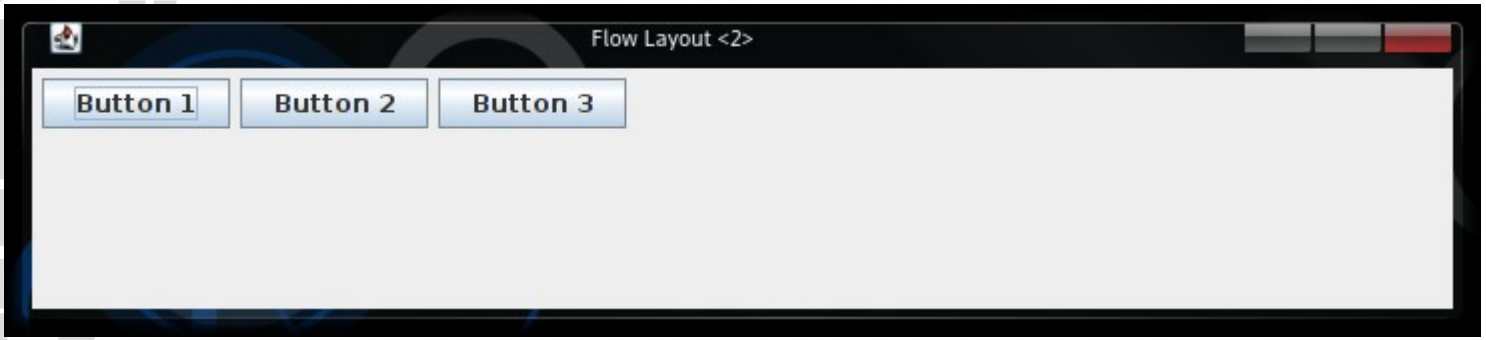
FlowLayout Manager

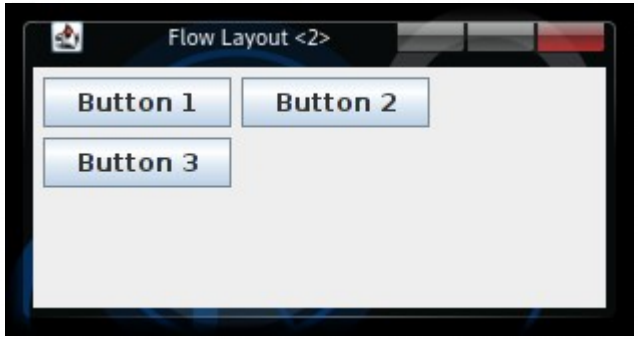
- The `FlowLayout` manager allows you to align components:
  - in the center of each row
  - along the left or right edges of each row .
- FlowLayout provides a constructor that takes one of the following options:
  - `FlowLayout.CENTER,`
  - `FlowLayout.LEFT, or`
  - `FlowLayout.RIGHT.`
- Example:

```
setLayout(new FlowLayout(FlowLayout.LEFT));
```

FlowLayout (Left)

```
setLayout(new FlowLayout(FlowLayout.LEFT));
```



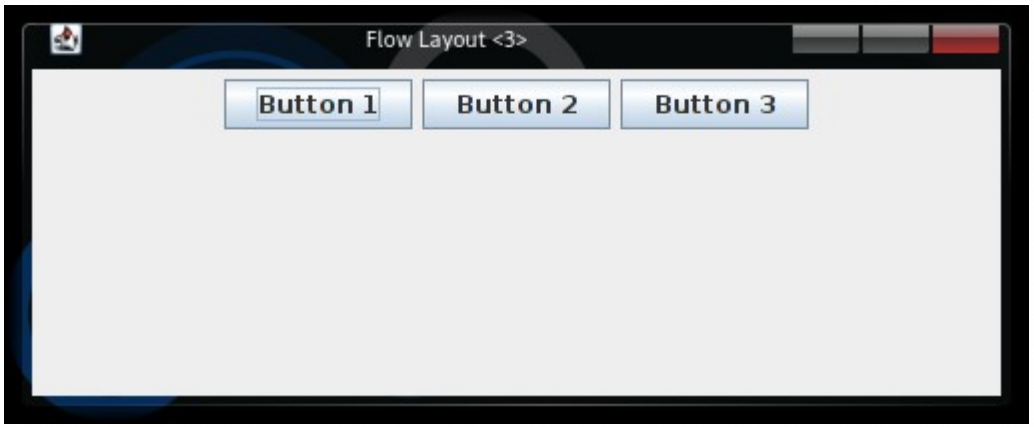


FlowLayout (Right)

```
setLayout(new FlowLayout(FlowLayout.RIGHT));
```

FlowLayout (Center)

```
setLayout(new FlowLayout(FlowLayout.CENTER));
```



FlowLayout Spacing

- `FlowLayout` inserts a gap of five pixels between components, horizontally and vertically.
- The `FlowLayout` constructor allows these to be adjusted.
- The constructor has the following format:

```
FlowLayout(int alignment,
           int horizontalGap,
           int verticalGap)
```

- Example:

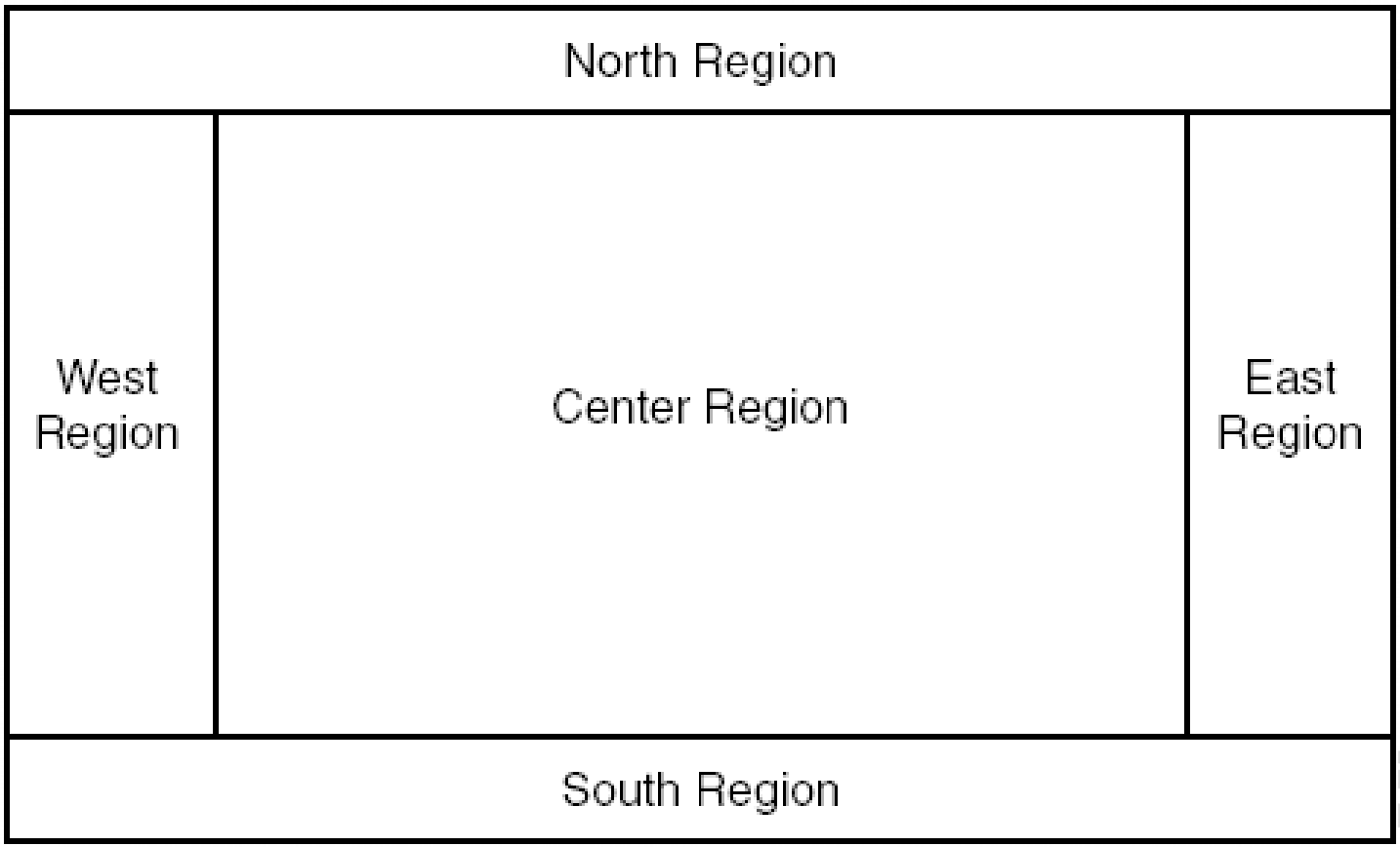
```
setLayout(new FlowLayout(FlowLayout.LEFT, 10, 7));
```

FlowLayout spacing

```
setLayout(new FlowLayout(FlowLayout.LEFT, 50, 50));
```

BorderLayout Manager

- BorderLayout manages five regions where components can be placed



BorderLayout

- A component placed into a container that is managed by a `BorderLayout` must be placed into one of five regions:
  - `BorderLayout.NORTH`
  - `BorderLayout.SOUTH`
  - `BorderLayout.EAST`
  - `BorderLayout.WEST`
  - `BorderLayout.CENTER`

BorderLayout

- Each region can hold only one component at a time.
- When a component is added to a region, it is stretched so it fills up the entire region.
- `BorderLayout` is the default manager for `JFrame` objects.

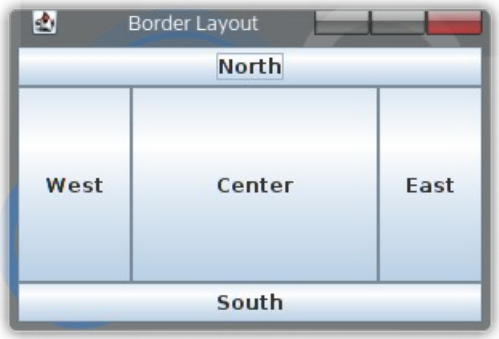
```
panel.add(button, BorderLayout.NORTH);
```

BorderLayout

- Normally the size of a button is just large enough to accommodate the text that it displays
- The buttons displayed in `BorderLayout` region will not retain their normal size.
- The components are stretched to fill all of the space in their regions.

```
JFrame window = new JFrame();
// Set the title bar text.
window.setTitle("Border Layout");

// Set the size of the window.
window.setSize(300, 200);

// Specify an action for the close button.
window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

// Add a BorderLayout manager to the content pane.
window.setLayout(new BorderLayout());

// Create three buttons.
JButton button1 = new JButton("North");
JButton button2 = new JButton("South");
JButton button3 = new JButton("East");
JButton button4 = new JButton("West");
JButton button5 = new JButton("Center");

// Add the three buttons to the content pane.
window.add(button1, BorderLayout.NORTH);
window.add(button2, BorderLayout.SOUTH);
window.add(button3, BorderLayout.EAST);
window.add(button4, BorderLayout.WEST);
window.add(button5, BorderLayout.CENTER);

  // Display the window.
window.setVisible(true);
```

- Border Layout resizes the buttons to fill the container:



BorderLayout

- If the user resizes the window, the sizes of the components will be changed as well.
- `BorderLayout` manager resizes components:
  - placed in the north or south regions may be resized horizontally so it fills up the entire region,
  - placed in the east or west regions may be resized vertically so it fills up the entire region.
  - A component that is placed in the center region may be resized both horizontally and vertically so it fills up the entire region.

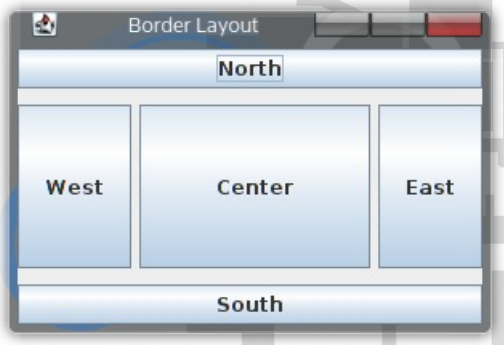
BorderLayout

- By default there is no gap between the regions.
- An overloaded `BorderLayout` constructor allows horizontal and vertical gaps to be specified (in pixels).
- The constructor has the following format

```
BorderLayout(int horizontalGap, int verticalGap)
```

- Example:

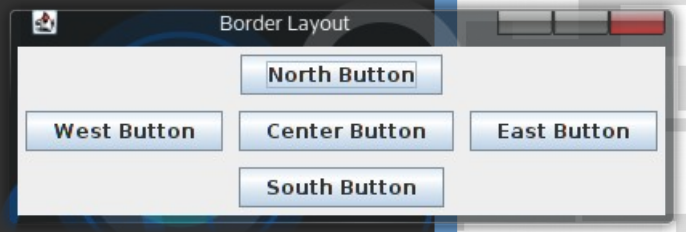
```
setLayout(new BorderLayout(5,10));
```



Nesting Components in a layout

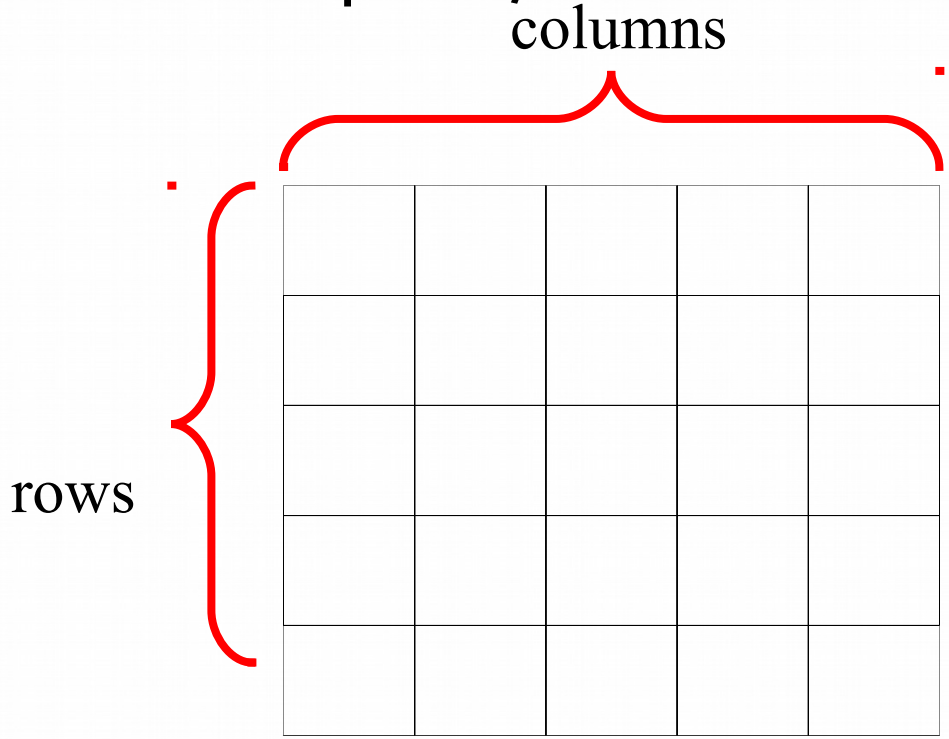
- Adding components to panels and then nesting the panels inside the regions can overcome the single component limitation of layout regions.
- By adding buttons to a `JPanel` and then adding the `JPanel` object to a region, sophisticated layouts can be achieved.
- This stops the stretching of the components and keeps them looking nicer

```
JFrame window = new JFrame();
window.setTitle("Border Layout");
window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

  // Add a BorderLayout manager to the content pane.
window.setLayout(new BorderLayout());
// Create five panels.
JPanel panel1 = new JPanel();
JPanel panel2 = new JPanel();
JPanel panel3 = new JPanel();
JPanel panel4 = new JPanel();
JPanel panel5 = new JPanel();
// Create five buttons.
JButton button1 = new JButton("North Button");
JButton button2 = new JButton("South Button");
JButton button3 = new JButton("East Button");
JButton button4 = new JButton("West Button");
JButton button5 = new JButton("Center Button");
// Add the buttons to the panels.
panel1.add(button1);
panel2.add(button2);
panel3.add(button3);
panel4.add(button4);
panel5.add(button5);
  // Add the five panels to the content pane.
window.add(panel1, BorderLayout.NORTH);
window.add(panel2, BorderLayout.SOUTH);
window.add(panel3, BorderLayout.EAST);
window.add(panel4, BorderLayout.WEST);
window.add(panel5, BorderLayout.CENTER);

  // Pack and display the window.
window.setVisible(true);
```



GridLayout

- GridLayout creates a grid with rows and columns like a spreadsheet
- A container that is managed by GridLayout is divided into equally sized cells



GridLayout

- The GridLayout constructor takes two integer arguments:
  - Number of Rows
  - Number of Columns
- Example:

```
setLayout(new GridLayout(2, 3));
```

```
JFrame window = new JFrame();
// Set the title bar text.
window.setTitle("Grid Layout");

// Set the size of the window.
window.setSize(WINDOW_WIDTH, WINDOW_HEIGHT);

// Specify an action for the close button.
window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

// Add a GridLayout manager to the content pane.
window.setLayout(new GridLayout(2, 3));

// Create six buttons.
JButton button1 = new JButton("Button 1");
JButton button2 = new JButton("Button 2");
JButton button3 = new JButton("Button 3");
JButton button4 = new JButton("Button 4");
JButton button5 = new JButton("Button 5");
JButton button6 = new JButton("Button 6");

// Add the six buttons to the content pane.
window.add(button1);  // Goes into row 1, column 1
window.add(button2);  // Goes into row 1, column 2
window.add(button3);  // Goes into row 1, column 3
window.add(button4);  // Goes into row 2, column 1
window.add(button5);  // Goes into row 2, column 2
window.add(button6);  // Goes into row 2, column 3

// Display the window.
window.setVisible(true);
```