Programming Languages -III

Graphical User Interface – Java Swing

Event-Driven Programming

- *Procedural programming* is executed in procedural order.
- In *event-driven programming*, code is executed upon *activation of events*.

Graphical User Interface in Java

- Programming in GUI is normally Event driven.
- Event: A type of signal to the program that something has happened.
- Gui depends on:
  - Components: an object having a graphical representation. Examples are Frame, Button etc.
  - Event Listeners: responds to an event.
    - The code that is executed once an event occurs.

Components

- A component is an object having a graphical representation
- Components can be displayed on the screen
- Swing provides many standard GUI **components** such as:
  - Buttons
  - Lists
  - Menus
  - text areas
- Components can be combined to create your program's GUI.
- Swing provides **containers(which are components that can include other components)** such as windows and tool bars.

Components: Abstract Window Toolkit (AWT) vs. Swing

AWT

- Used before Swing was introduced.
- All components are heavyweight because they are tied to the local platform’s windowing system.
- The *look-and-feel* of the components is uniform on all platforms.

Swing

- Introduced after the AWT.
- Some components are lightweight, however, some components like AWT are heavyweight because they are tied to the underlying platform’s windowing system.
- The *look-and-feel* of the components is uniform on all platforms.

Overview of Swing Components

- JLabel – Displays un-editable text or icons.
- JTextField – Enables user to enter data from the keyboard. Can also display editable/un-editable text.
- JButton – Used to perform an action.
- JCheckBox – Specifies an option that can be selected or not selected .
- JComboBox – Provides a drop-down list of items from which the user can make a selection by clicking an item or possibly by typing into the box.
- JList – Provides a list of items from which the user can make a selection by clicking on any item in the list. Multiple elements can be selected.
- JPanel – Provides an area in which components can be placed and organized. Can also be used as a drawing area for graphics.

Containers

- Components that can contain other components.
- Components are added to a container using one of the various forms of its **add** method

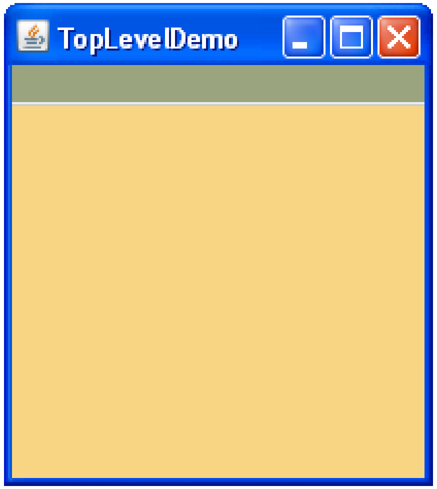
```
    panel.add(component);
```

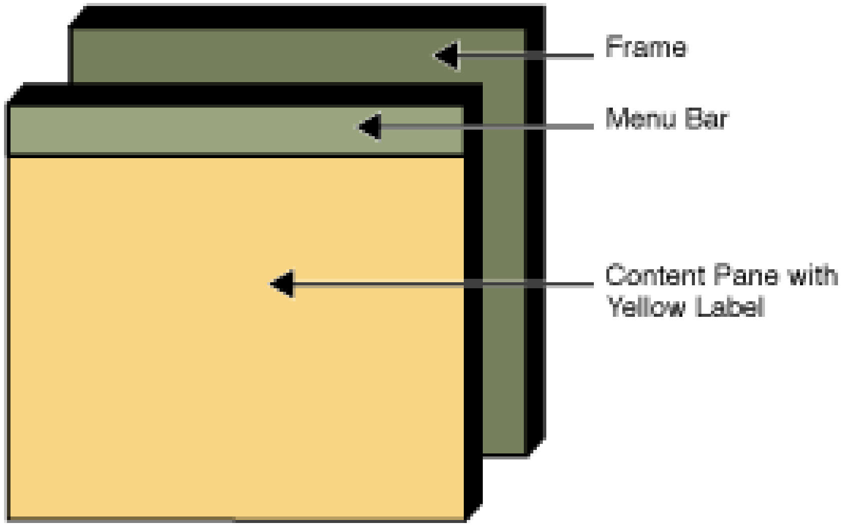
- Components can be positioned manually, but a large number of Components would be difficult to manage.
- A layout manager helps with the placement of components in a container and size of components.

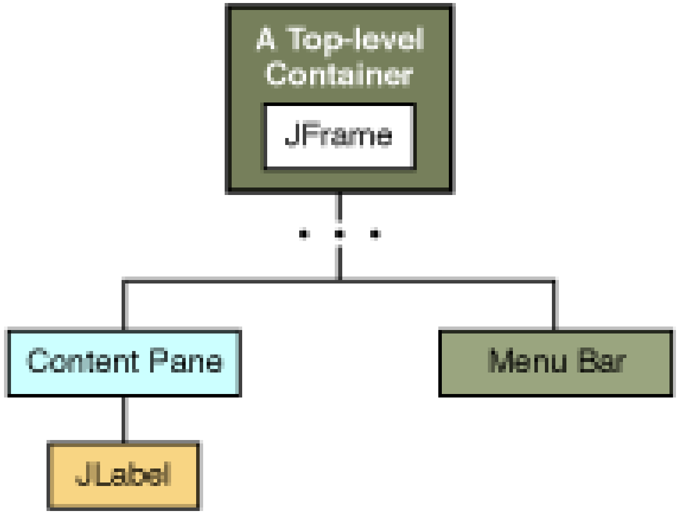
Top Level Containers

- Every program that presents a Swing GUI contains at least one top-level container.
- A Top level container provides the support that Swing components need to perform their painting and event-handling.
- Each top-level container has a content pane that, generally speaking, contains (directly or indirectly) the visible components in that top-level container's GUI
- Swing provides the following top-level containers:
  - JFrame (Main window)
  - JDialog (Secondary window)
  - JApplet (An applet display area within a browser window)

Top Level Container

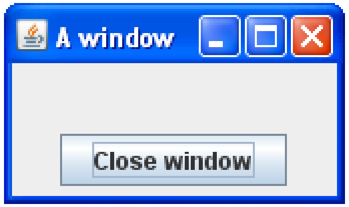






JFrame

- **`javax.swing.JFrame`**: JFrame is part of Java swing.
- JFrame is an indirect subclass of class java.awt.Window that provides the basic attributes and behaviours of the window.
- Top-level window with a title and a border.
- Usually used as a program's main window.
- Visible Components are added to the Content Pane layer.
  - Use `getContentPane()` to obtain it



JFrame

```
import javax.swing.*;
public class MainClass {
public static void main(String[] args) {
JFrame f1 = new JFrame ();
f1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
f1.pack();
f1.setVisible(true);
}
}
```

Jframe with Buttons

```
import javax.swing.*;
public class MainClass {
public static void main(String[] args) {
JFrame f1 = new JFrame ();
f1. getContentPane().add(new JButton(“B1”));
f1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
f1.pack();
f1.setVisible(true);
}
}
```

Jframe with Buttons Alternative Approach

```
import javax.swing.*;
public class MainClass extends JFrame {
public MainClass(){
getContentPane().add(new JButton(“B1”));
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
pack();
}
public static void main(String[] args) {
MainClass f1 = new MainClass ();
f1.setVisible(true);
}
}
```

JLabel

- Displays un-editable text or icons.

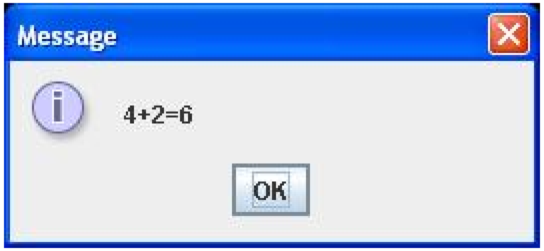
```
import javax.swing.*;
public class testLabel1 {
            public static void main(String[] args) {
            JFrame f1 = new JFrame ();
            f1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

   // creating a label and adding it to the frame (container).
            JLabel l1 = new JLabel("Hello World");
            f1.getContentPane().add(l1);

            f1.pack();
            f1.setVisible(true);
            }
}
```

Example 2: Frame with a Label

```
import javax.swing.*;

public class HelloWorldFrame extends JFrame {
  public HelloWorldFrame() {
    super(“HelloWorldSwing”);
    final JLabel label = new JLabel("Hello World");
    getContentPane().add(label);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    pack();
    setVisible(true);
  }
  public static void main(String[] args) {
    HelloWorldFrame frame = new HelloWorldFrame();
  }
}
```

JDialog

- **`javax.swing.JDialog`**:
- More simple and limited than frames
- Typically used for showing a short message on the screen
- Also has a border and a title bar
- May have an owner
  - If the owner is invisible the dialog will also be invisible



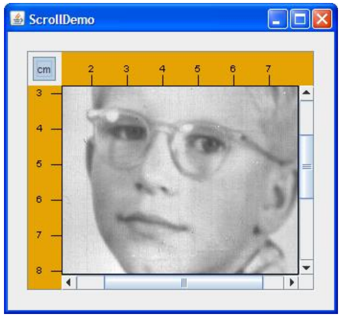
JOptionPane for JDialog

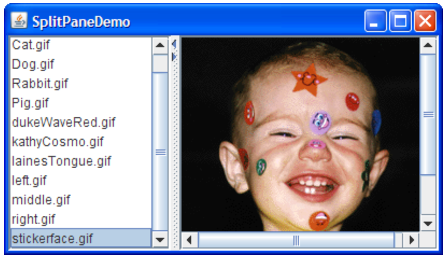
- Dialog boxes are normally used to interact with the user.
- Provides pre-built dialog boxes.
- Dialogs are displayed using static JOptionPane methods.

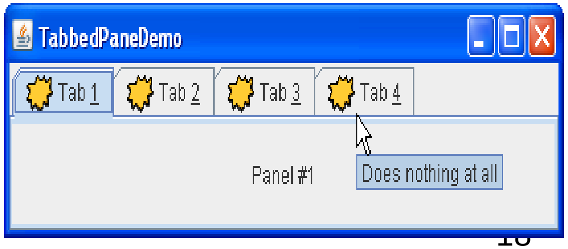
```
import javax.swing.JOptionPane;
public class testJOptionPane {
public static void main(String[] args) {
// TODO Auto-generated method stub
// Obtain first user input from JOptionPane input dialogs
String firstNumber = JOptionPane.showInputDialog("Enter First Integer");
// Obtain second user input from JOptionPane input dialogs
String secondNumber = JOptionPane.showInputDialog("Enter Second Integer");
// Convert string inputs to int values for use in a calculation
int number1 = Integer.parseInt(firstNumber);
int number2 = Integer.parseInt(secondNumber);
int sum = number1 + number2;
//display result in JOptionPane message dialog
JOptionPane.showMessageDialog(null, "The sum is " + sum);
} // end main method
}  // end testJOptionPane
```

Internal Containers

- Not Top level containers
- Can contain other non-top level components
- Examples:
  - `JScrollPane`: Provides a scrollable view of its components
  - `JSplitPane`: Separates two components
  - `JTabbedPane`: User chooses which component to see







Containers - Layout

- Each container has a layout manager
  - Determines the size, location of contained components.

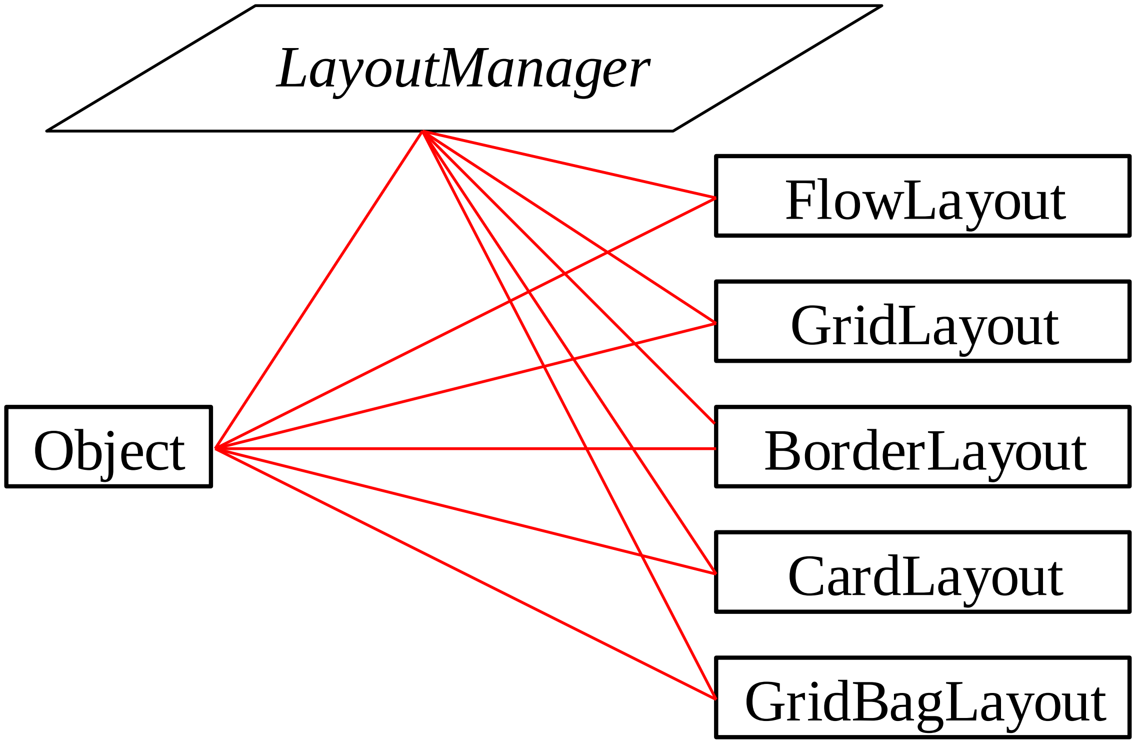
- Setting the current layout of a container:
  void setLayout(LayoutManager lm)

- *LayoutManager* implementing classes:
  - BorderLayout
  - BoxLayout
  - FlowLayout
  - GridLayout

Layout Managers

- Control the placement of components on the container.
- This is an alternative to hardcoding the pixel locations of the components.
- Advantage: resizing the container (frame) will not occlude or distort the view of the components.
- Main layout managers:
  - FlowLayout, GridLayout, BorderLayout, CardLayout, and GridBagLayout

Layout Manager Hierarchy



LayoutManager is an **interface**. All the layout classes **implement** this interface

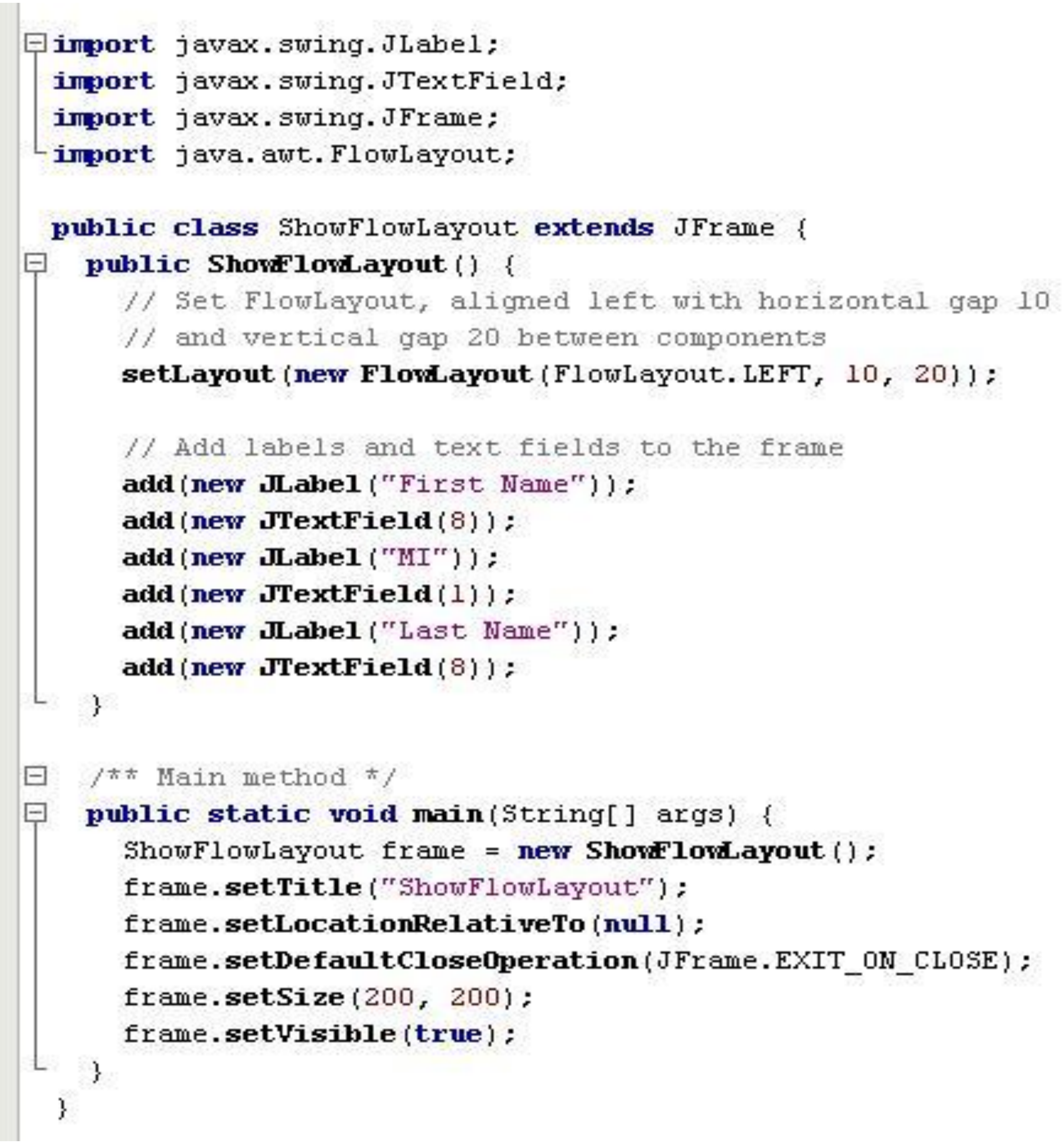
FlowLayout

- Places components sequentially (left-to-right) in the order they were added
- Components will wrap around if the width of the container is not wide enough to hold them all in a row.
- Default for applets and panels, but not for frames
- Options:
  - left, center (this is the default), or right
- Typical syntax: in your Frame class's constructor

  setLayout(new FlowLayout(FlowLayout.LEFT)) OR

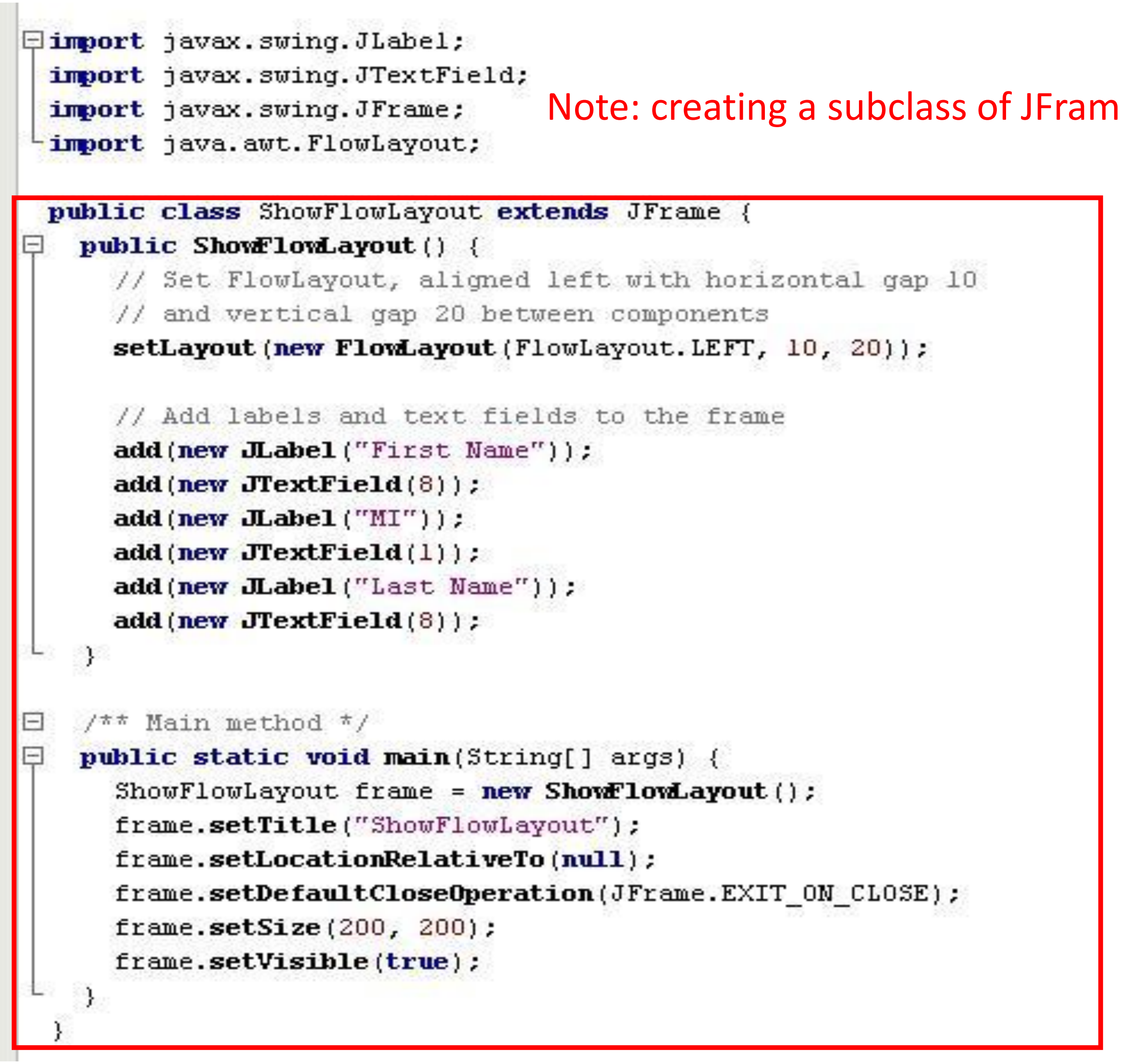
  setLayout(new FlowLayout(FlowLayout.LEFT,hgap,vgap))

A Frame class that uses FlowLayout layout manager



```
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JFrame;
import java.awt.FlowLayout;

public class ShowFlowLayout extends JFrame {
  public ShowFlowLayout() {
    // Set FlowLayout, aligned left with horizontal gap 10
    // and vertical gap 20 between components
    setLayout(new FlowLayout(FlowLayout.LEFT, 10, 20));

    // Add labels and text fields to the frame
    add(new JLabel("First Name"));
    add(new JTextField(8));
    add(new JLabel("MI"));
    add(new JTextField(1));
    add(new JLabel("Last Name"));
    add(new JTextField(8));
  }

  /** Main method */
  public static void main(String[] args) {
    ShowFlowLayout frame = new ShowFlowLayout();
    frame.setTitle("ShowFlowLayout");
    frame.setLocationRelativeTo(null);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(200, 200);
    frame.setVisible(true);
  }
}
```

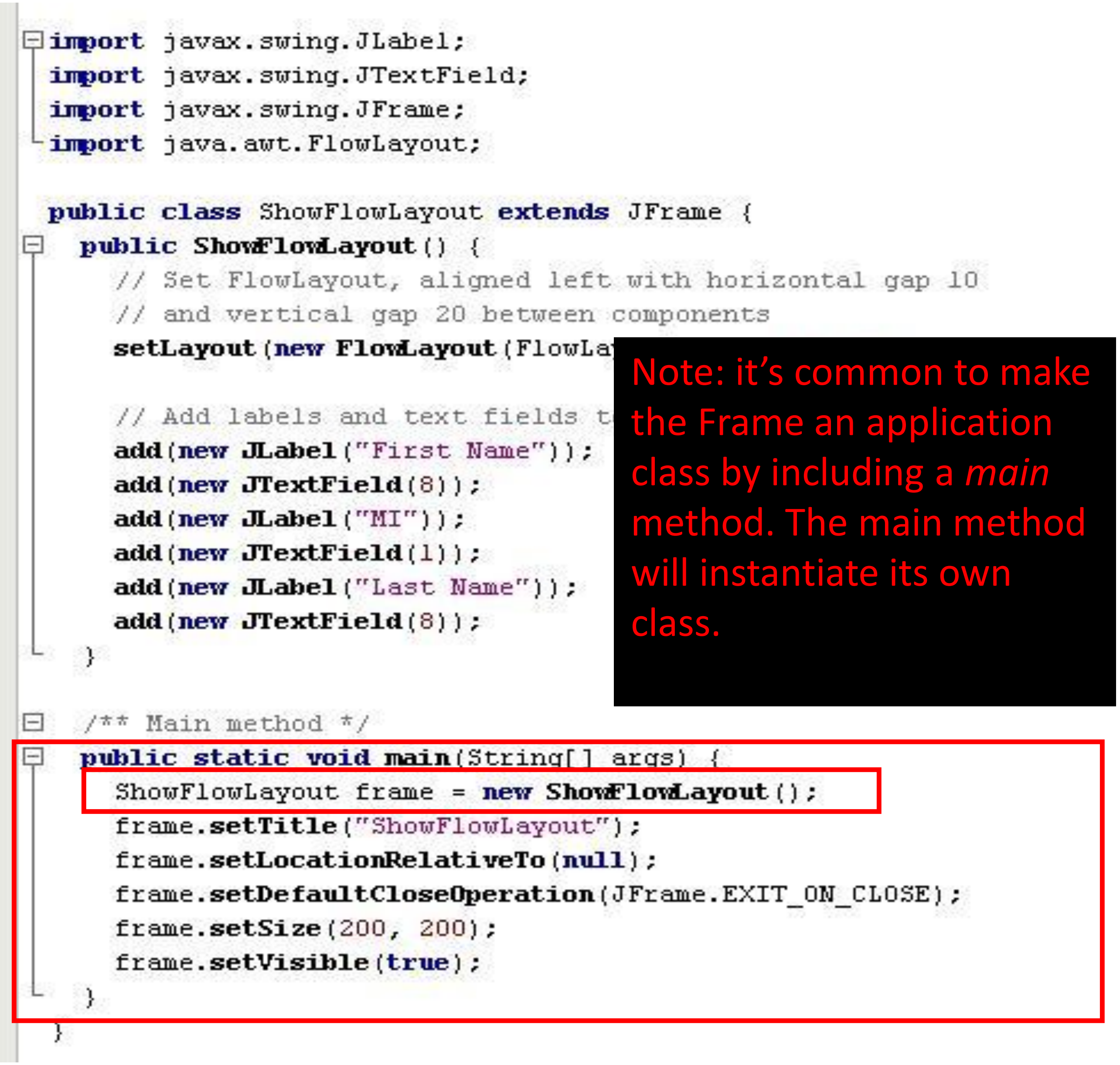
A Frame class that uses FlowLayout layout manager



Note: creating a subclass of JFrame

```
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JFrame;
import java.awt.FlowLayout;

public class ShowFlowLayout extends JFrame {
  public ShowFlowLayout() {
    // Set FlowLayout, aligned left with horizontal gap 10
    // and vertical gap 20 between components
    setLayout(new FlowLayout(FlowLayout.LEFT, 10, 20));

    // Add labels and text fields to the frame
    add(new JLabel("First Name"));
    add(new JTextField(8));
    add(new JLabel("MI"));
    add(new JTextField(1));
    add(new JLabel("Last Name"));
    add(new JTextField(8));
  }

  /** Main method */
  public static void main(String[] args) {
    ShowFlowLayout frame = new ShowFlowLayout();
    frame.setTitle("ShowFlowLayout");
    frame.setLocationRelativeTo(null);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(200, 200);
    frame.setVisible(true);
  }
}
```

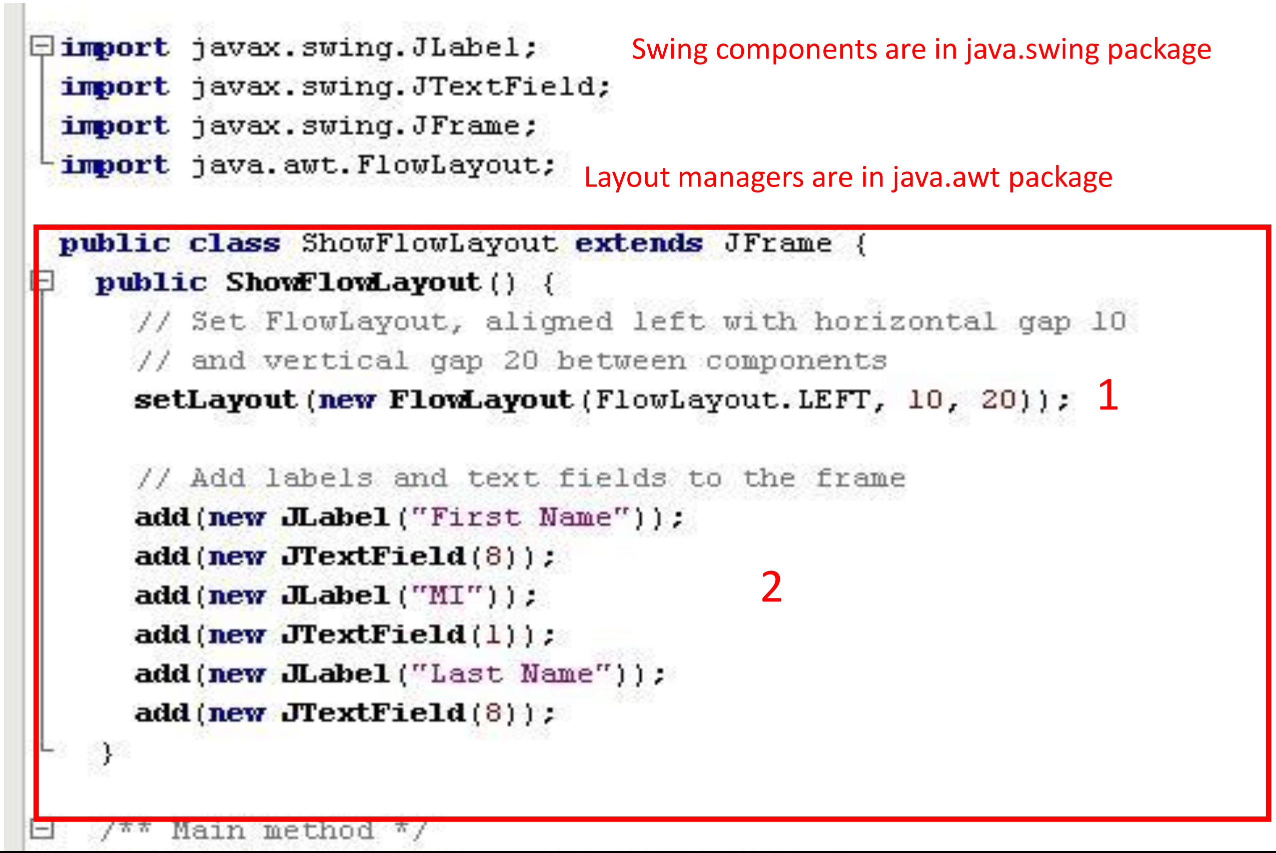
A Frame class that uses FlowLayout layout manager



```
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JFrame;
import java.awt.FlowLayout;

public class ShowFlowLayout extends JFrame {
  public ShowFlowLayout() {
    // Set FlowLayout, aligned left with horizontal gap 10
    // and vertical gap 20 between components
    setLayout(new FlowLayout(FlowLa

    // Add labels and text fields t
    add(new JLabel("First Name"));
    add(new JTextField(8));
    add(new JLabel("MI"));
    add(new JTextField(1));
    add(new JLabel("Last Name"));
    add(new JTextField(8));
  }

  /** Main method */
  public static void main(String[] args) {
    ShowFlowLayout frame = new ShowFlowLayout();
    frame.setTitle("ShowFlowLayout");
    frame.setLocationRelativeTo(null);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(200, 200);
    frame.setVisible(true);
  }
}
```

Note: it’s common to make the Frame an application class by including a *main* method. The main method will instantiate its own class.

A Frame class that uses FlowLayout layout manager



```java
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JFrame;
import java.awt.FlowLayout;

public class ShowFlowLayout extends JFrame {
  public ShowFlowLayout() {
    // Set FlowLayout, aligned left with horizontal gap 10
    // and vertical gap 20 between components
    setLayout(new FlowLayout(FlowLayout.LEFT, 10, 20));

    // Add labels and text fields to the frame
    add(new JLabel("First Name"));
    add(new JTextField(8));
    add(new JLabel("MI"));
    add(new JTextField(1));
    add(new JLabel("Last Name"));
    add(new JTextField(8));
  }

  /** Main method */
```

Swing components are in java.swing package

Layout managers are in java.awt package

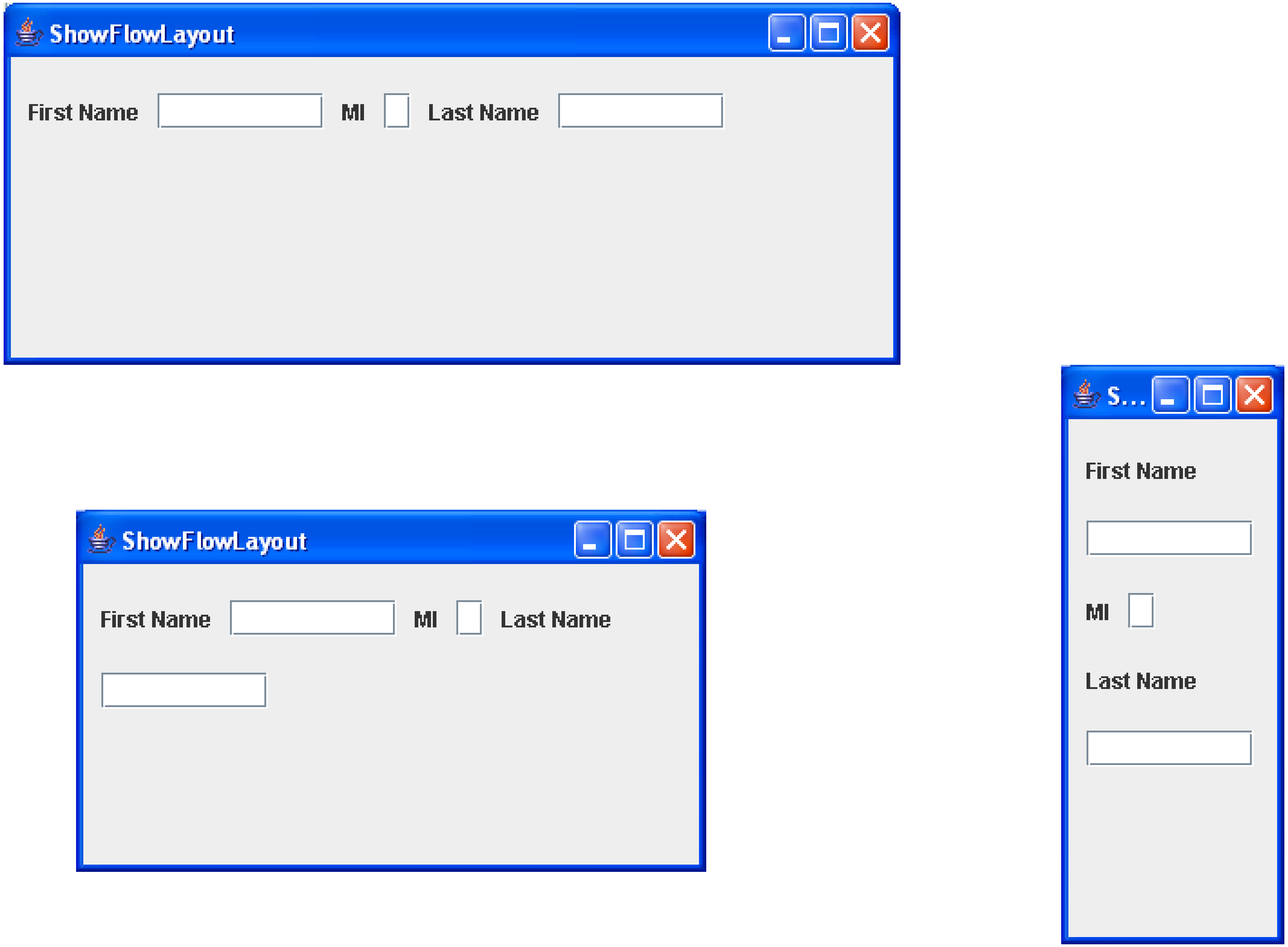
1

2

The constructor will typically do the following:

1) Set the layout manager for the frame’s content pane
2) Add the components to the frame’s content pane

In this case, the layout is Flow, and 6 Swing components are added



Resizing the frame causes the components to wrap around when necessary.

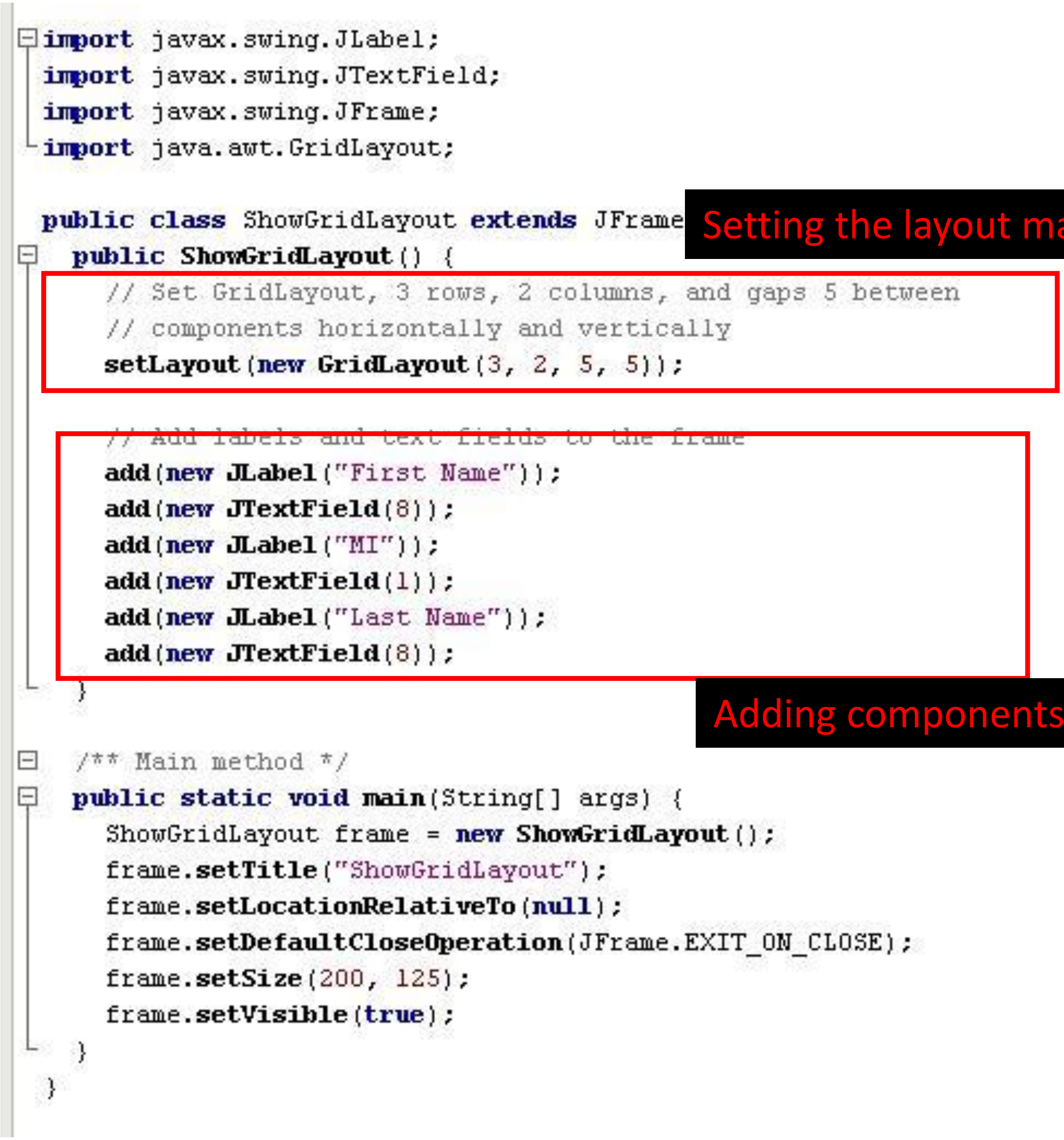
GridLayout

- Arranges components into rows and columns
- In Frame's constructor:
  - ***setLayout***

    (new GridLayout(rows,columns))

    OR
  - ***setLayout(new GridLayout(rows,columns,hgap,vgap))***
- Components will be added in order, left to right, row by row
- Components will be equal in size
- As container is resized, components will resize accordingly, and remain in same grid arrangement

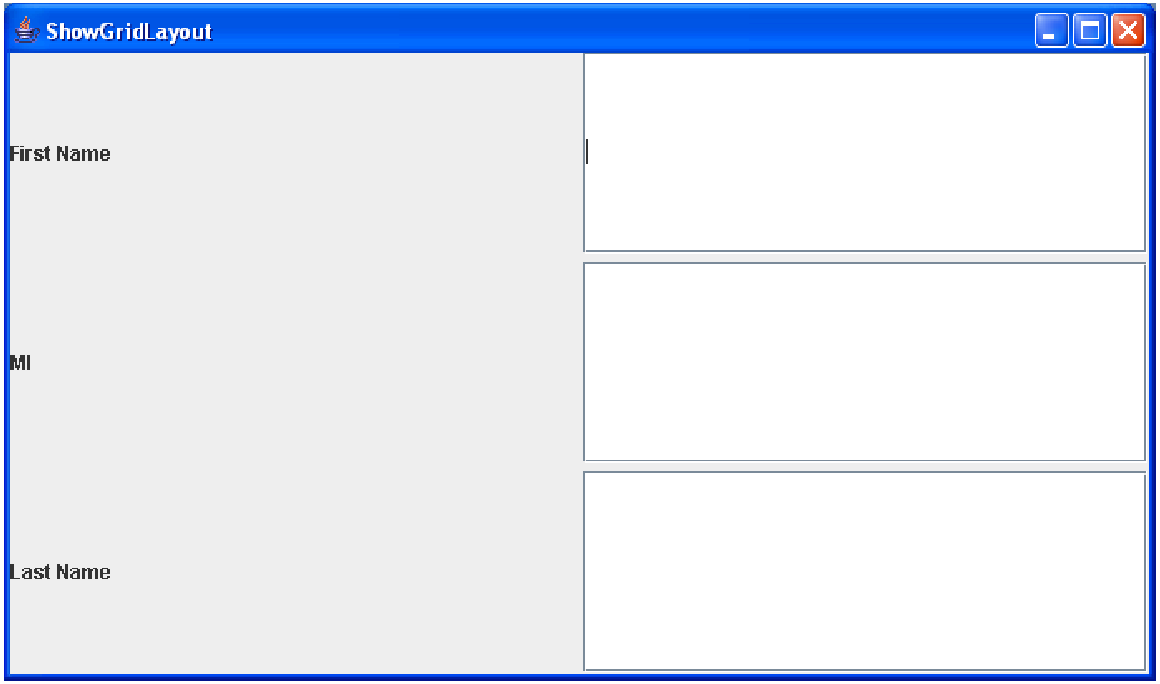
A Frame class that uses GridLayout layout manager

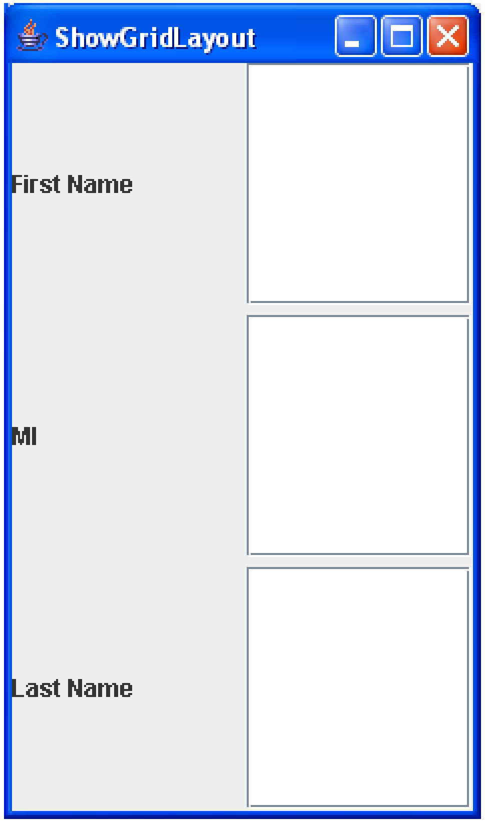


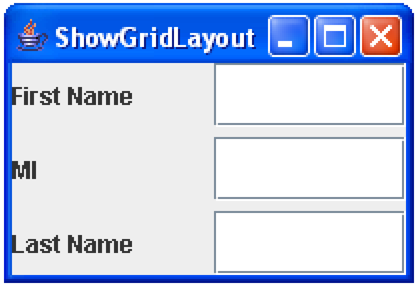
```
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JFrame;
import java.awt.GridLayout;

public class ShowGridLayout extends JFrame {
  public ShowGridLayout() {
    // Set GridLayout, 3 rows, 2 columns, and gaps 5 between
    // components horizontally and vertically
    setLayout(new GridLayout(3, 2, 5, 5));

    // Add labels and text fields to the frame
    add(new JLabel("First Name"));
    add(new JTextField(8));
    add(new JLabel("MI"));
    add(new JTextField(1));
    add(new JLabel("Last Name"));
    add(new JTextField(8));
  }

  /** Main method */
  public static void main(String[] args) {
    ShowGridLayout frame = new ShowGridLayout();
    frame.setTitle("ShowGridLayout");
    frame.setLocationRelativeTo(null);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(200, 125);
    frame.setVisible(true);
  }
}
```

Setting the layout manager

Adding components





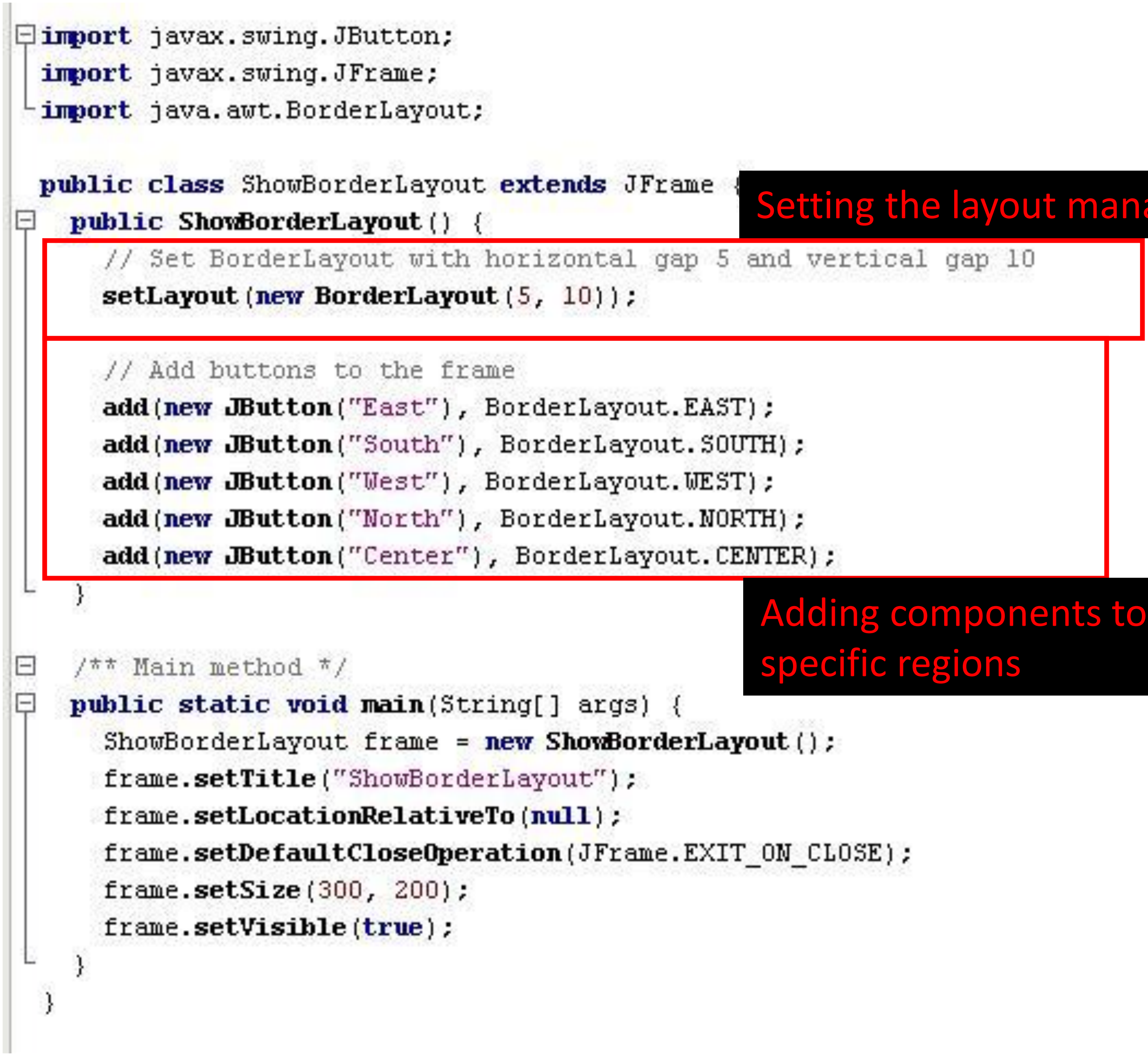


Resizing the frame causes the components to resize and maintain their same grid pattern.

BorderLayout

- Arranges components into five areas: North, South, East, West, and Center
- In the constructor:
  - ***setLayout(new BorderLayout())***
    - **OR**
  - ***setLayout(new BorderLayout(hgap,vgap))***
  - for each component:
    - ***add (the_component, region)***
    - do for each area desired:
      - BorderLayout.EAST, BorderLayout.SOUTH, BorderLayout.WEST, BorderLayout.NORTH, or BorderLayout.CENTER
- Behavior: when the container is resized, the components will be resized but remain in the same locations.
- NOTE: only a maximum of five components can be added and seen in this case, one to each region.

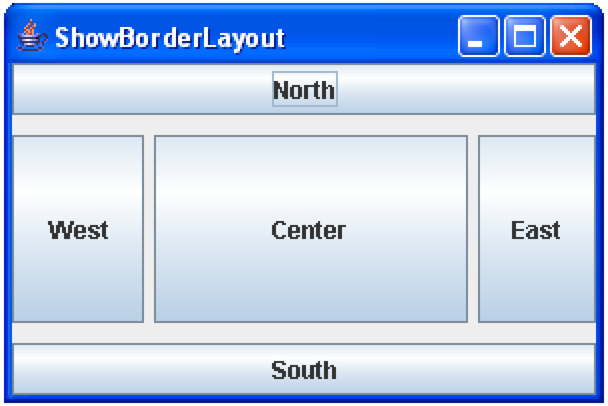
A Frame class that uses BorderLayout layout manager



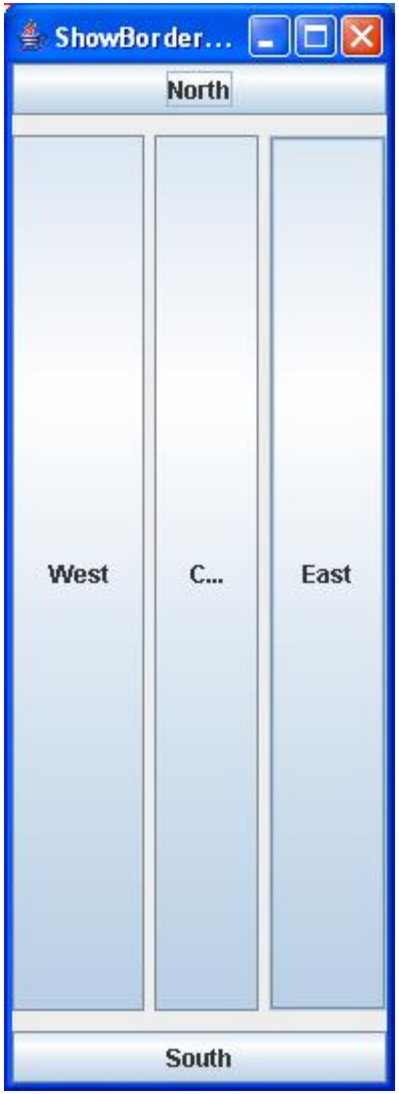
```java
import javax.swing.JButton;
import javax.swing.JFrame;
import java.awt.BorderLayout;

public class ShowBorderLayout extends JFrame {
  public ShowBorderLayout() {
    // Set BorderLayout with horizontal gap 5 and vertical gap 10
    setLayout(new BorderLayout(5, 10));

    // Add buttons to the frame
    add(new JButton("East"), BorderLayout.EAST);
    add(new JButton("South"), BorderLayout.SOUTH);
    add(new JButton("West"), BorderLayout.WEST);
    add(new JButton("North"), BorderLayout.NORTH);
    add(new JButton("Center"), BorderLayout.CENTER);
  }

  /** Main method */
  public static void main(String[] args) {
    ShowBorderLayout frame = new ShowBorderLayout();
    frame.setTitle("ShowBorderLayout");
    frame.setLocationRelativeTo(null);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(300, 200);
    frame.setVisible(true);
  }
}
```

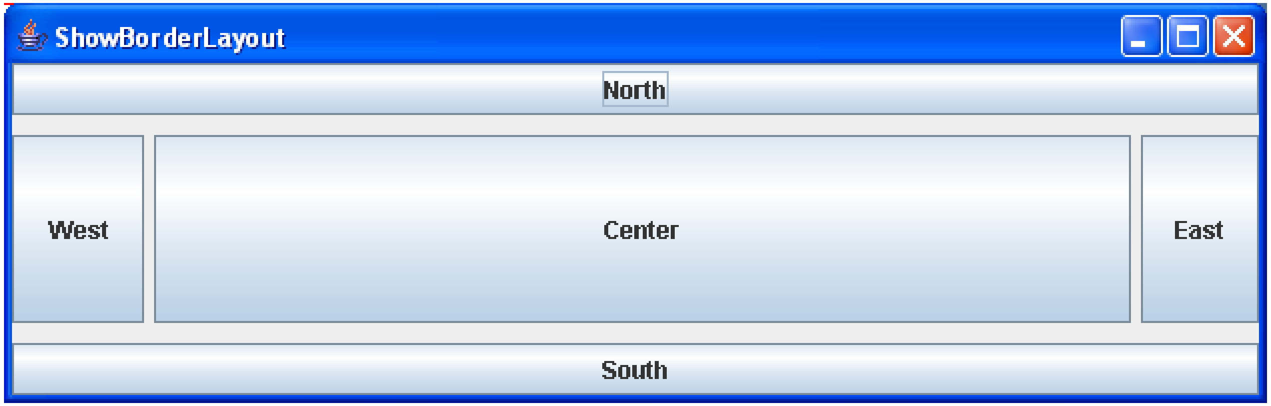
Setting the layout manager

Adding components to specific regions



Resizing the frame causes the components to resize and maintain their same regions.





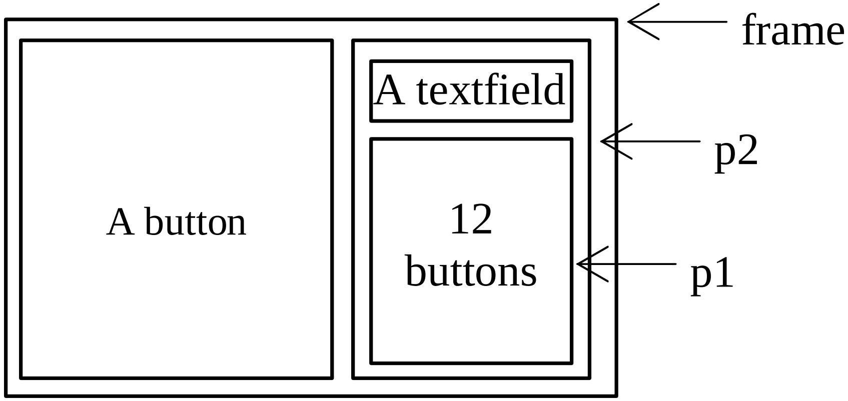
NOTE: the CENTER region dominates the sizing.

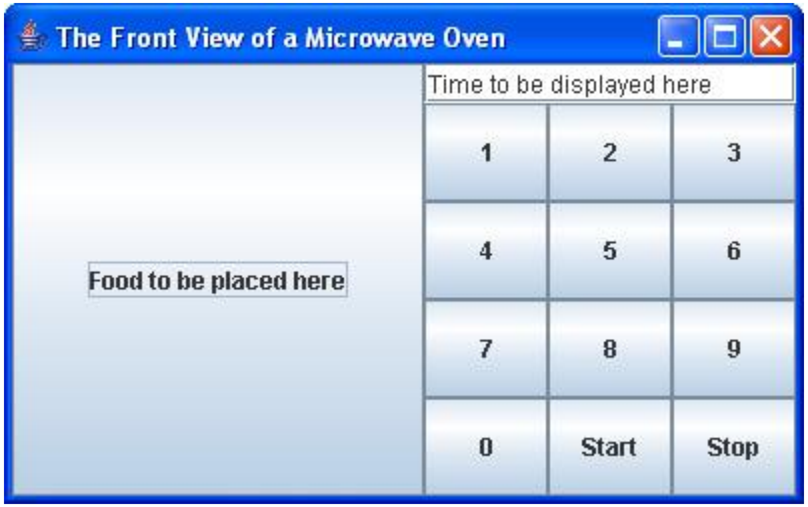
Using Panels as “Sub-Containers”

- JPanel is a container that can contain other components.
- As containers, JPanels can have their own layout managers.
- This way, you can combine layouts within the same frame by adding panels to the frame and by adding other components to the panels.
- Therefore, like JFrames, you can use these methods with JPanels:
  - add() – to add components to the panel
  - setLayout() – to associate a layout manager for the panel

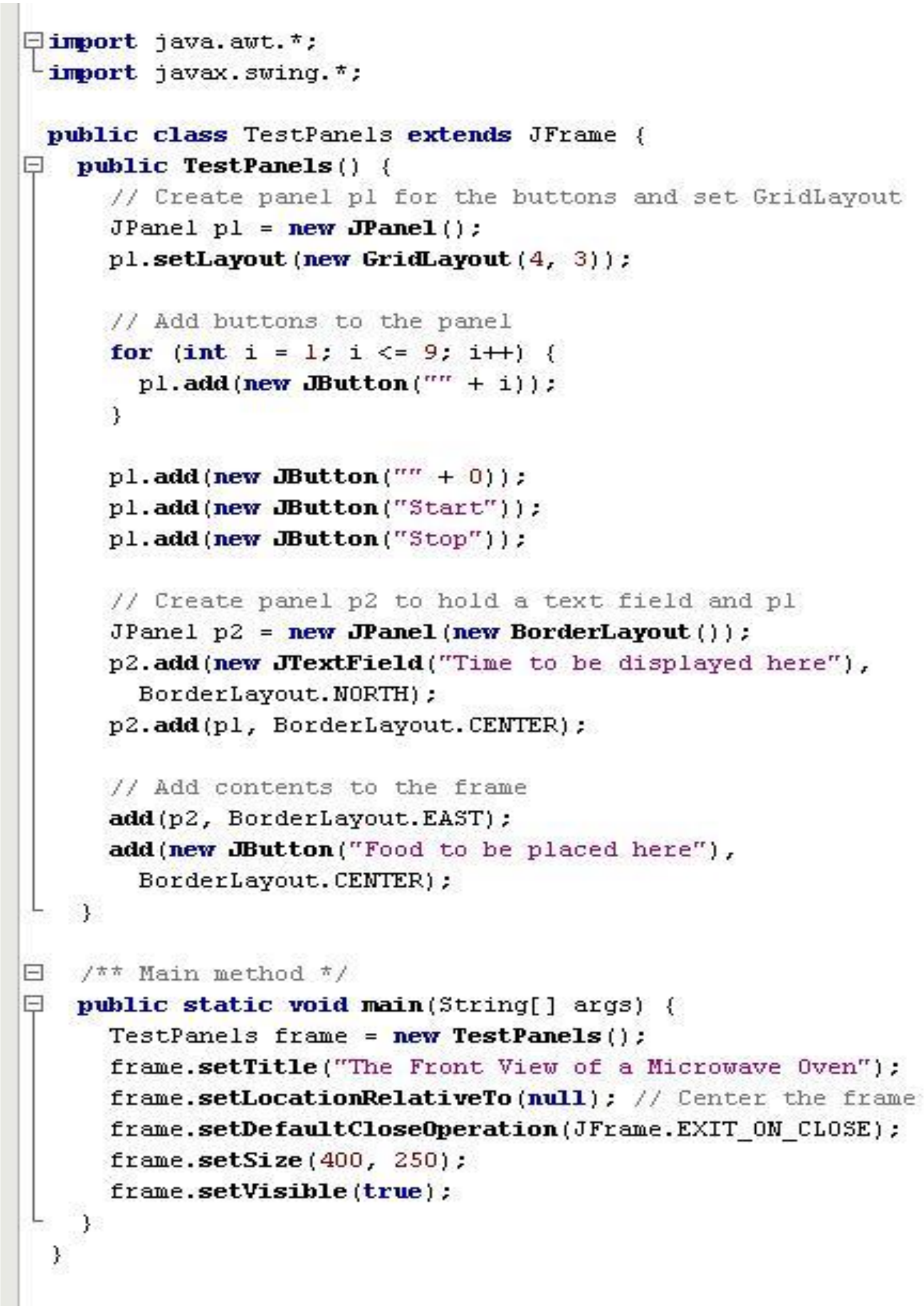
Using Panels

This example uses panels to organize components. The program creates a user interface for a Microwave oven.



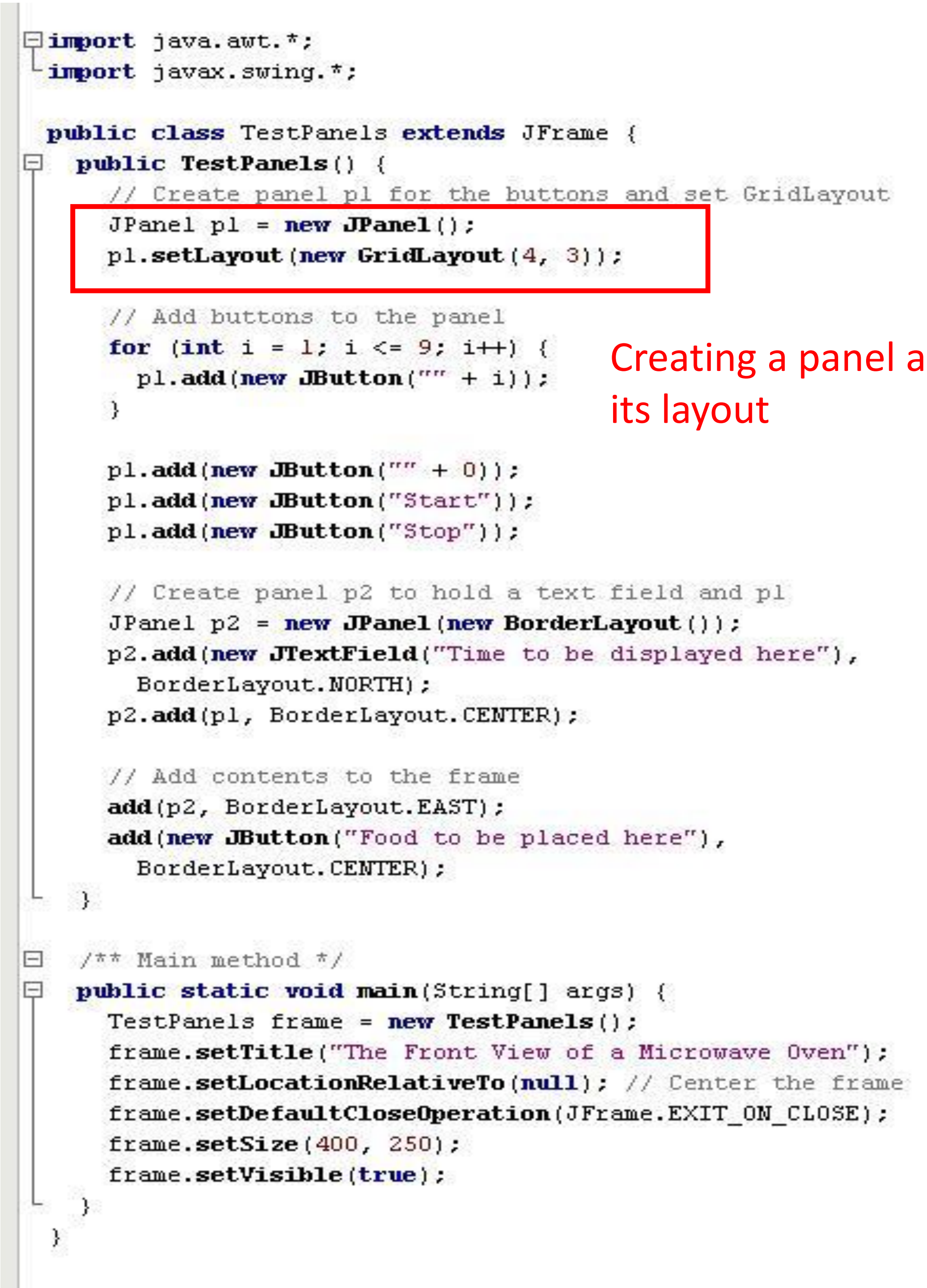


A Frame class that contains panels for organizing components



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

public class TestPanels extends JFrame {
  public TestPanels() {
    // Create panel p1 for the buttons and set GridLayout
    JPanel p1 = new JPanel();
    p1.setLayout(new GridLayout(4, 3));

    // Add buttons to the panel
    for (int i = 1; i <= 9; i++) {
      p1.add(new JButton("" + i));
    }

    p1.add(new JButton("" + 0));
    p1.add(new JButton("Start"));
    p1.add(new JButton("Stop"));

    // Create panel p2 to hold a text field and p1
    JPanel p2 = new JPanel(new BorderLayout());
    p2.add(new JTextField("Time to be displayed here"),
      BorderLayout.NORTH);
    p2.add(p1, BorderLayout.CENTER);

    // Add contents to the frame
    add(p2, BorderLayout.EAST);
    add(new JButton("Food to be placed here"),
      BorderLayout.CENTER);
  }

  /** Main method */
  public static void main(String[] args) {
    TestPanels frame = new TestPanels();
    frame.setTitle("The Front View of a Microwave Oven");
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(400, 250);
    frame.setVisible(true);
  }
}
```

A Frame class that contains panels for organizing components

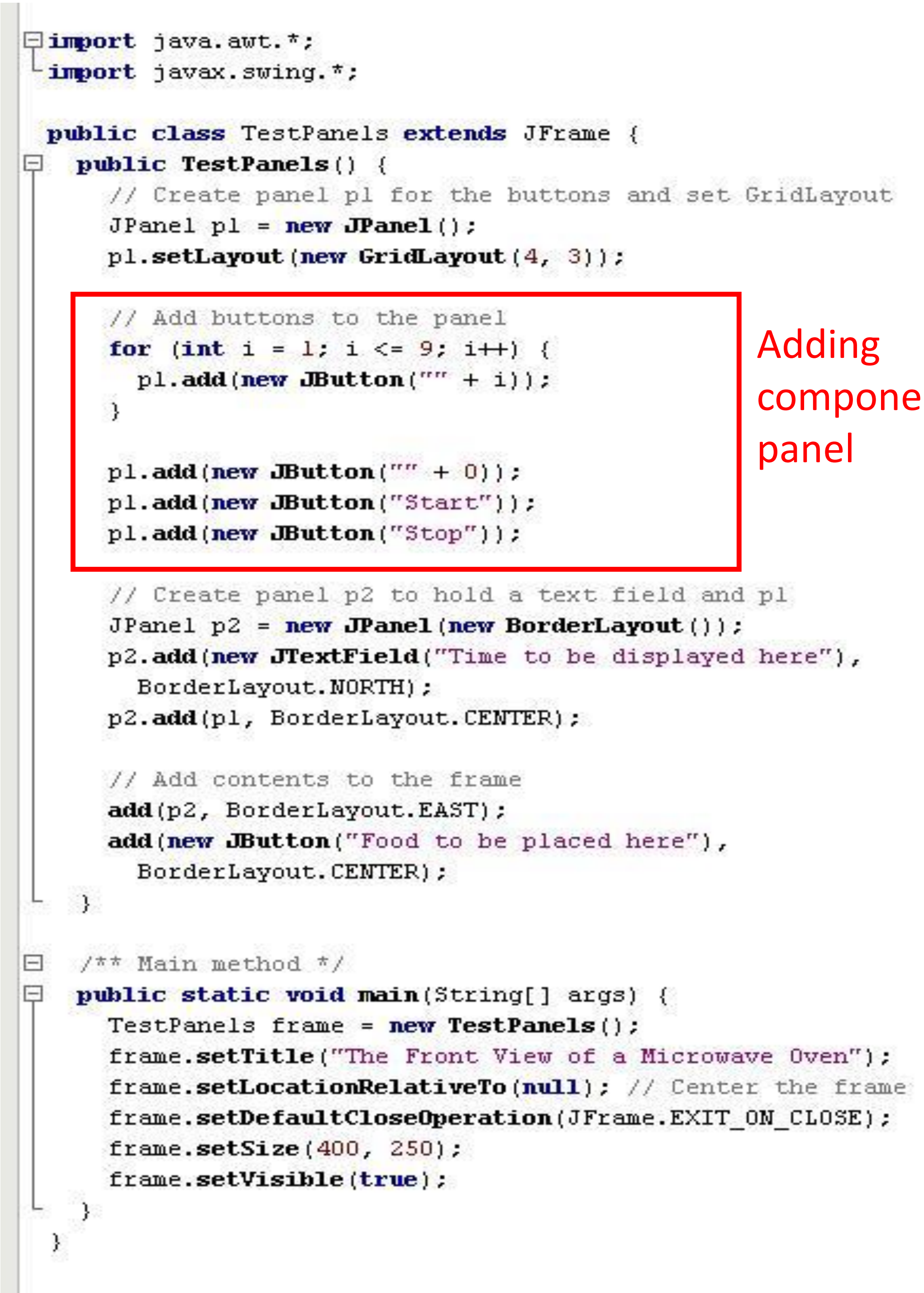


```
import java.awt.*;
import javax.swing.*;

public class TestPanels extends JFrame {
  public TestPanels() {
    // Create panel p1 for the buttons and set GridLayout
    JPanel p1 = new JPanel();
    p1.setLayout(new GridLayout(4, 3));

    // Add buttons to the panel
    for (int i = 1; i <= 9; i++) {
      p1.add(new JButton("" + i));
    }

    p1.add(new JButton("" + 0));
    p1.add(new JButton("Start"));
    p1.add(new JButton("Stop"));

    // Create panel p2 to hold a text field and p1
    JPanel p2 = new JPanel(new BorderLayout());
    p2.add(new JTextField("Time to be displayed here"),
      BorderLayout.NORTH);
    p2.add(p1, BorderLayout.CENTER);

    // Add contents to the frame
    add(p2, BorderLayout.EAST);
    add(new JButton("Food to be placed here"),
      BorderLayout.CENTER);
  }

  /** Main method */
  public static void main(String[] args) {
    TestPanels frame = new TestPanels();
    frame.setTitle("The Front View of a Microwave Oven");
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(400, 250);
    frame.setVisible(true);
  }
}
```

Creating a panel and setting its layout

Listing 12.6 p 414:

A Frame class that contains panels for organizing components

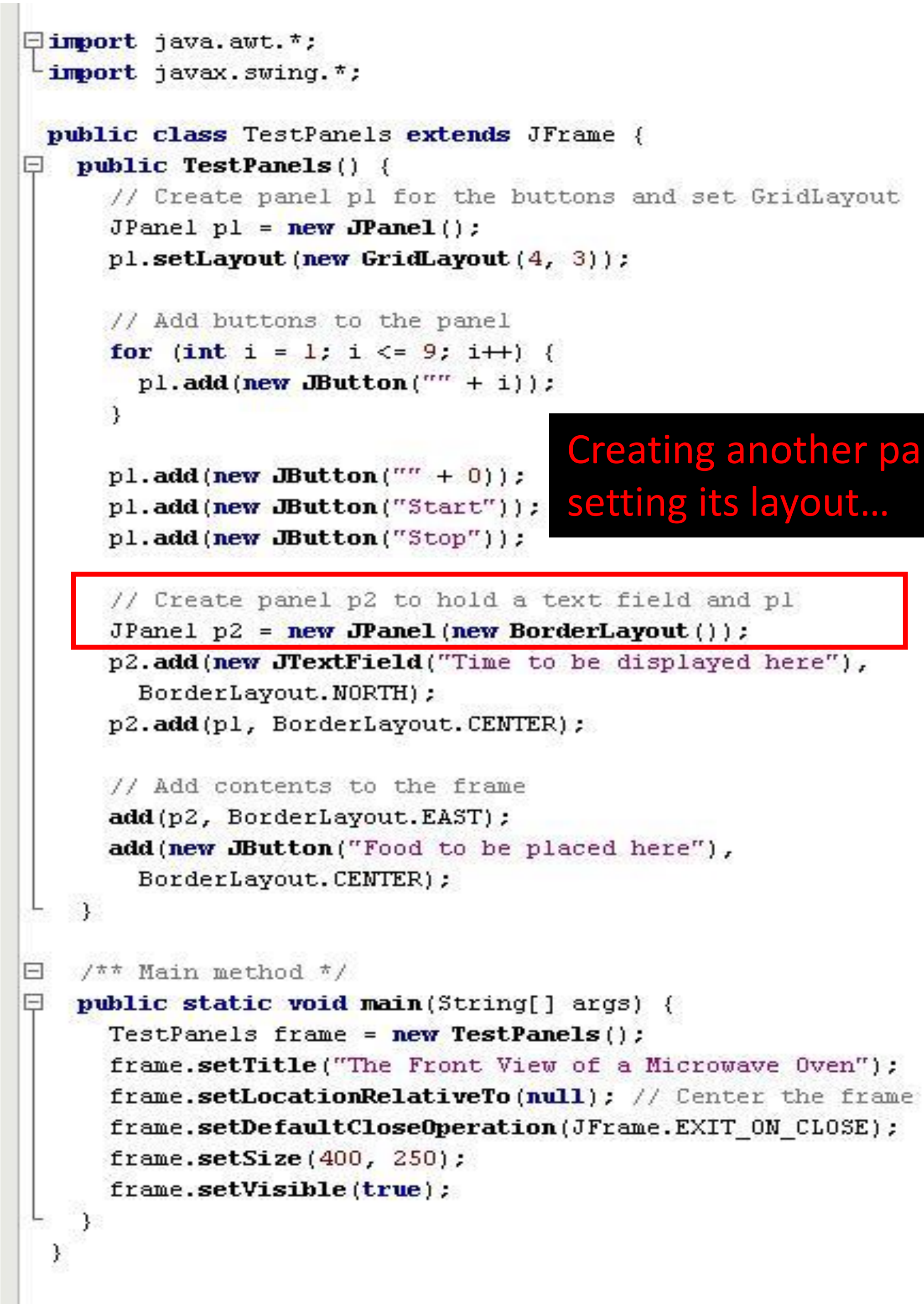


```
import java.awt.*;
import javax.swing.*;

public class TestPanels extends JFrame {
  public TestPanels() {
    // Create panel p1 for the buttons and set GridLayout
    JPanel p1 = new JPanel();
    p1.setLayout(new GridLayout(4, 3));

    // Add buttons to the panel
    for (int i = 1; i <= 9; i++) {
      p1.add(new JButton("" + i));
    }

    p1.add(new JButton("" + 0));
    p1.add(new JButton("Start"));
    p1.add(new JButton("Stop"));

    // Create panel p2 to hold a text field and p1
    JPanel p2 = new JPanel(new BorderLayout());
    p2.add(new JTextField("Time to be displayed here"),
      BorderLayout.NORTH);
    p2.add(p1, BorderLayout.CENTER);

    // Add contents to the frame
    add(p2, BorderLayout.EAST);
    add(new JButton("Food to be placed here"),
      BorderLayout.CENTER);
  }

  /** Main method */
  public static void main(String[] args) {
    TestPanels frame = new TestPanels();
    frame.setTitle("The Front View of a Microwave Oven");
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(400, 250);
    frame.setVisible(true);
  }
}
```

Adding components to the panel

Listing 12.6 p 414:

A Frame class that contains panels for organizing components

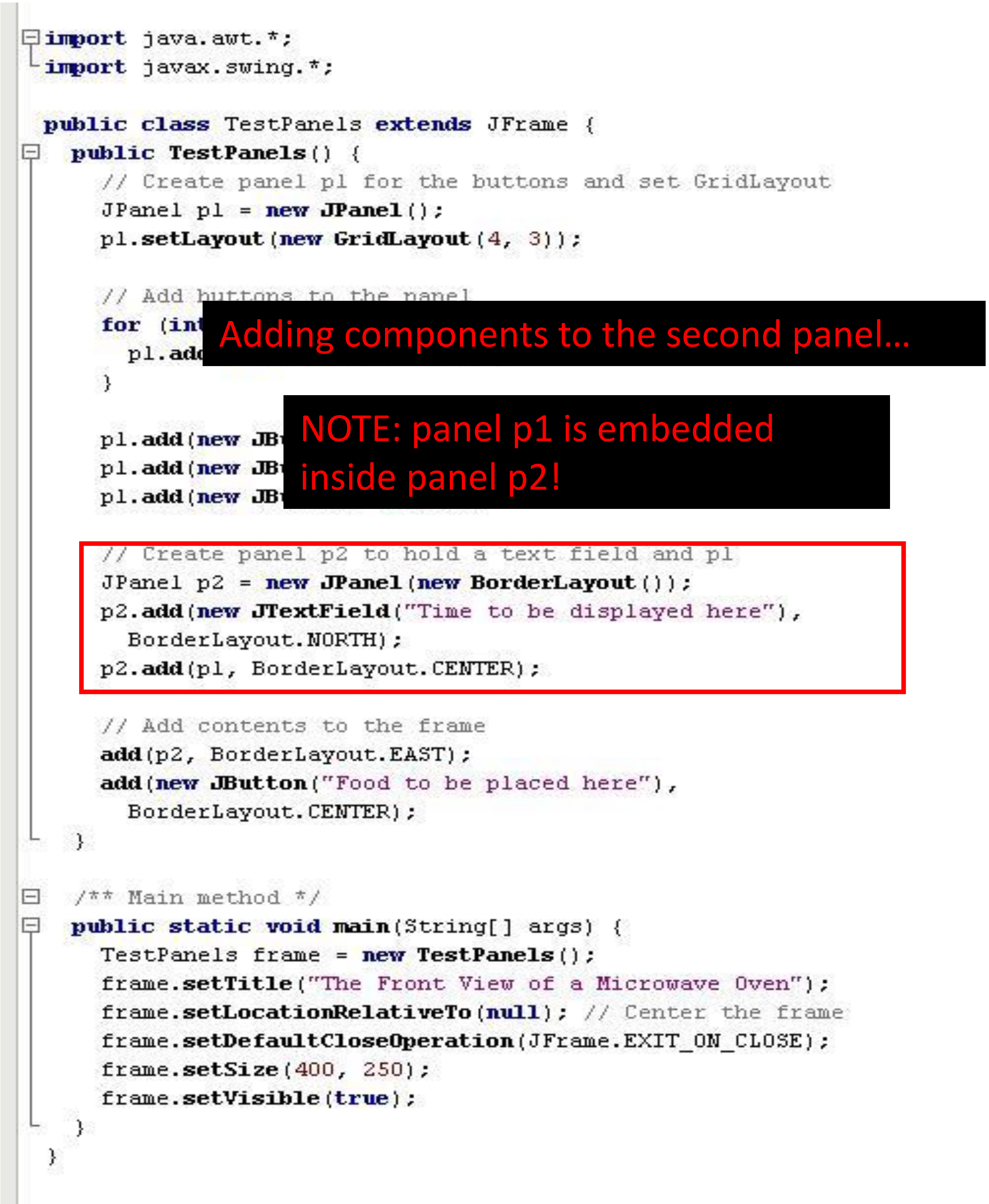


```
import java.awt.*;
import javax.swing.*;

public class TestPanels extends JFrame {
  public TestPanels() {
    // Create panel p1 for the buttons and set GridLayout
    JPanel p1 = new JPanel();
    p1.setLayout(new GridLayout(4, 3));

    // Add buttons to the panel
    for (int i = 1; i <= 9; i++) {
      p1.add(new JButton("" + i));
    }

    p1.add(new JButton("" + 0));
    p1.add(new JButton("Start"));
    p1.add(new JButton("Stop"));

    // Create panel p2 to hold a text field and p1
    JPanel p2 = new JPanel(new BorderLayout());
    p2.add(new JTextField("Time to be displayed here"),
      BorderLayout.NORTH);
    p2.add(p1, BorderLayout.CENTER);

    // Add contents to the frame
    add(p2, BorderLayout.EAST);
    add(new JButton("Food to be placed here"),
      BorderLayout.CENTER);
  }

  /** Main method */
  public static void main(String[] args) {
    TestPanels frame = new TestPanels();
    frame.setTitle("The Front View of a Microwave Oven");
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(400, 250);
    frame.setVisible(true);
  }
}
```

Creating another panel and setting its layout…

Listing 12.6 p 414:

A Frame class that contains panels for organizing components



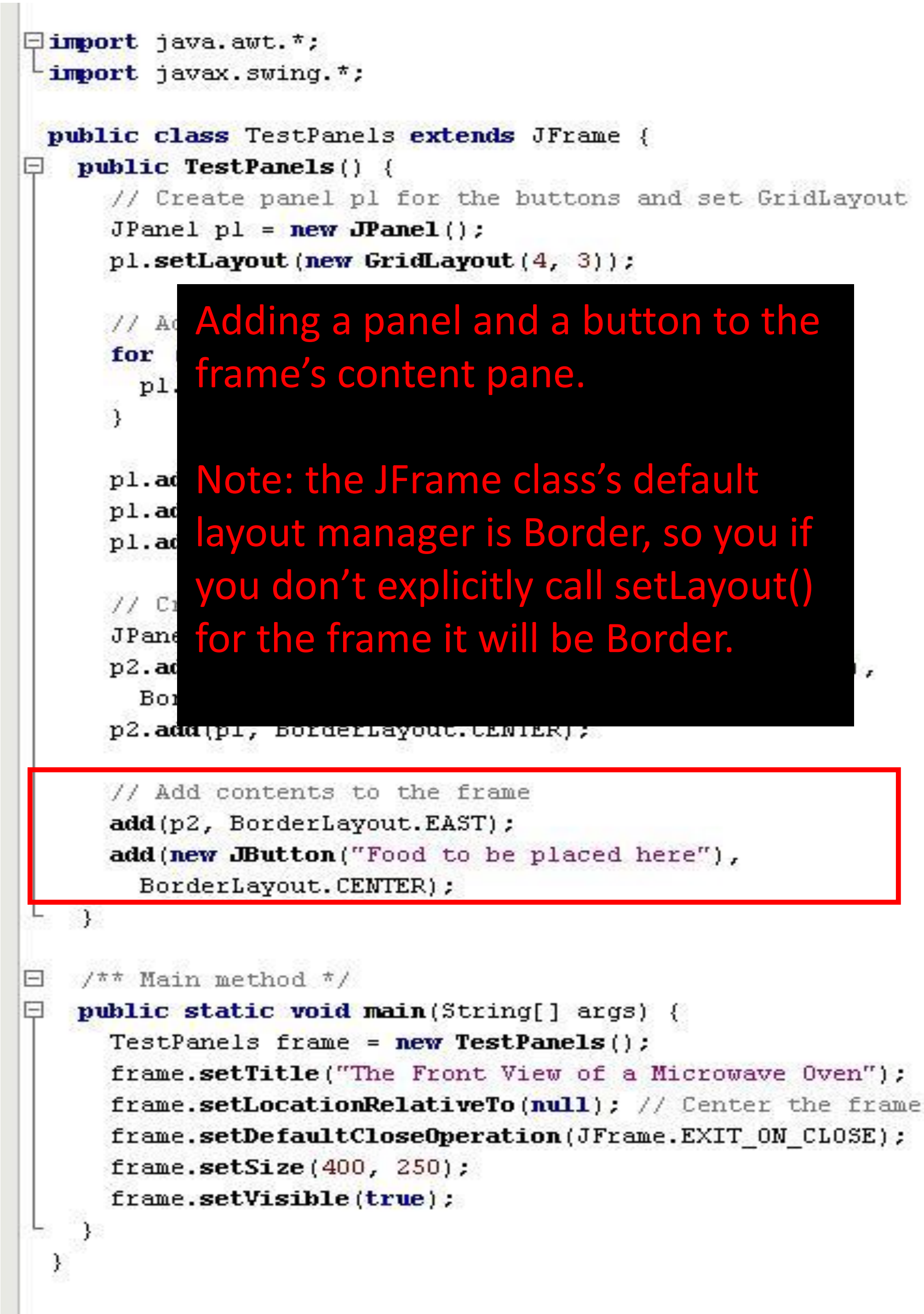
```
import java.awt.*;
import javax.swing.*;

public class TestPanels extends JFrame {
  public TestPanels() {
    // Create panel p1 for the buttons and set GridLayout
    JPanel p1 = new JPanel();
    p1.setLayout(new GridLayout(4, 3));

    // Add buttons to the panel
    for (int
      p1.add
    }

    p1.add(new JB
    p1.add(new JB
    p1.add(new JB

    // Create panel p2 to hold a text field and p1
    JPanel p2 = new JPanel(new BorderLayout());
    p2.add(new JTextField("Time to be displayed here"),
      BorderLayout.NORTH);
    p2.add(p1, BorderLayout.CENTER);

    // Add contents to the frame
    add(p2, BorderLayout.EAST);
    add(new JButton("Food to be placed here"),
      BorderLayout.CENTER);
  }

  /** Main method */
  public static void main(String[] args) {
    TestPanels frame = new TestPanels();
    frame.setTitle("The Front View of a Microwave Oven");
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(400, 250);
    frame.setVisible(true);
  }
}
```

Adding components to the second panel…

NOTE: panel p1 is embedded inside panel p2!

Listing 12.6 p 414:

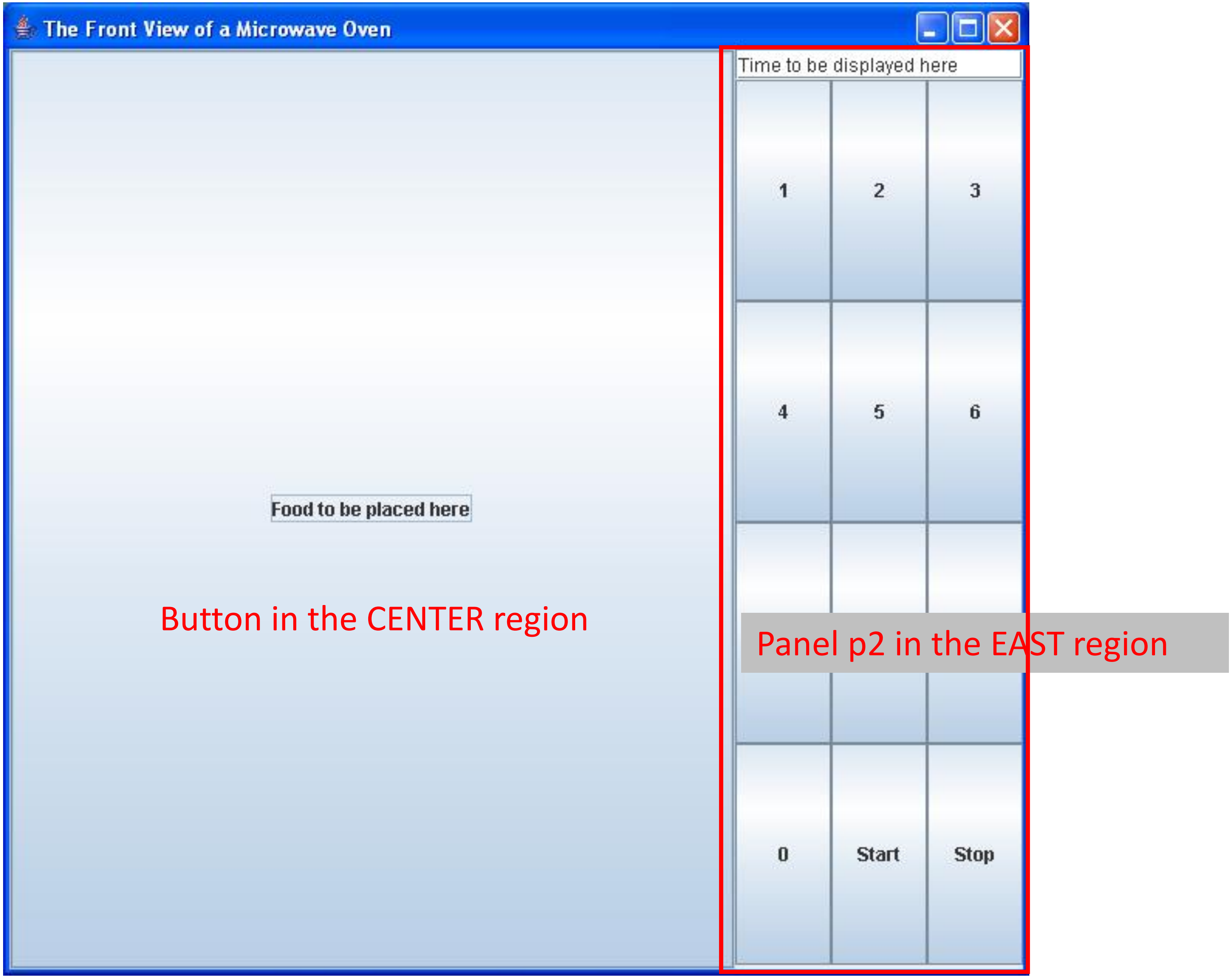
A Frame class that contains panels for organizing components



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

public class TestPanels extends JFrame {
  public TestPanels() {
    // Create panel p1 for the buttons and set GridLayout
    JPanel p1 = new JPanel();
    p1.setLayout(new GridLayout(4, 3));

    // Ad
    for (
      p1.
    }

    p1.ad
    p1.ad
    p1.ad

    // Cr
    JPane
    p2.ad                                              ),
      Bor
    p2.add(p1, BorderLayout.CENTER);

    // Add contents to the frame
    add(p2, BorderLayout.EAST);
    add(new JButton("Food to be placed here"),
      BorderLayout.CENTER);
  }

  /** Main method */
  public static void main(String[] args) {
    TestPanels frame = new TestPanels();
    frame.setTitle("The Front View of a Microwave Oven");
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(400, 250);
    frame.setVisible(true);
  }
}
```

Adding a panel and a button to the frame’s content pane.

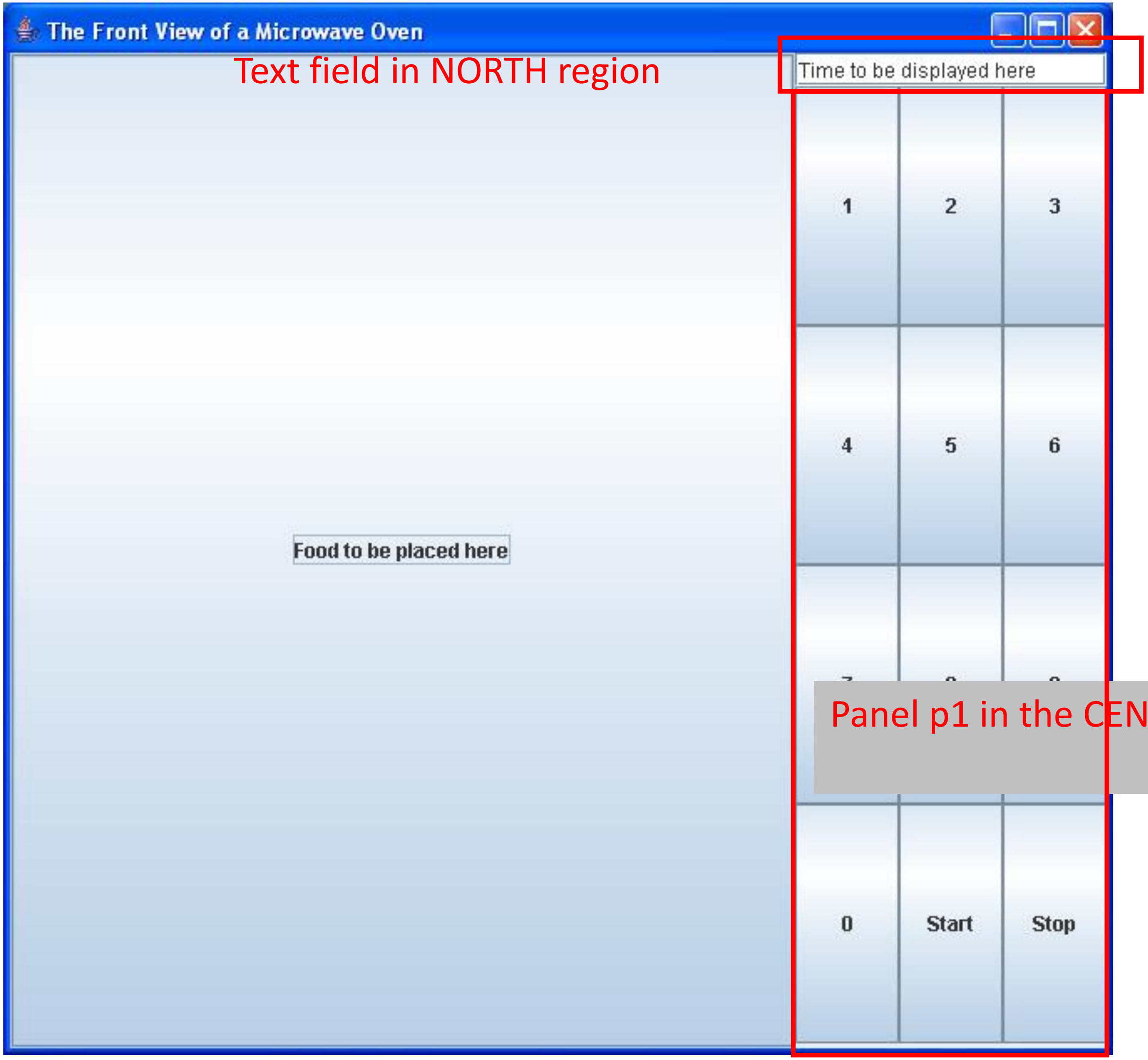
Note: the JFrame class’s default layout manager is Border, so you if you don’t explicitly call setLayout() for the frame it will be Border.



Button in the CENTER region

Panel p2 in the EAST region

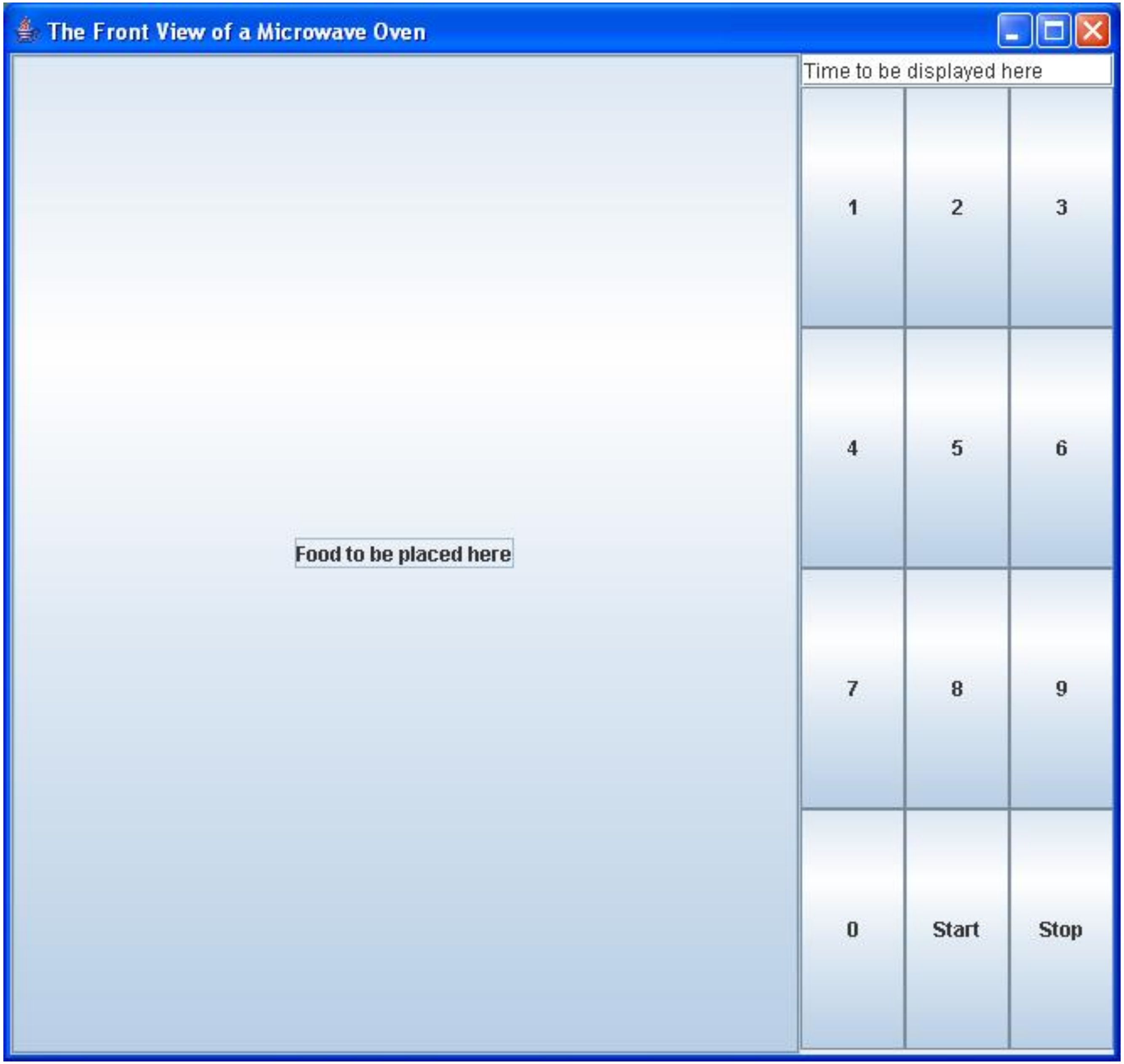
Frame has BorderLayout manager



Text field in NORTH region

Panel p1 in the CENTER region

Panel p2 has BorderLayout manager



Panel p1 has GridLayout manager, four rows and three columns

Absolute Positioning of Swing Components in a Container

- Not recommended because the container can be resized etc.
- Using the method:

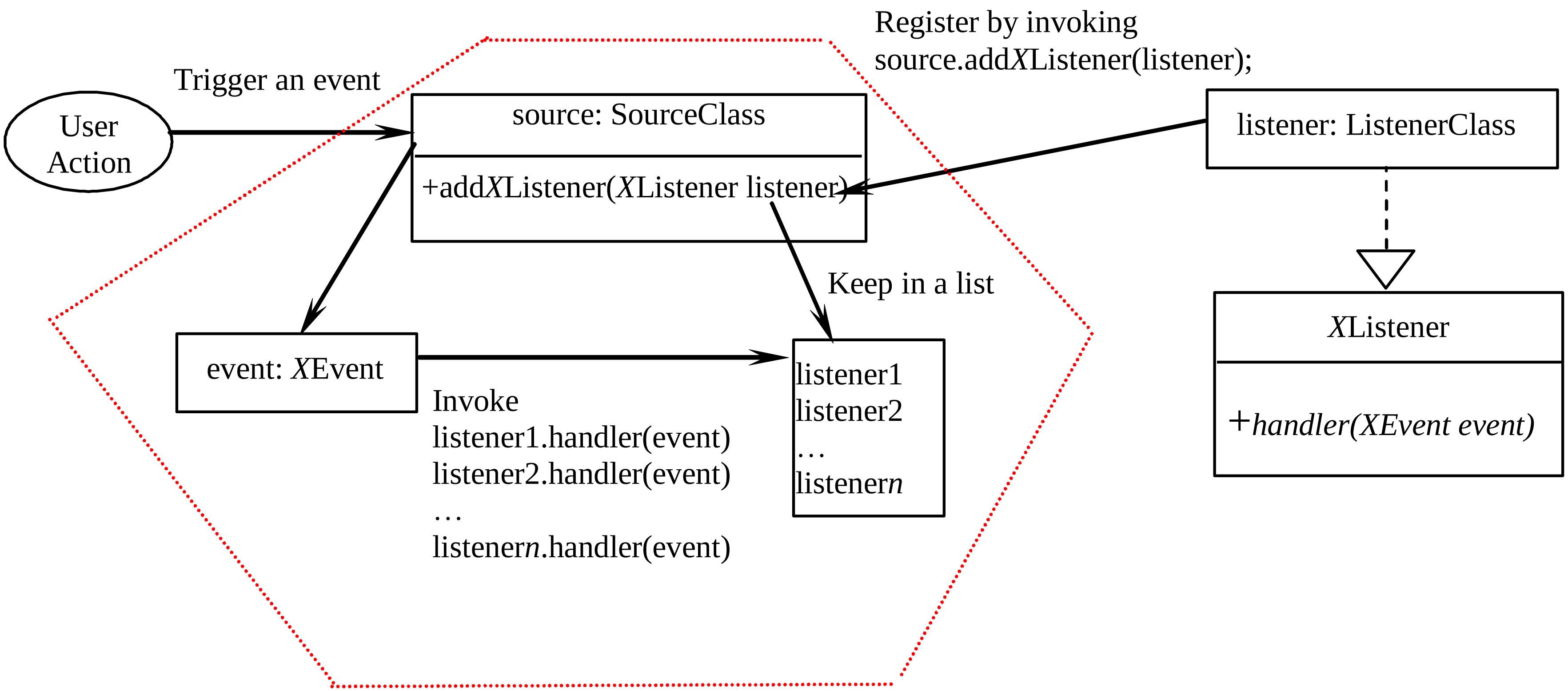
setBounds(int x, int y, int width, int height)

```
...
setSize(400, 400);
setDefaultCloseOperation(JFrame.EXIT_ON_
CLOSE);
JPanel panel = new JPanel(null);
JTextField textField = new
JTextField(20);
textField.setBounds(50, 50, 100, 20);
JButton button = new JButton("Button");
Button.setBounds(200, 100, 100, 20);
JCheckBox checkBox = new JCheckBox("Check
Me!");
checkBox.setBounds(300, 250, 100, 20);
panel.add(textField);
panel.add(button);
panel.add(checkBox);
setContentPane(panel);
...
```

Events and Listeners

- An *event* can be defined as a type of signal to the program that something has happened.
- The event is generated by external user actions such as mouse movements, mouse button clicks, and keystrokes, or by the operating system, such as a timer.
- Events are responded to by event *listeners*

Event Handling in Java



Event-generating Objects send Events to Listener Objects

Each event-generating object (usually a component) maintains a set of listeners for each event that it generates.

To be on the list, a listener object must register itself with the event-generating object.

Listeners have event-handling methods that respond to the event.

Selected User Actions

User Action	**Source Object**	**Event Type Generated**
Click a button	`JButton`	`ActionEvent`
Click a check box	`JCheckBox`	`ItemEvent`, `ActionEvent`
Click a radio button	`JRadioButton`	`ItemEvent`, `ActionEvent`
Press return on a text field	`JTextField`	`ActionEvent`
Select a new item	`JComboBox`	`ItemEvent`, `ActionEvent`
Select an item from a List	`JList`	`ListSelectionEvent`
Window opened, closed, etc.	`Window`	`WindowEvent`
Mouse pressed, released, etc.	Any `Component`	`MouseEvent`

Java AWT Event
Listener Interfaces

- ActionListener
- AdjustmentListener
- ComponentListener
- ContainerListener
- FocusListener
- ItemListener
- MouseListener
- MouseMotionListener
- TextListener
- WindowListener
- ListSelectionListener

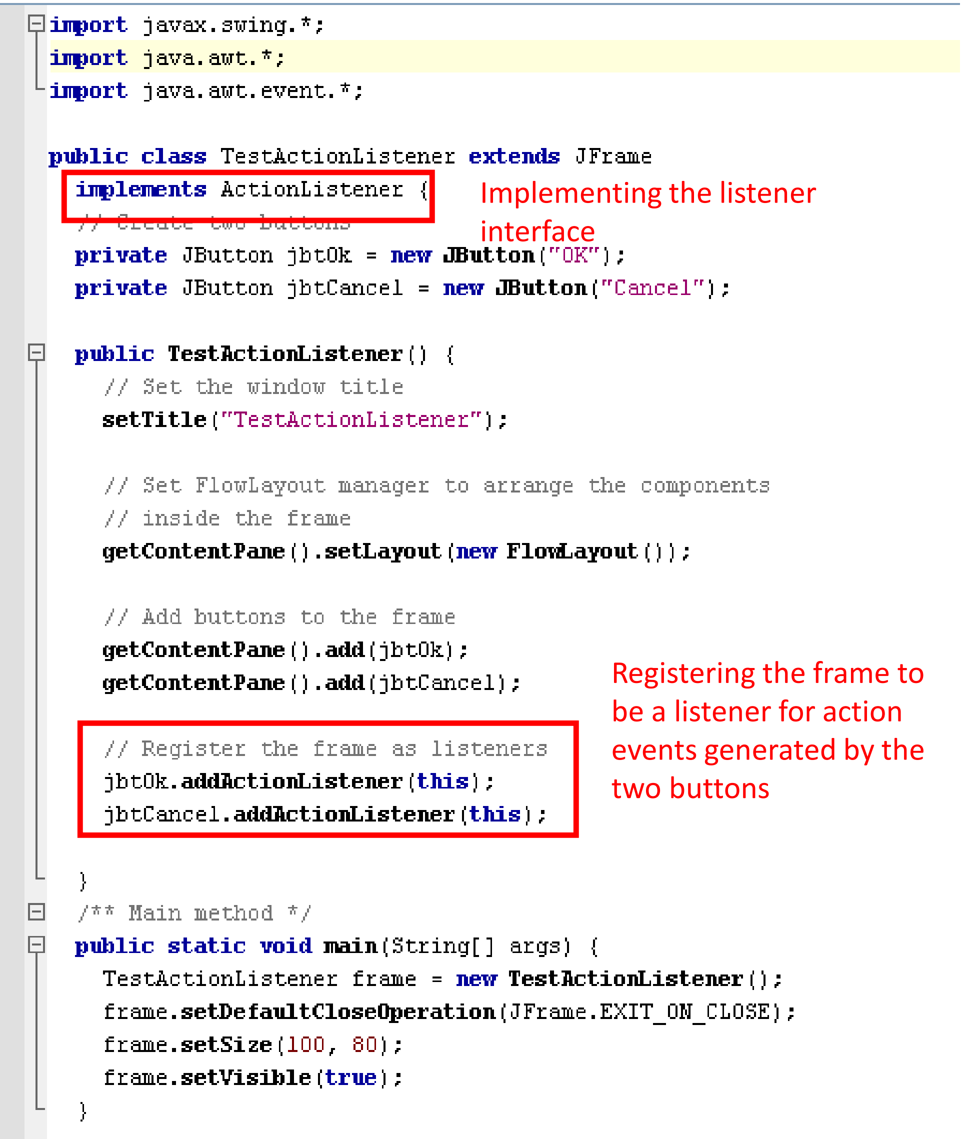
All are in the java.awt.event or javax.swing.event package
All are derived from EventListener in the java.util package

NOTE: any object that will respond to an event must **implement** a **listener interface**.

How to Implement a Listener Interface

- Use the **implements** keyword in the class declaration
- Register the object as a listener for a component’s event, using the component’s addXListener method. (where X is the type of event).

Handling Simple Action Events

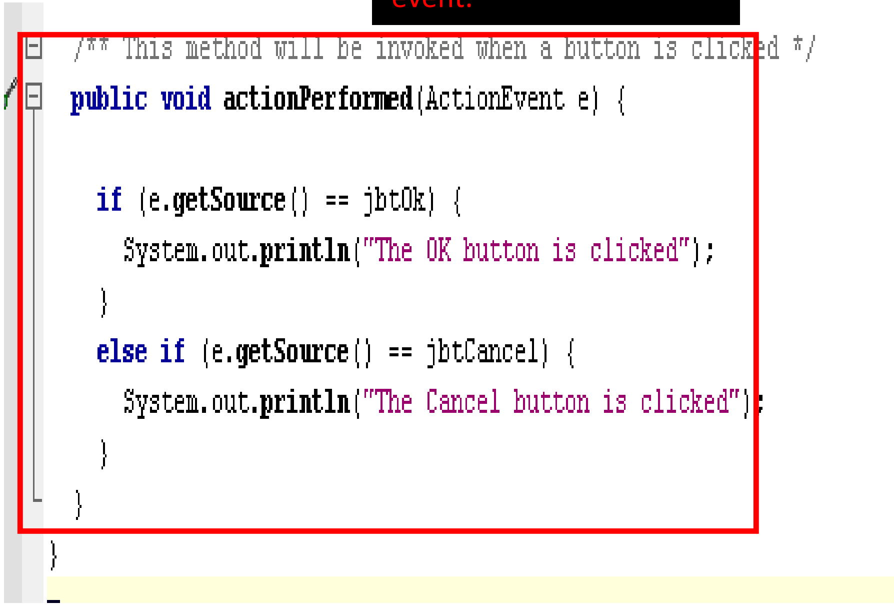


```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class TestActionListener extends JFrame
  implements ActionListener {
  // Create two buttons
  private JButton jbtOk = new JButton("OK");
  private JButton jbtCancel = new JButton("Cancel");

  public TestActionListener() {
    // Set the window title
    setTitle("TestActionListener");

    // Set FlowLayout manager to arrange the components
    // inside the frame
    getContentPane().setLayout(new FlowLayout());

    // Add buttons to the frame
    getContentPane().add(jbtOk);
    getContentPane().add(jbtCancel);

    // Register the frame as listeners
    jbtOk.addActionListener(this);
    jbtCancel.addActionListener(this);

  }
  /** Main method */
  public static void main(String[] args) {
    TestActionListener frame = new TestActionListener();
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(100, 80);
    frame.setVisible(true);
  }
```

Implementing the listener interface

Registering the frame to be a listener for action events generated by the two buttons

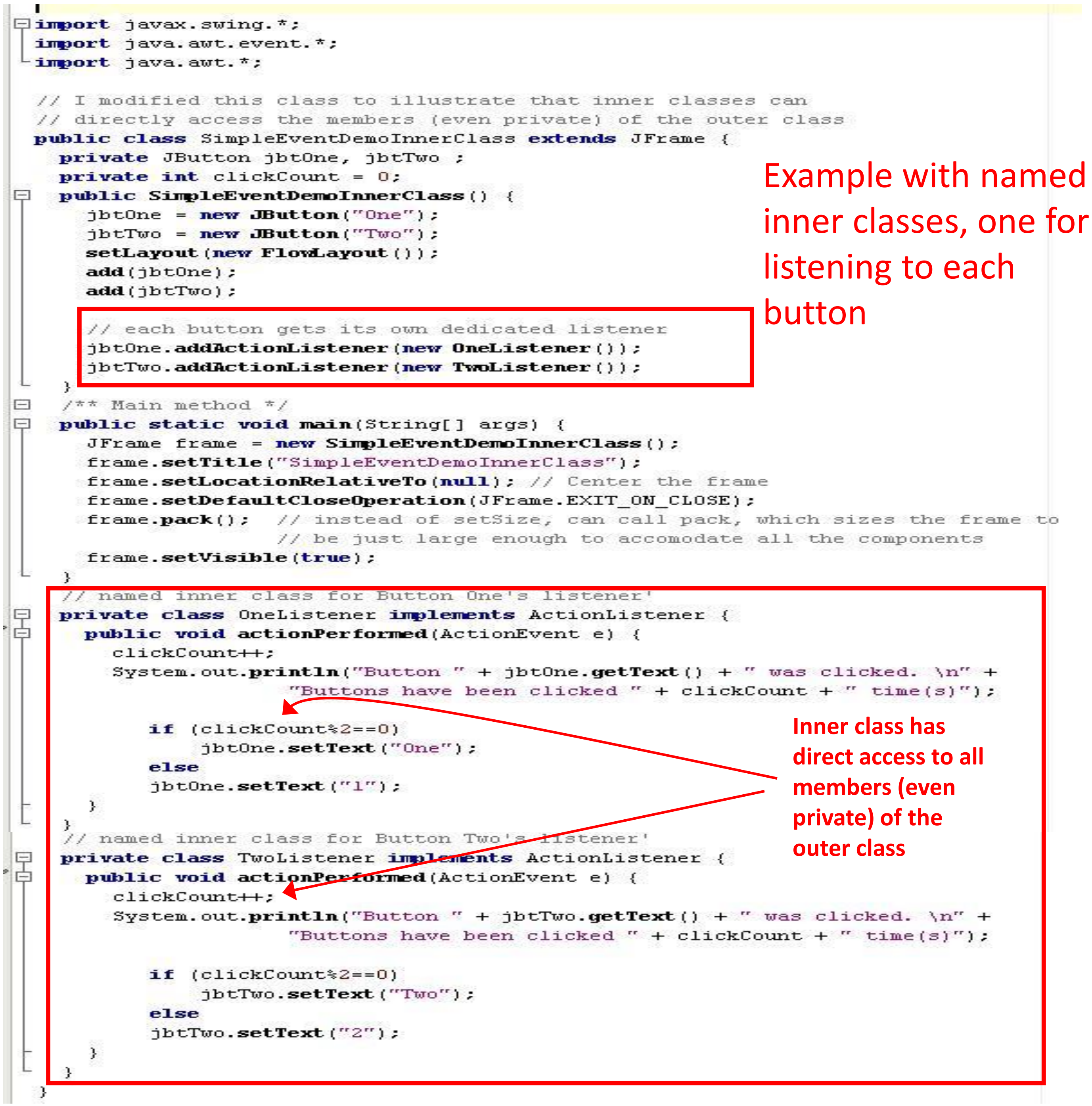
The method for responding to an Action event.



```java
  /** This method will be invoked when a button is clicked */
  public void actionPerformed(ActionEvent e) {

    if (e.getSource() == jbtOk) {
      System.out.println("The OK button is clicked");
    }
    else if (e.getSource() == jbtCancel) {
      System.out.println("The Cancel button is clicked");
    }
  }
}
```

Alternative Approaches to Listening

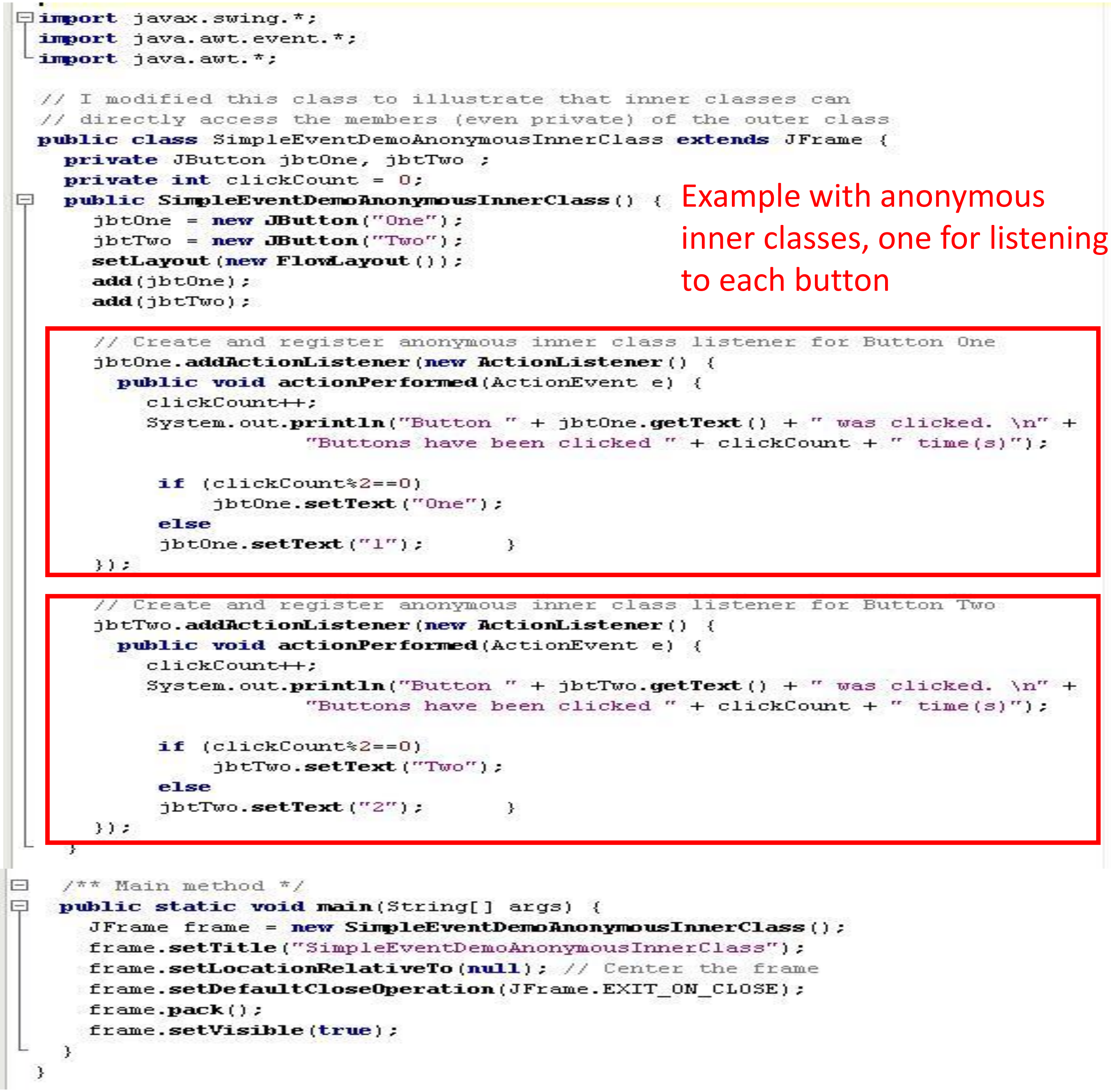
- Implement the listener with the **main application class**, and have the one listener assigned to all components generating the events
  - Advantage: simplicity for beginner programmers
  - Disadvantage: event-handler method may require if-statement or switch with several branches when multiple components generate the event
- Use **inner classes** to implement the listeners and create a different instance as each component's listener.
  - ***Named*** inner class or ***anonymous*** inner class (This is the approach used in the textbook most of the time)
  - Advantage: no need to test within the listeners for determining which component sent the event. Each component has its own dedicated listener
  - Disadvantage: harder to understand



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

// I modified this class to illustrate that inner classes can
// directly access the members (even private) of the outer class
public class SimpleEventDemoInnerClass extends JFrame {
  private JButton jbtOne, jbtTwo ;
  private int clickCount = 0;
  public SimpleEventDemoInnerClass() {
    jbtOne = new JButton("One");
    jbtTwo = new JButton("Two");
    setLayout(new FlowLayout());
    add(jbtOne);
    add(jbtTwo);

    // each button gets its own dedicated listener
    jbtOne.addActionListener(new OneListener());
    jbtTwo.addActionListener(new TwoListener());
  }
  /** Main method */
  public static void main(String[] args) {
    JFrame frame = new SimpleEventDemoInnerClass();
    frame.setTitle("SimpleEventDemoInnerClass");
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.pack();   // instead of setSize, can call pack, which sizes the frame to
                    // be just large enough to accomodate all the components
    frame.setVisible(true);
  }
  // named inner class for Button One's listener'
  private class OneListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      clickCount++;
      System.out.println("Button " + jbtOne.getText() + " was clicked. \n" +
                    "Buttons have been clicked " + clickCount + " time(s)");

         if (clickCount%2==0)
             jbtOne.setText("One");
         else
         jbtOne.setText("1");
    }
  }
  // named inner class for Button Two's listener'
  private class TwoListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      clickCount++;
      System.out.println("Button " + jbtTwo.getText() + " was clicked. \n" +
                    "Buttons have been clicked " + clickCount + " time(s)");

         if (clickCount%2==0)
             jbtTwo.setText("Two");
         else
         jbtTwo.setText("2");
    }
  }
}
```

Example with named inner classes, one for listening to each button

Inner class has direct access to all members (even private) of the outer class



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

// I modified this class to illustrate that inner classes can
// directly access the members (even private) of the outer class
public class SimpleEventDemoAnonymousInnerClass extends JFrame {
 private JButton jbtOne, jbtTwo ;
 private int clickCount = 0;
 public SimpleEventDemoAnonymousInnerClass() {
   jbtOne = new JButton("One");
   jbtTwo = new JButton("Two");
   setLayout(new FlowLayout());
   add(jbtOne);
   add(jbtTwo);

  // Create and register anonymous inner class listener for Button One
  jbtOne.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
    clickCount++;
    System.out.println("Button " + jbtOne.getText() + " was clicked. \n" +
          "Buttons have been clicked " + clickCount + " time(s)");

    if (clickCount%2==0)
        jbtOne.setText("One");
    else
    jbtOne.setText("1");       }
  });

  // Create and register anonymous inner class listener for Button Two
  jbtTwo.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
    clickCount++;
    System.out.println("Button " + jbtTwo.getText() + " was clicked. \n" +
          "Buttons have been clicked " + clickCount + " time(s)");

    if (clickCount%2==0)
        jbtTwo.setText("Two");
    else
    jbtTwo.setText("2");       }
  });
 }

 /** Main method */
 public static void main(String[] args) {
   JFrame frame = new SimpleEventDemoAnonymousInnerClass();
   frame.setTitle("SimpleEventDemoAnonymousInnerClass");
   frame.setLocationRelativeTo(null); // Center the frame
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   frame.pack();
   frame.setVisible(true);
 }
}
```

Example with anonymous inner classes, one for listening to each button