5COSC019W – Object Oriented Programming Week 7

Dr. Barbara Villarini

b.villarini@westminster.ac.uk

UNIVERSITY OF WESTMINSTER

Elements of GUI Programming

- Components
  - Visual objects that appear on the screen
- Layouts
  - Control over the positioning of components within a *container*
- Events
  - Responses to user actions

Implementing GUI in Java

Two categories of Java Component classes:

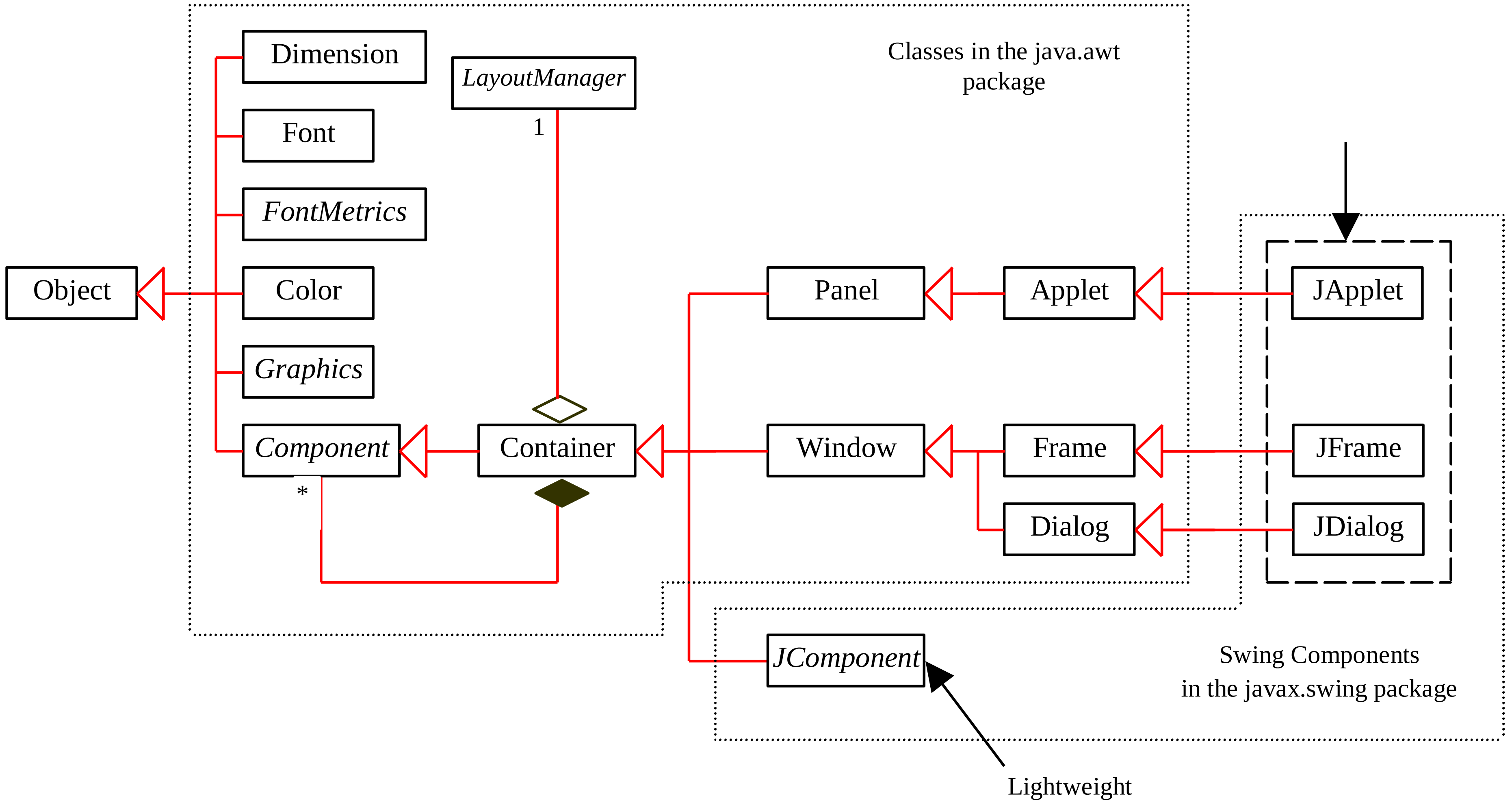
- AWT – Abstract Windows Toolkit (java.awt package)
  - The older version of the components
  - Rely on “peer architecture”…drawing done by the OS platform on which the application/applet is running
  - Considered to be “heavy-weight”
- Swing (javax.swing package)
  - Newer version of the components
  - No “peer architecture”…components draw themselves
  - Most are considered to be “lightweight”

Implementing GUI in Java : Java FX

- **JavaFX** is a software platform for creating and delivering desktop applications, as well as **rich internet applications** (RIAs) that can run across a wide variety of devices.

- It includes all these features in a single library: Media, UI controls, Web, 2D and 3D, etc.. In addition, the developers can also access the existing features of a Java library such as Swings.

Container Class
GUI Class Hierarchy (Swing)

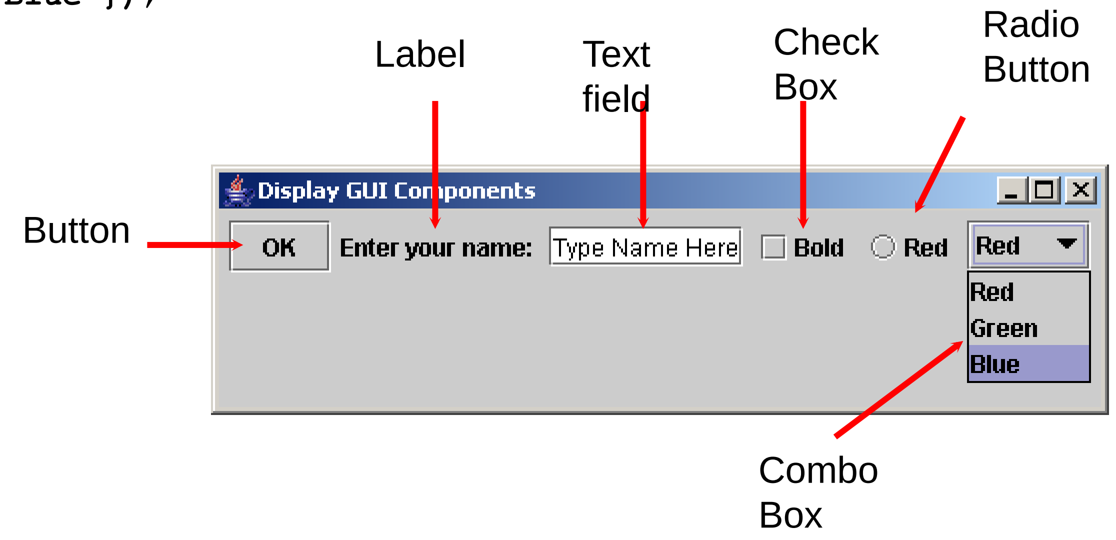


AWT vs. Swing

- Swing does not replace the AWT; it is built on top of it
- All the AWT components are heavyweight: the corresponding Swing components are lightweights
- Swing component names begin with “J”:
  - `Component` (AWT) – `JComponent` (Swing)
  - `Button` (AWT) – `JButton` (Swing)
- Always use Swing components: however, since Swing is built on top of AWT, you will need to know some AWT methods

Creating GUI Objects

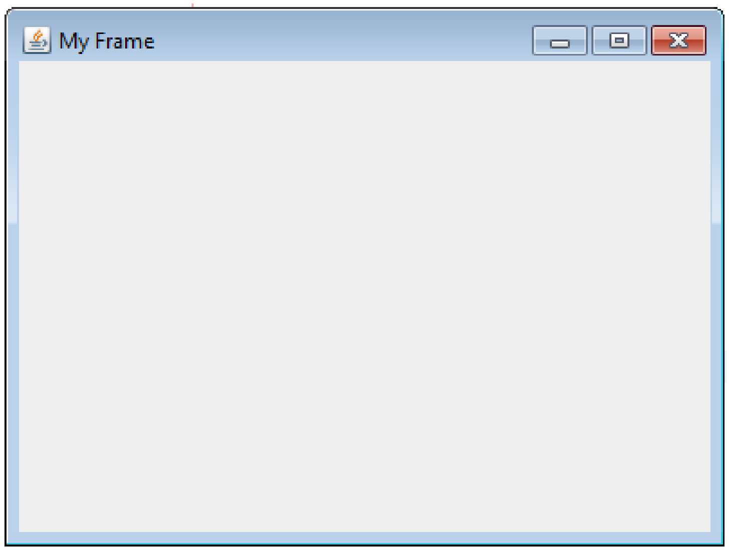
```
// Create a button with text OK
JButton jbtOK = new JButton("OK");

// Create a label with text "Enter your name: "
JLabel jlblName = new JLabel("Enter your name: ");

// Create a text field with text "Type Name Here"
JTextField jtfName = new JTextField("Type Name Here");

// Create a check box with text bold
JCheckBox jchkBold = new JCheckBox("Bold");

// Create a radio button with text red
JRadioButton jrbRed = new JRadioButton("Red");

// Create a combo box with choices red, green, and blue
JComboBox jcboColor = new JComboBox(new String[]{"Red",
  "Green", "Blue"});
```



UNIVERSITY OF WESTMINSTER

Frames

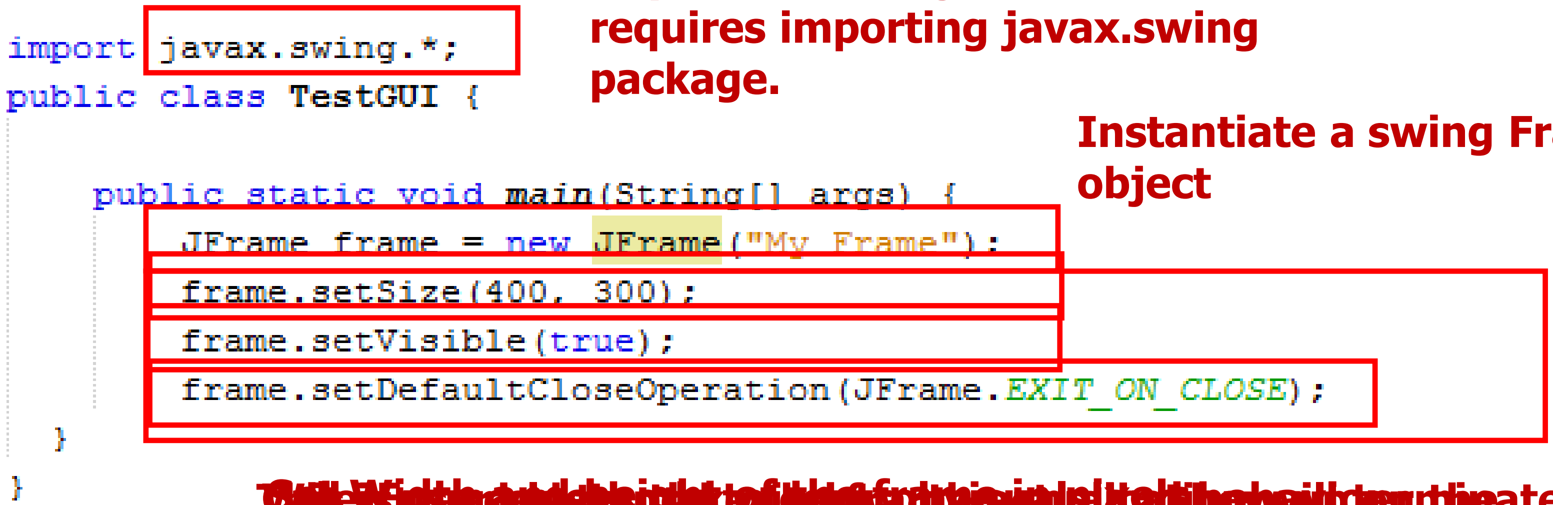
- Frame is a window that is not contained inside another window.
- Frame is the basis to contain other user interface components in Java graphical applications.
- The Frame class can be used to create windows.



MyFrame

Any use of Swing classes requires importing javax.swing package.

Instantiate a swing Frame object

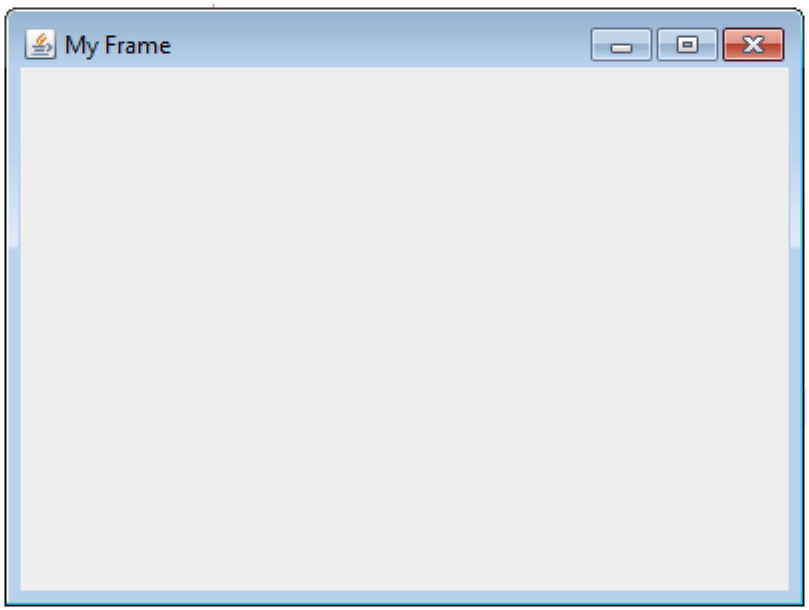


```
import javax.swing.*;
public class TestGUI {

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

[illegible]
screen

UNIVERSITY OF WESTMINSTER

MyFrame



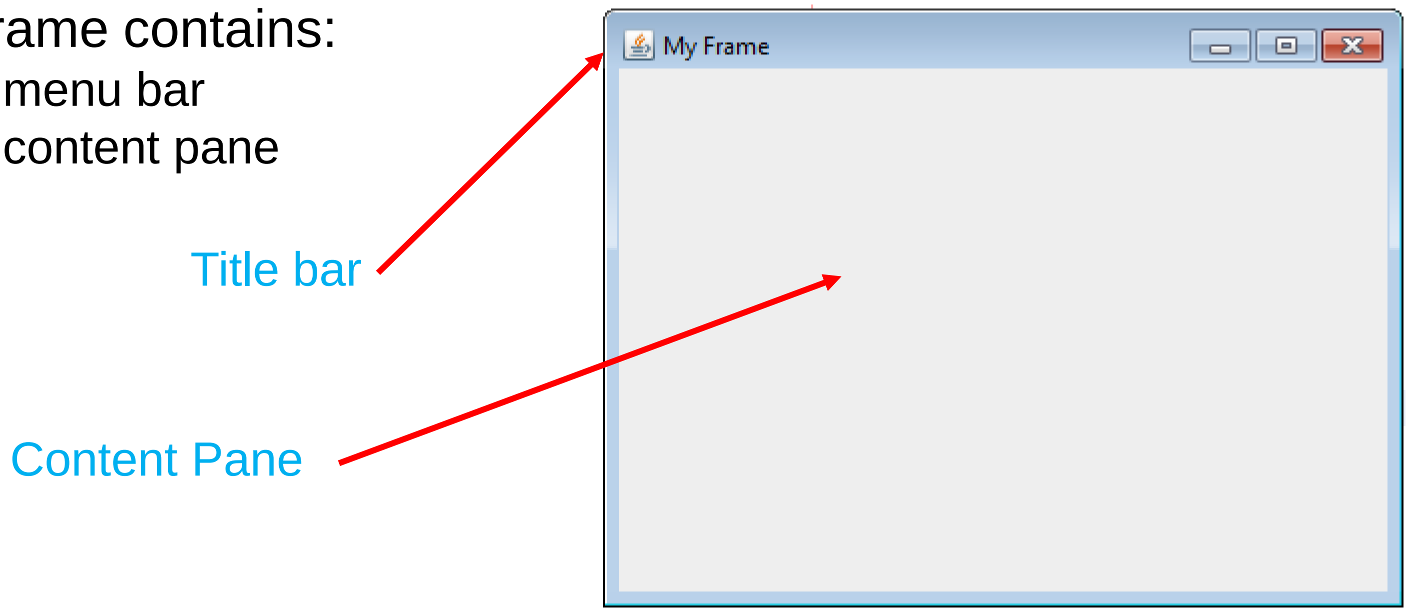
This is what a frame looks like.

Note the title bar, the content area, the minimize, maximize/restore, and close icons.

Caption in the title bar was determined from the argument to the constructor.

Frames with Components

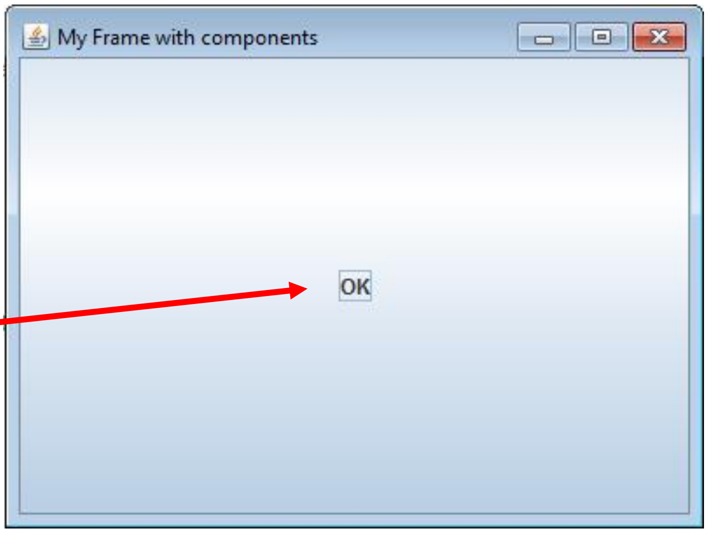
- A Frame is a container. Therefore it can contain other components (like buttons, text fields, etc.)
- Components are added to the content pane of a frame.
- The content pane is the grey area in the Frame window.
- A simplistic way to look at containment is this:
  - A JFrame contains:
    1. A menu bar
    2. A content pane



UNIVERSITY OF WESTMINSTER

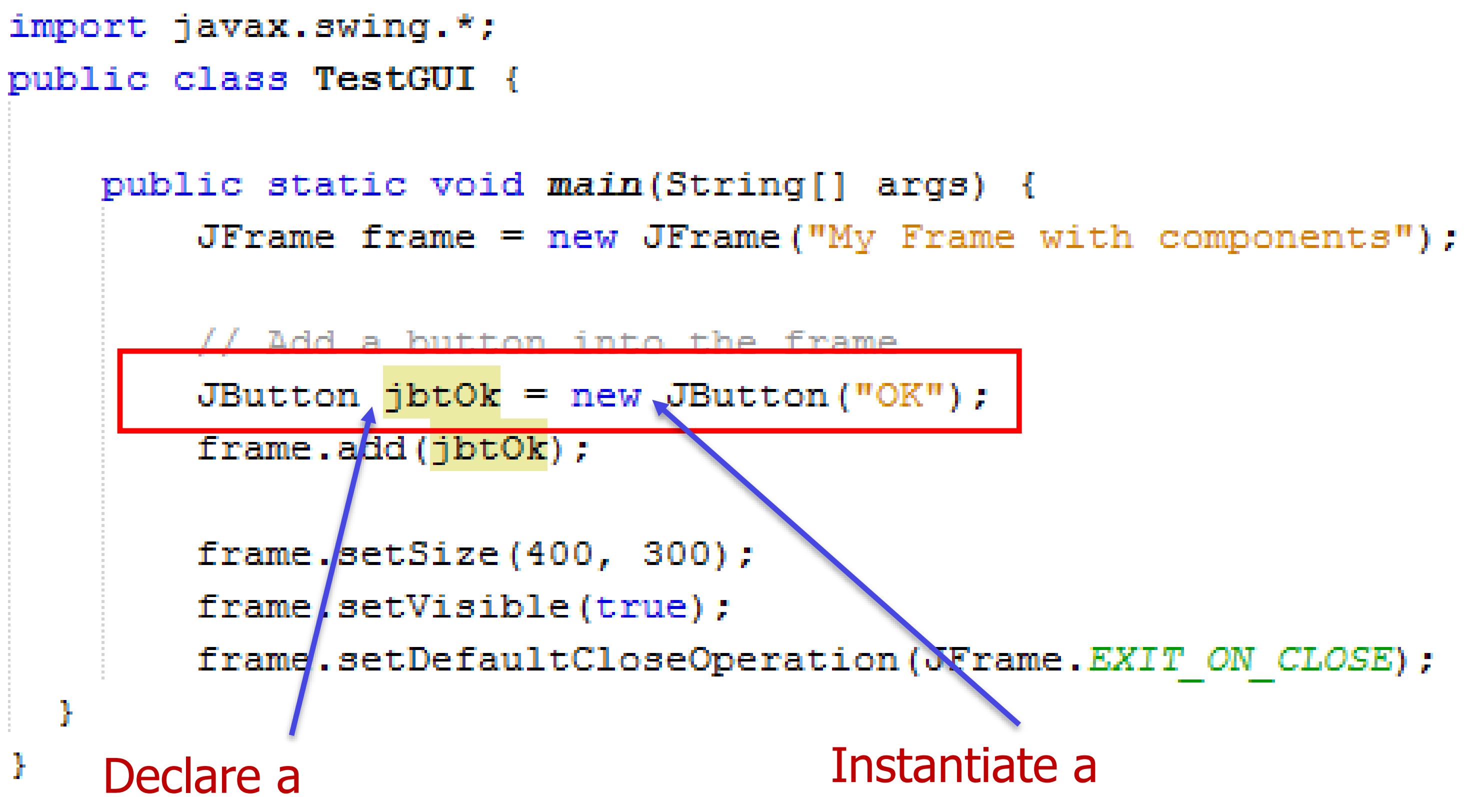
Adding component into a Frame

```
import javax.swing.*;
public class TestGUI {

    public static void main(String[] args) {
        JFrame frame = new JFrame("My Frame with components");

        // Add a button into the frame
        JButton jbtOk = new JButton("OK");
        frame.add(jbtOk);

        frame.setSize(400, 300);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```



Button

Adding component into a Frame



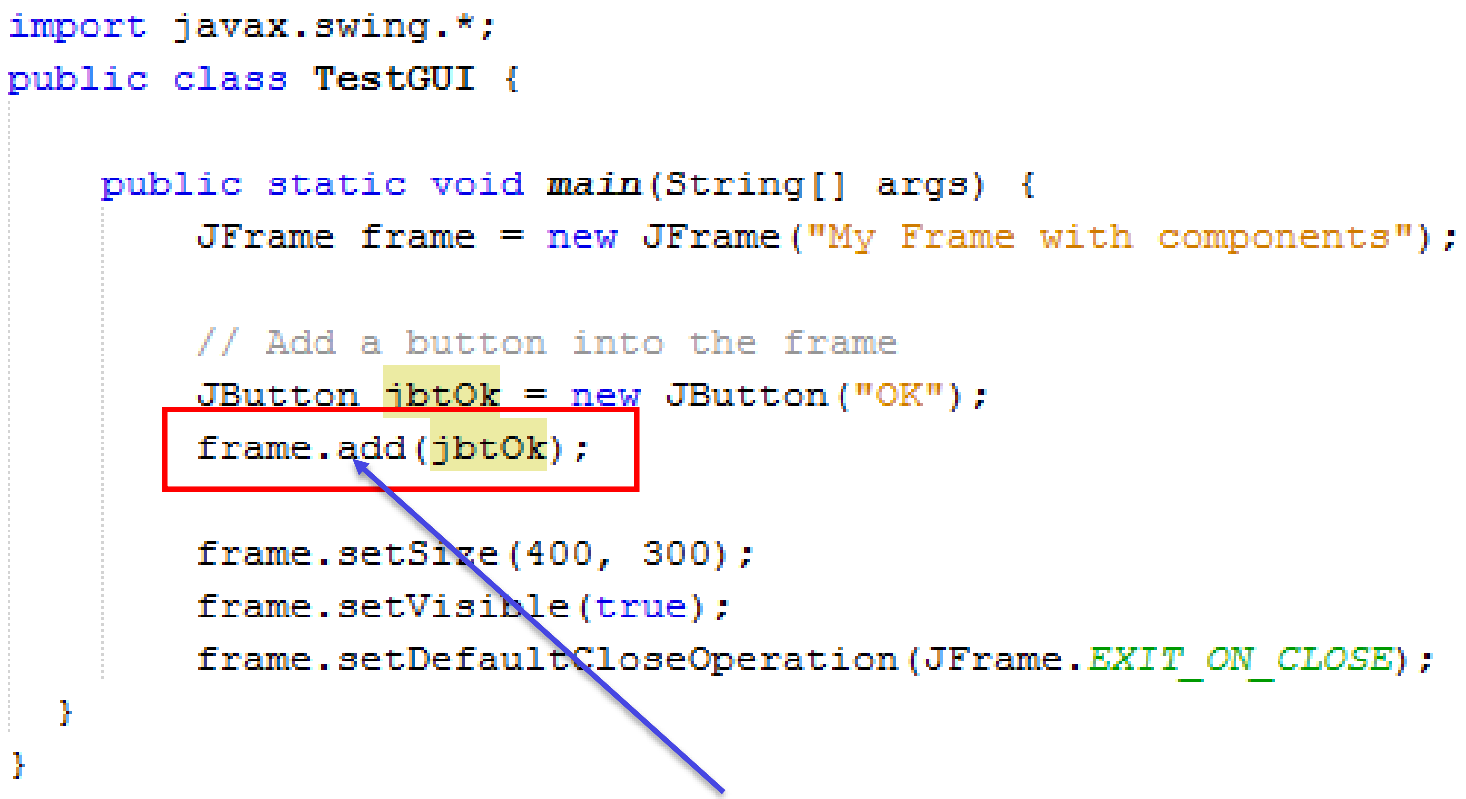
```
import javax.swing.*;
public class TestGUI {

    public static void main(String[] args) {
        JFrame frame = new JFrame("My Frame with components");

        // Add a button into the frame
        JButton jbtOk = new JButton("OK");
        frame.add(jbtOk);

        frame.setSize(400, 300);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Declare a reference variable for a button object.

Instantiate a button

UNIVERSITY OF WESTMINSTER

Adding component into a Frame



```
import javax.swing.*;
public class TestGUI {

    public static void main(String[] args) {
        JFrame frame = new JFrame("My Frame with components");

        // Add a button into the frame
        JButton jbtOk = new JButton("OK");
        frame.add(jbtOk);

        frame.setSize(400, 300);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Add the button to the content pane of the frame.

UNIVERSITY OF WESTMINSTER

Layout Managers

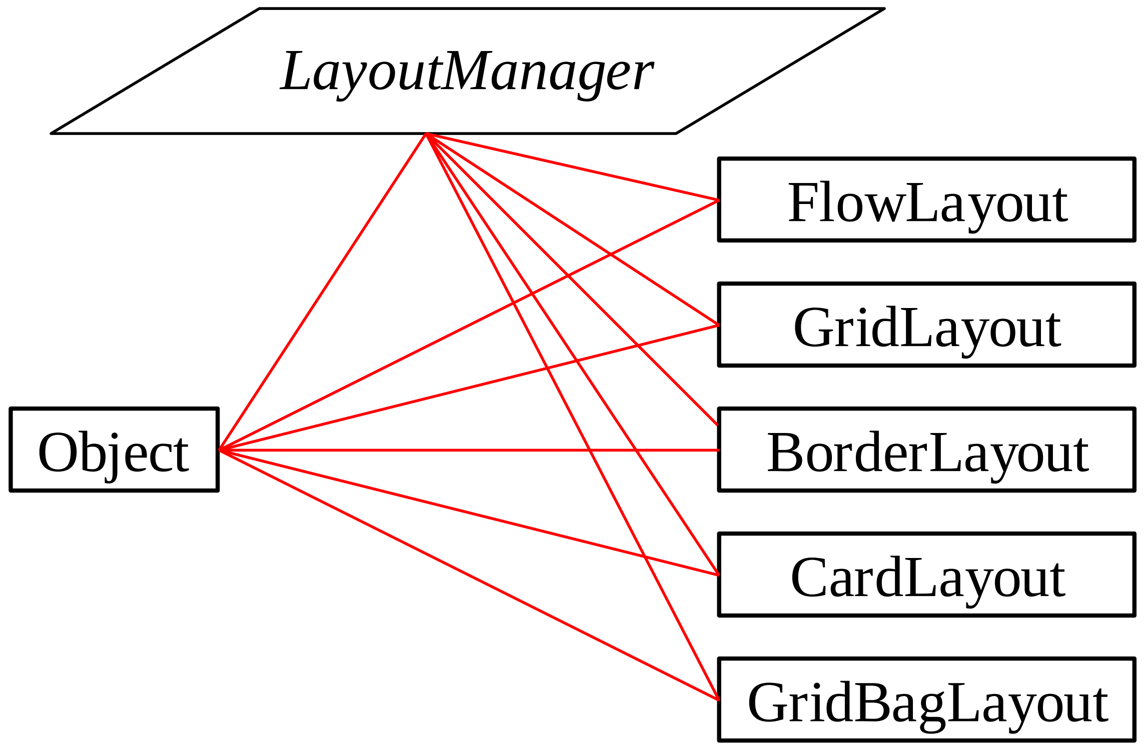
- The UI components are placed in containers. Each container has a layout manager to arrange the UI components within the container.

- Advantage: resizing the container (frame) will not occlude or distort the view of the components.

- Main layout managers:
  - FlowLayout, GridLayout, BorderLayout, CardLayout, and GridBagLayout

UNIVERSITY OF WESTMINSTER

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

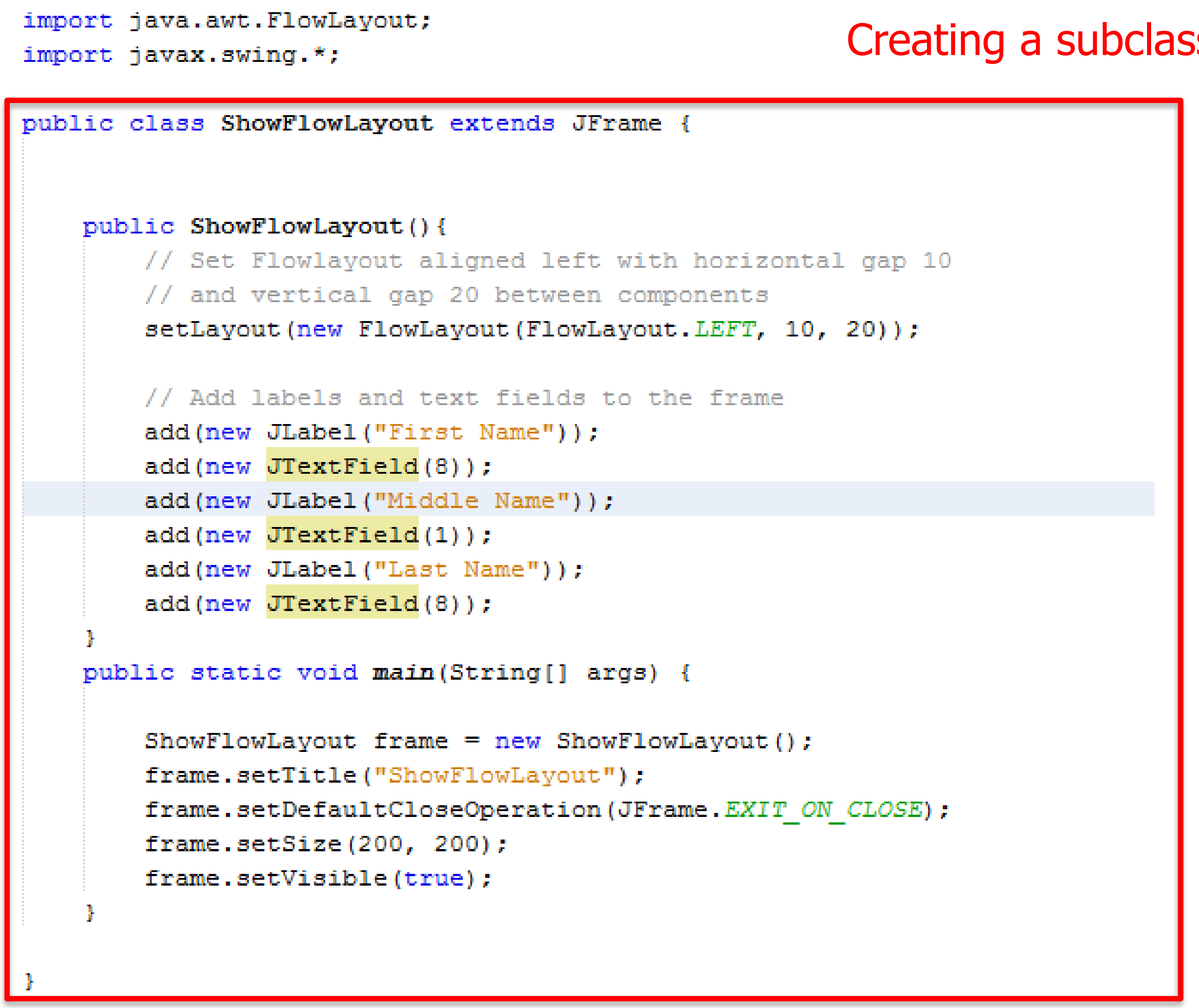
```
setLayout(new FlowLayout(FlowLayout.LEFT))
```

OR

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

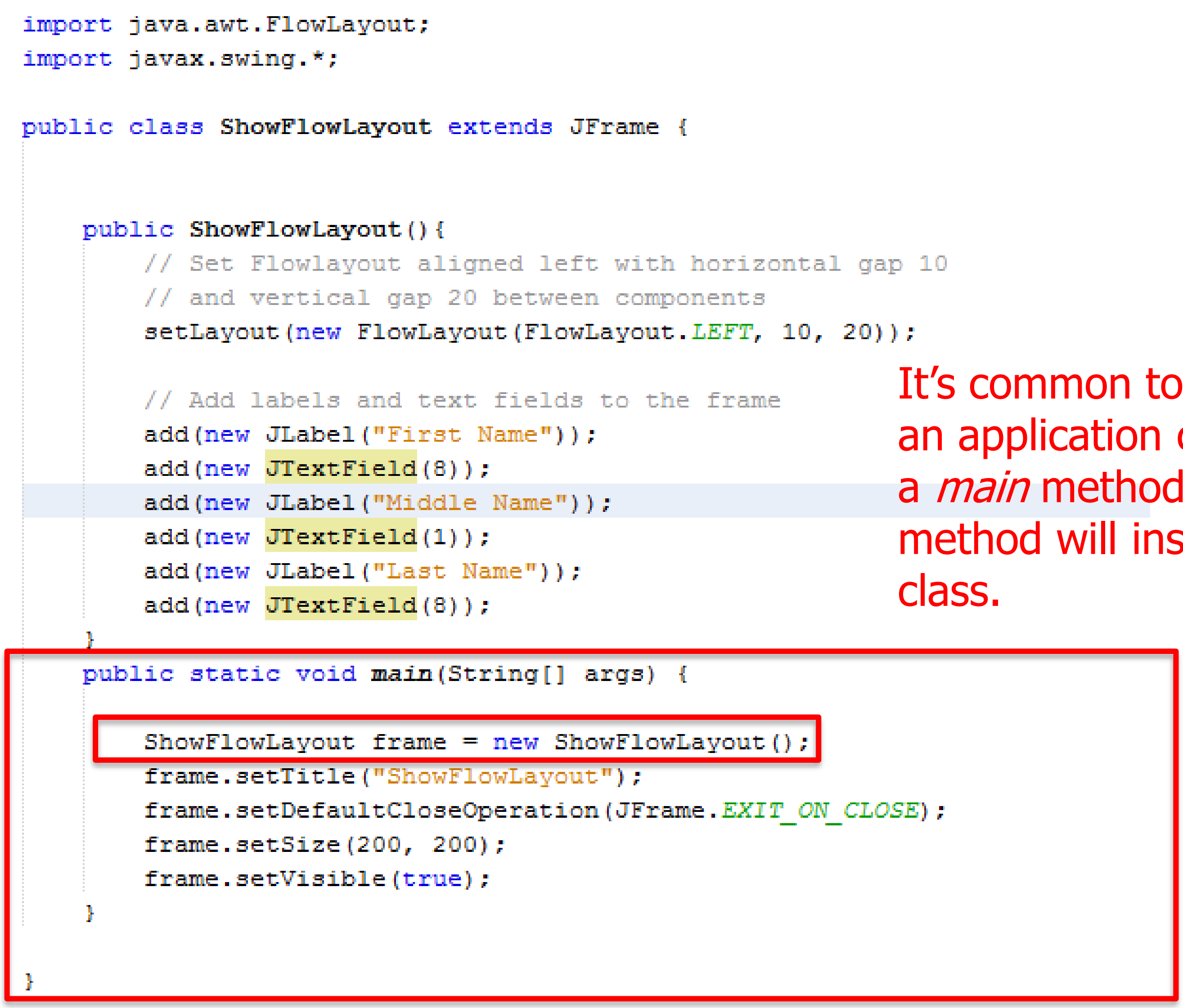
Example – FlowLayout

Creating a subclass of JFrame



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

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("Middle Name"));
        add(new JTextField(1));
        add(new JLabel("Last Name"));
        add(new JTextField(8));
    }
    public static void main(String[] args) {

        ShowFlowLayout frame = new ShowFlowLayout();
        frame.setTitle("ShowFlowLayout");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(200, 200);
        frame.setVisible(true);
    }

}
```

Example – FlowLayout

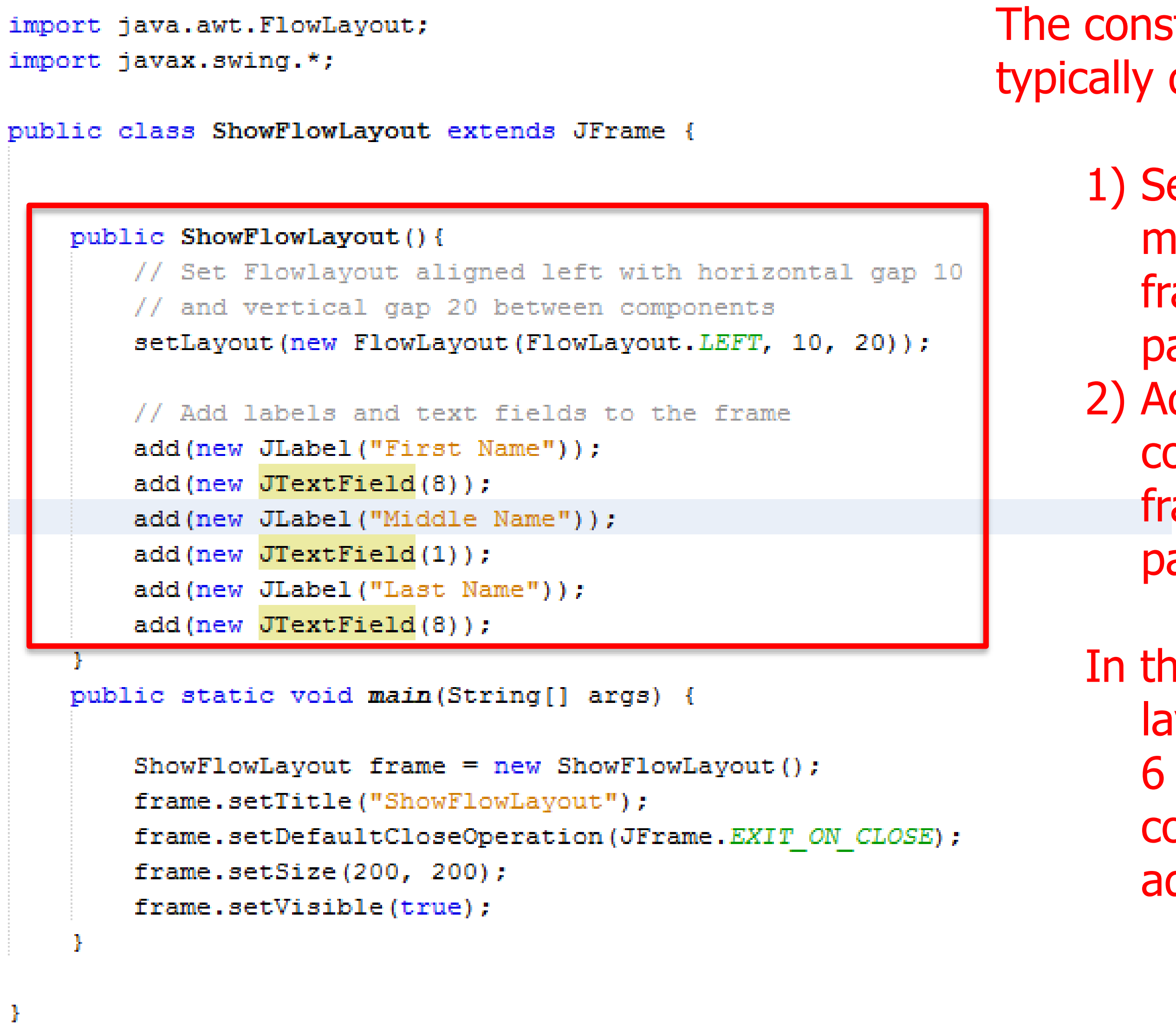


```
import java.awt.FlowLayout;
import javax.swing.*;

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("Middle Name"));
        add(new JTextField(1));
        add(new JLabel("Last Name"));
        add(new JTextField(8));
    }
    public static void main(String[] args) {

        ShowFlowLayout frame = new ShowFlowLayout();
        frame.setTitle("ShowFlowLayout");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(200, 200);
        frame.setVisible(true);
    }


}
```

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

UNIVERSITY OF WESTMINSTER

Example – FlowLayout



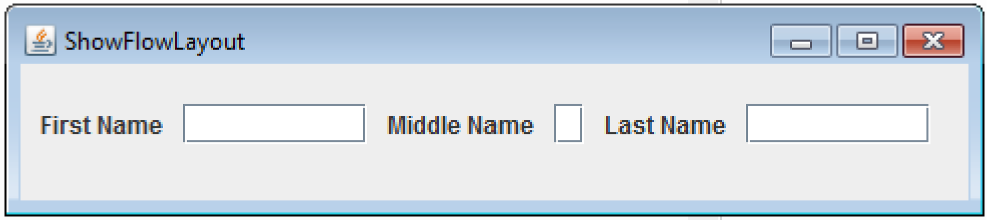
```
import java.awt.FlowLayout;
import javax.swing.*;

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("Middle Name"));
        add(new JTextField(1));
        add(new JLabel("Last Name"));
        add(new JTextField(8));
    }
    public static void main(String[] args) {

        ShowFlowLayout frame = new ShowFlowLayout();
        frame.setTitle("ShowFlowLayout");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(200, 200);
        frame.setVisible(true);
    }

}
```

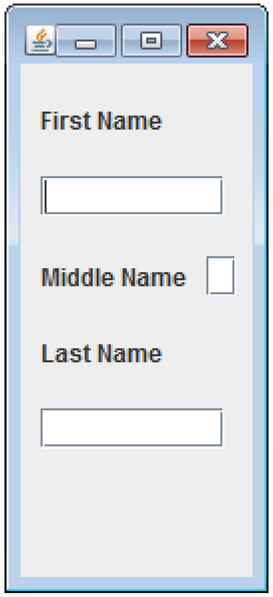
The constructor will typically do the following:

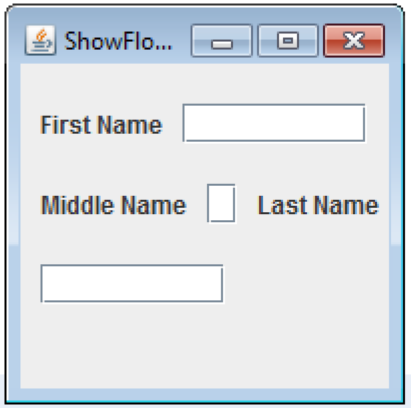
1) Set the layout manager for the frame's content pane
2) Add the components to the frame's content pane

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

What we get





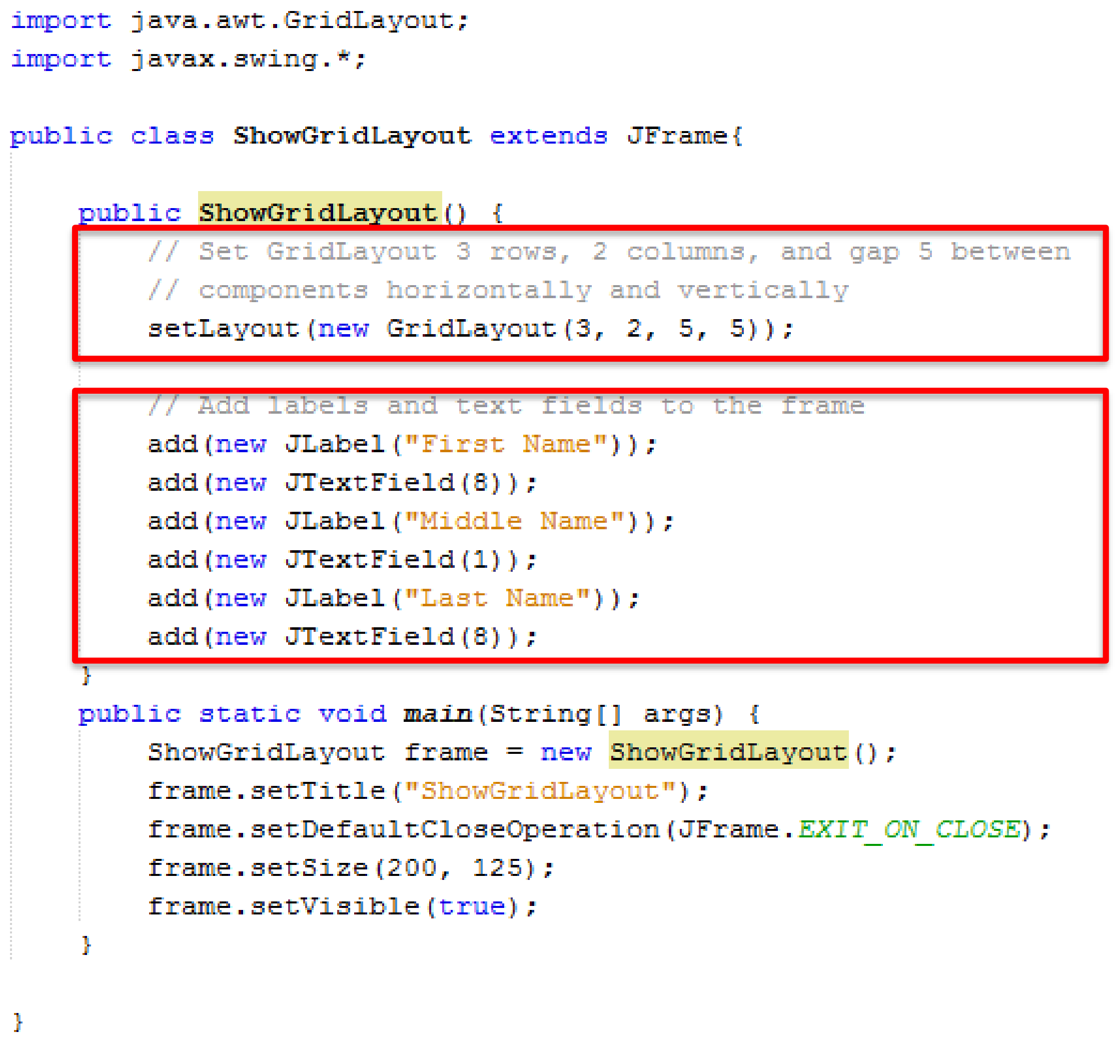


Resizing the frame causes the components to wrap around when necessary.

GridLayout

- Arranges components into rows and columns
- In Frame’s constructor:
  - `setLayout(new GridLayout(rows,columns))`
  - `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

Example - GridLayout

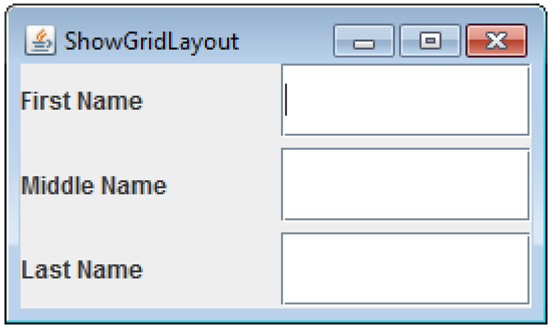


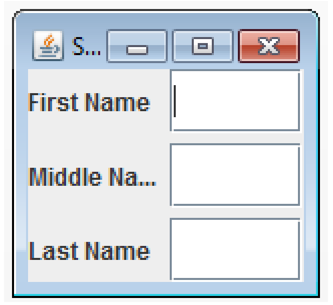
```
import java.awt.GridLayout;
import javax.swing.*;

public class ShowGridLayout extends JFrame{

    public ShowGridLayout() {
        // Set GridLayout 3 rows, 2 columns, and gap 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("Middle Name"));
        add(new JTextField(1));
        add(new JLabel("Last Name"));
        add(new JTextField(8));
    }
    public static void main(String[] args) {
        ShowGridLayout frame = new ShowGridLayout();
        frame.setTitle("ShowGridLayout");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(200, 125);
        frame.setVisible(true);
    }

}
```

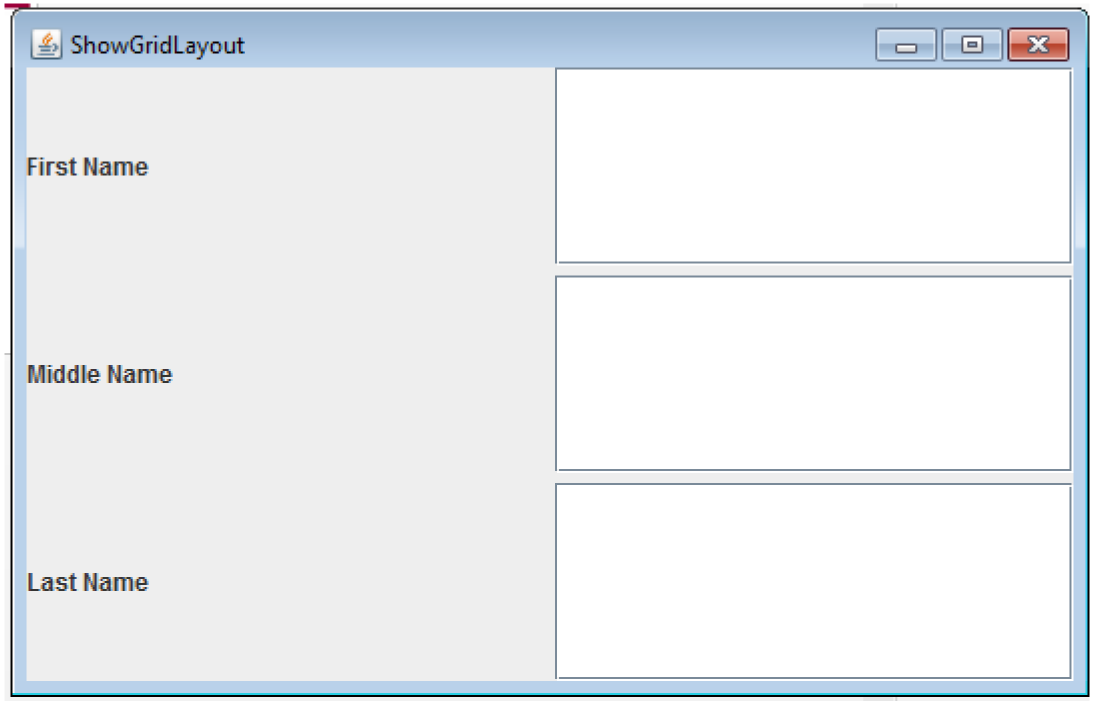
Setting the layout manager

Adding components

What we get







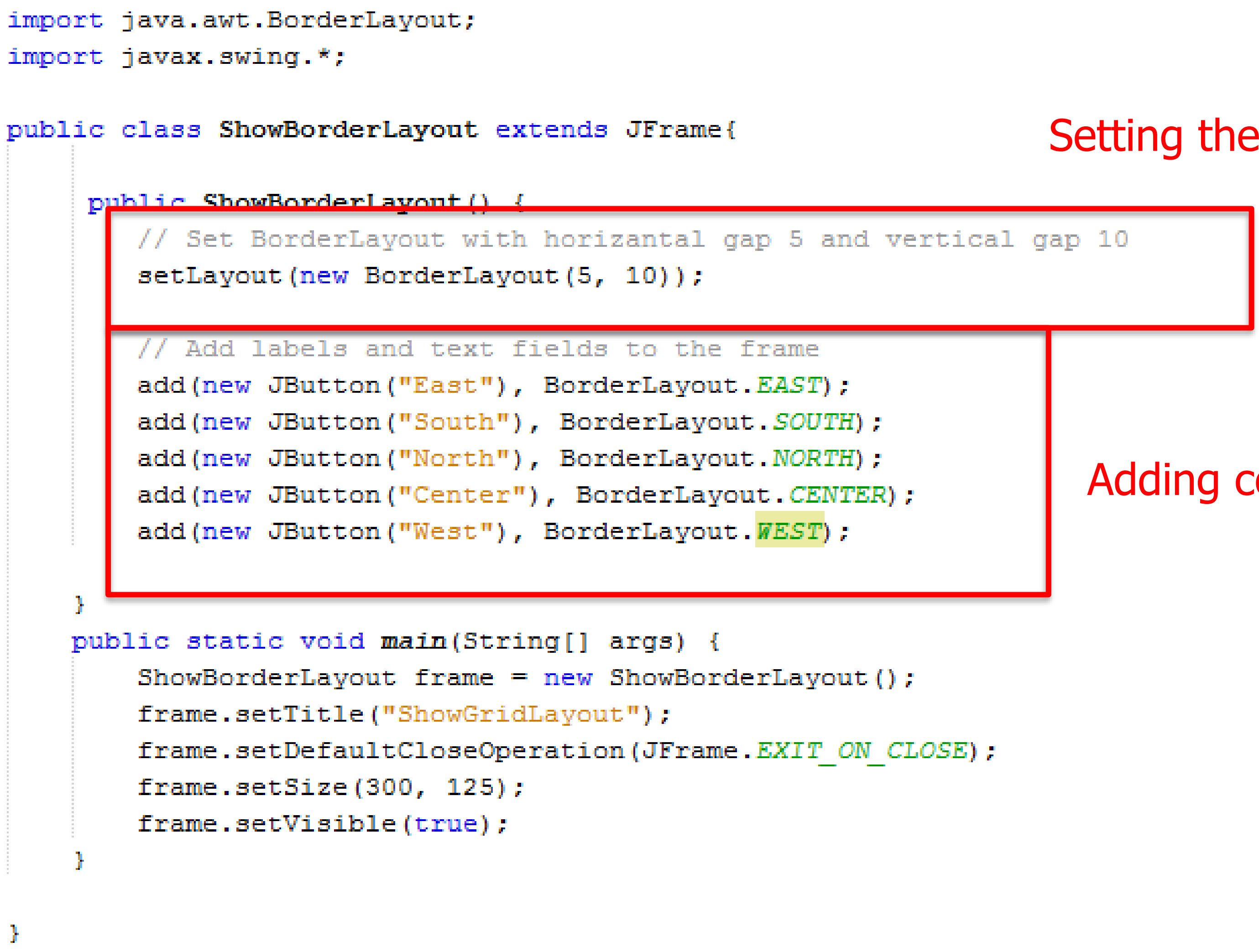
Resizing the frame causes the components to resize and maintain their same grid pattern.

BorderLayout

- Arranges components into five areas: North, South, East, West, and Center
- In the constructor:
  - `setLayout(new BorderLayout())`
  - `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, 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.

Example - BorderLayout

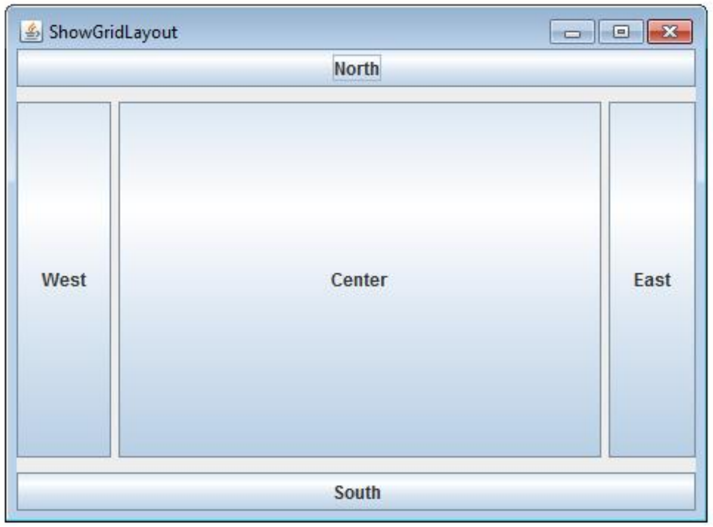


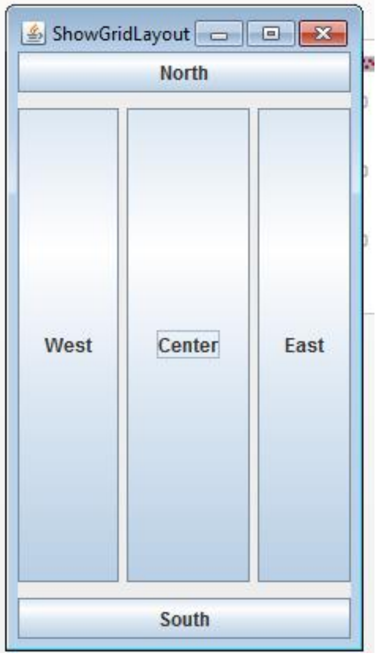
```java
import java.awt.BorderLayout;
import javax.swing.*;

public class ShowBorderLayout extends JFrame{

    public ShowBorderLayout() {
        // Set BorderLayout with horizantal gap 5 and vertical gap 10
        setLayout(new BorderLayout(5, 10));

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

    }
    public static void main(String[] args) {
        ShowBorderLayout frame = new ShowBorderLayout();
        frame.setTitle("ShowGridLayout");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 125);
        frame.setVisible(true);
    }

}
```

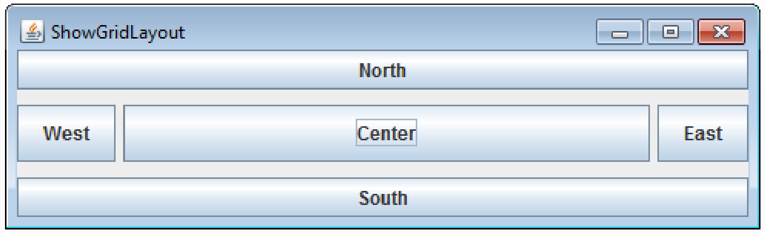
Setting the layout manager

Adding components

What we get







- Resizing the frame causes the components to resize and maintain their same regions.
- The CENTER region dominates the sizing.

Using Panels as “Sub-Containers”

- JPanel is a class of special components 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

UNIVERSITY OF WESTMINSTER

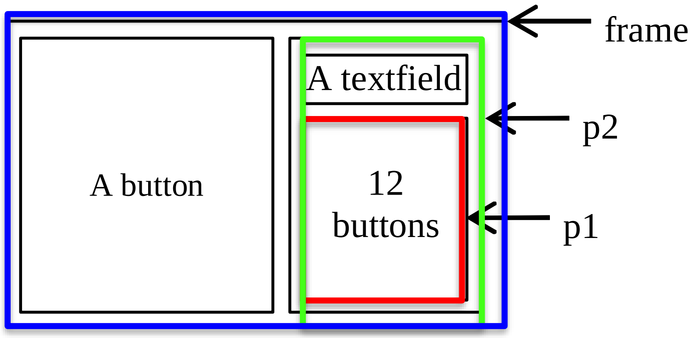
Creating a JPanel

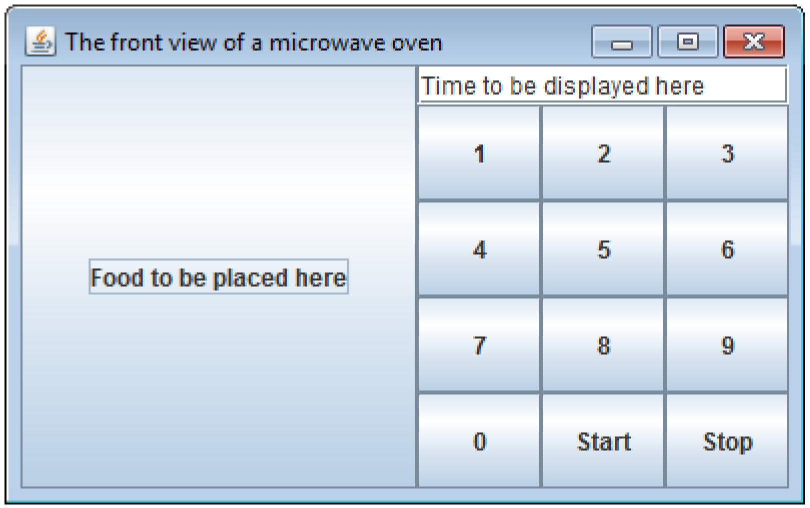
- You can use `new JPanel()` to create a panel with a default `FlowLayout` manager or `new JPanel(LayoutManager)` to create a panel with the specified layout manager.
- Use the `add(Component)` method to add a component to the panel. For example,

```
JPanel p = new JPanel();
p.add(new JButton("OK"));
```

Panels Example

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





Code

```java
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);
        }

    public static void main(String[] args) {
        TestPanels frame = new TestPanels();
        frame.setTitle("The front view of a microwave oven");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400,250);
        frame.setVisible(true);
    }
}
```

The Color Class

- You can set colors for GUI components by using the java.awt.Color class.
- Colors are made of red, green, and blue components, each of which is represented by a byte value that describes its intensity, ranging from 0 (darkest shade) to 255 (lightest shade). This is known as the *RGB model*.

```
Color c = new Color(r, g, b);
```

- `r`, `g`, and `b` specify a color by its red, green, and blue components.

Example:

```
Color c = new Color(228, 100, 255);
```

- Thirteen standard colors (black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow) are defined as constants in java.awt.Color.

Setting Colors

You can use the following methods to set the component's background and foreground colors:

```
setBackground(Color c)

setForeground(Color c)
```

Example:

```
jbt.setBackground(Color.yellow);

jbt.setForeground(Color.red);
```

The Font Class

- Font Names

  Standard font names that are supported in all platforms are: SansSerif, Serif, Monospaced, Dialog, or DialogInput.

- Font Style:

  Font.PLAIN (0), Font.BOLD (1), Font.ITALIC (2), and Font.BOLD + Font.ITALIC (3)

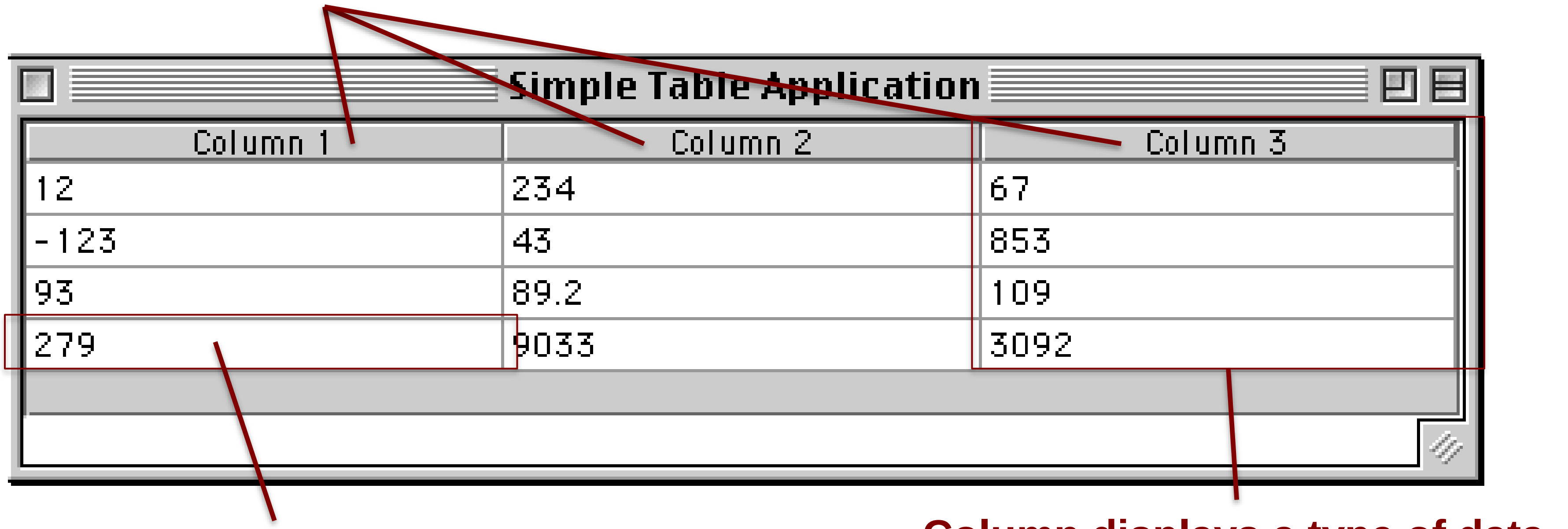
```
Font myFont = new Font(name, style, size);
```

Example:

```
Font myFont1 = new Font("SansSerif ", Font.BOLD, 16);
Font myFont2 = new Font("Serif", Font.BOLD+Font.ITALIC,
12);

JButton jbtOK = new JButton("OK“);
jbtOK.setFont(myFont1);
```

JTable

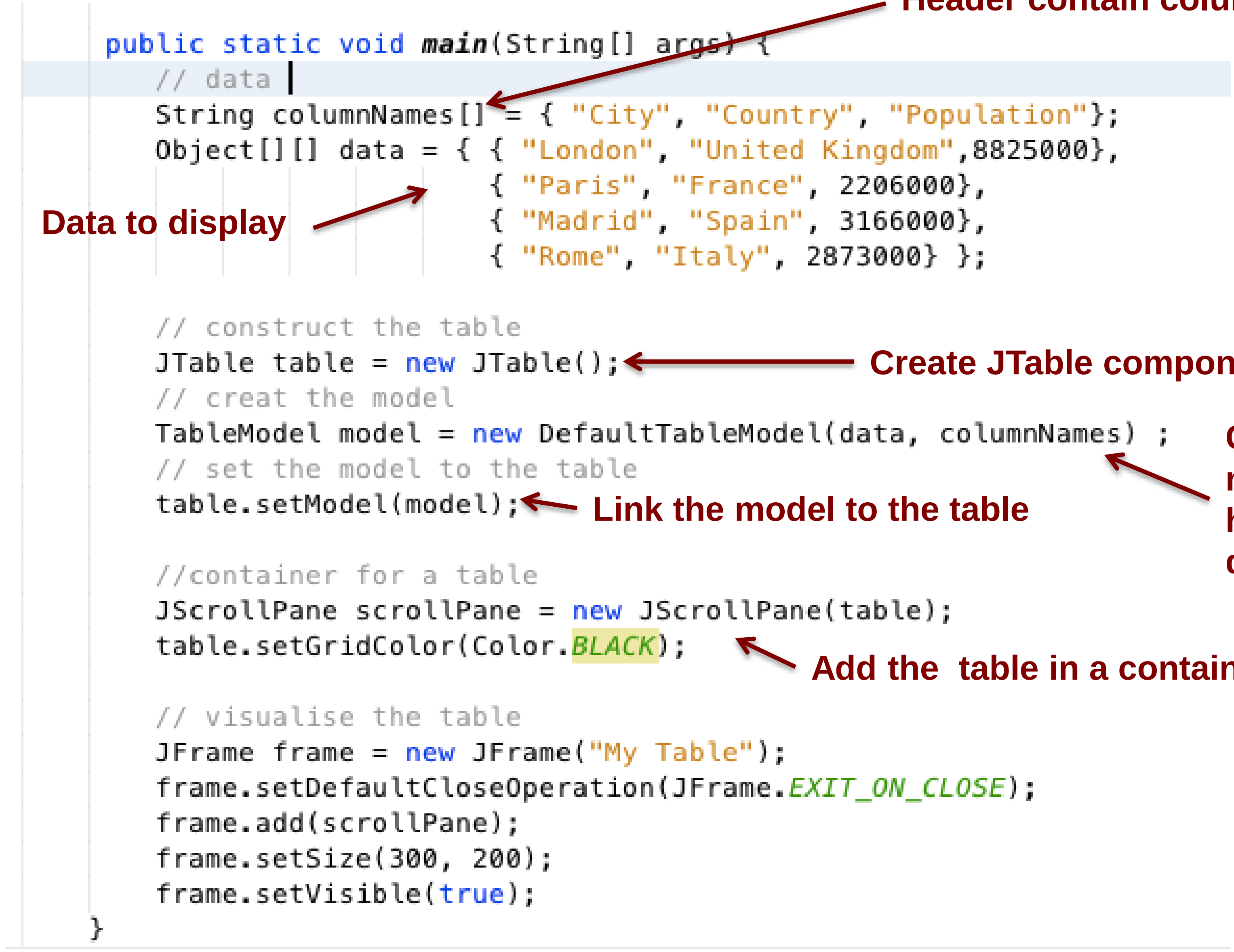
- Displays a grid of data consisting of rows and columns similar to a spreadsheet
- Does not contain or cache data but it is a simple view of the data


Header contain column labels
Simple Table Application
Cell containing data item
Column displays a type of data

Column 1	Column 2	Column 3
12	234	67
-123	43	853
93	89.2	109
279	9033	3092

UNIVERSITY OF WESTMINSTER

Create a simple table



```
public static void main(String[] args) {
    // data
    String columnNames[] = { "City", "Country", "Population"};
    Object[][] data = { { "London", "United Kingdom",8825000},
                        { "Paris", "France", 2206000},
                        { "Madrid", "Spain", 3166000},
                        { "Rome", "Italy", 2873000} };

    // construct the table
    JTable table = new JTable();
    // creat the model
    TableModel model = new DefaultTableModel(data, columnNames) ;
    // set the model to the table
    table.setModel(model);

    //container for a table
    JScrollPane scrollPane = new JScrollPane(table);
    table.setGridColor(Color.BLACK);

    // visualise the table
    JFrame frame = new JFrame("My Table");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.add(scrollPane);
    frame.setSize(300, 200);
    frame.setVisible(true);
}
```

Header contain column labels

Data to display

Create JTable component

Create the model that holds the data

Link the model to the table

Add the table in a container

Model/View/Controller

- Swing architecture is rooted in the *model-view-controller* (*MVC*) design
- MVC architecture calls for a visual application to be broken up into three separate parts:
  - A *model* that represents the data for the application
  - The *view* that is the visual representation of that data
  - A *controller* that takes user input on the view and translates that to changes in the model.”

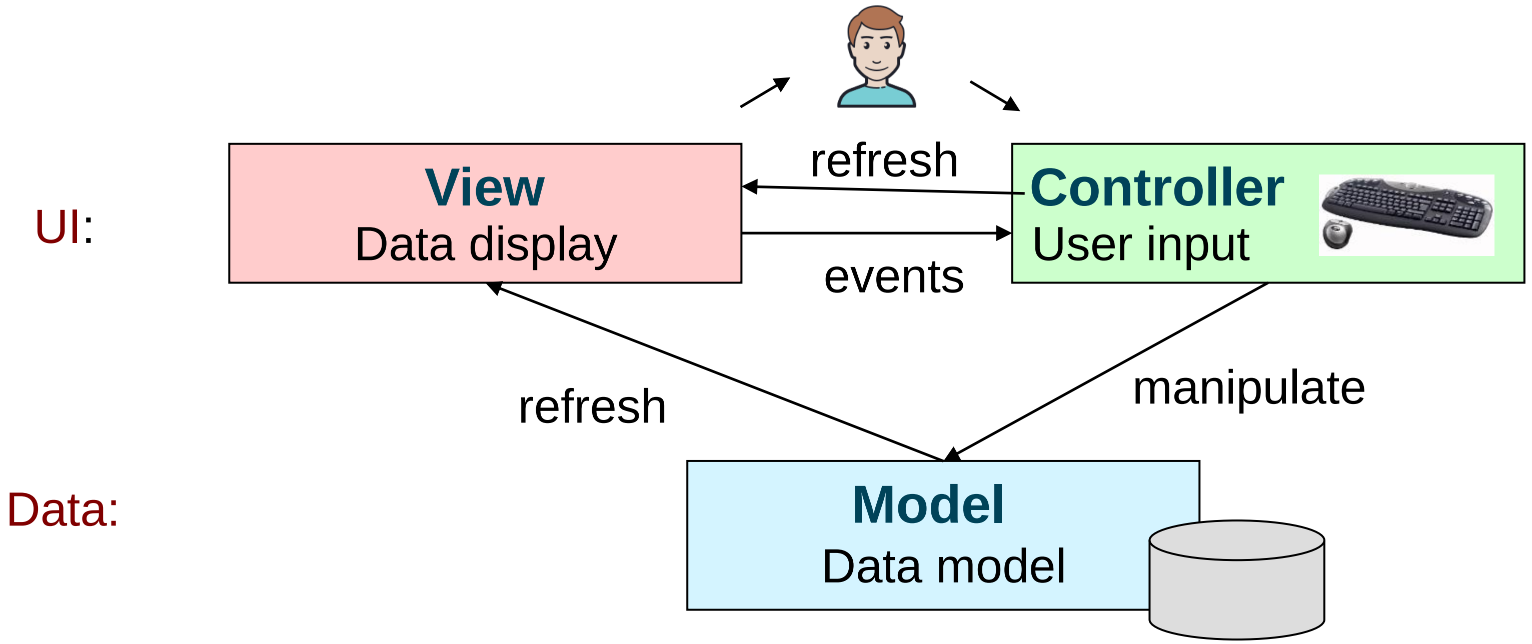


Table Model

- Each `JTable` has a `TableModel` object that holds the data shown by the table.
- **`DefaultTableModel`** is a default implementation of `TableModel`
  - By default it makes all the cells editable
- We can customize its behavior by subclassing
  - Or we can implement `TableModel` from scratch creating a class that **`extends AbstractTableModel`**

  ADVANTAGES:

  Data synchronisation easier: Suppose you have an ArrayList of some Java objects. You have to convert this data into Object[][] to populate the JTable. If you make changes to the JTable, you have to remember to make corresponding changes in the ArrayList of the underlying data

Create your own Table Model

- You need to override the following abstract methods:

➢ `public int getRowCount();`
➢ `public int getColumnCount();`
➢ `public Object getValueAt(int row, int column);`

```
public class BookTableModel extends AbstractTableModel {

    private String[] columnNames = {"Title","Author","Price"};
    private ArrayList<Book> myList;

    public BookTableModel(ArrayList<Book> bkList) {
        myList = bkList;
    }
    public int getColumnCount() {
        return columnNames.length;
    }
    public int getRowCount() {
        return myList.size();
    }
```

Constructor

Suppose we have Book objects, where the instance variables are Title, Author and Price.

Continue in the next slide ->

…continue

```
public Object getValueAt(int row, int col) {
        Object temp = null;
        if (col == 0) {
            temp = myList.get(row).getTitle();
        }
        else if (col == 1) {
            temp = myList.get(row).getAuthor();
        }
        else if (col == 2) {
            temp = new Double(myList.get(row).getPrice());
        }
        return temp;
    }
    // needed to show column names in JTable
    public String getColumnName(int col) {
        return columnNames[col];
    }
    public Class getColumnClass(int col) {
        if (col == 2) {
            return Double.class;
        }
        else {
            return String.class;
        }
    }
}
```

Using your own model:

```
....
ArrayList<Book> myList = new ArrayList<Book>();

// ... Here some code to fill the arrey list with Book objects

BookTableModel tableModel = new BookTableModel(myList);

Jtable table = new JTable(tableModel);

// add to the container and continue the code
JScrollPane scrollPane = new JScrollPane(table);

...
```

EVENT HANDLING

What is an Event

- GUI components communicate with the rest of the applications through events.
- An event is an object that describes a state changes in a source
- An event occurs every time the user
  - Types a character or
  - Pushes a mouse button
- The source of an event is the component that causes that event to occur.
- The listener of an event is an object that receives the event and processes it appropriately.
- That object must:
  - Implement the appropriate interface
  - Be registered as an *event listener* on the appropriate *event source*.

UNIVERSITY OF WESTMINSTER

Event Handling

- GUIs are event-driven
  - A user interaction creates an event
    - Common events are clicking a button, typing in a text field, selecting an item from a menu, closing and window and moving the mouse
  - The event causes a call to a method called an event handler

- Class `java.awt.AWTEvent`

Delegation Event Handling Model

- Three parts
  - Event source
    - GUI component which user interacts with
  - Event object
    - Encapsulates information about event that occurred
  - Event listener
    - Receives event object when notified, then responds
- Programmer must perform two tasks
  1. **Register event listener for event source**
  2. **Implement event-handling method (event handler)**

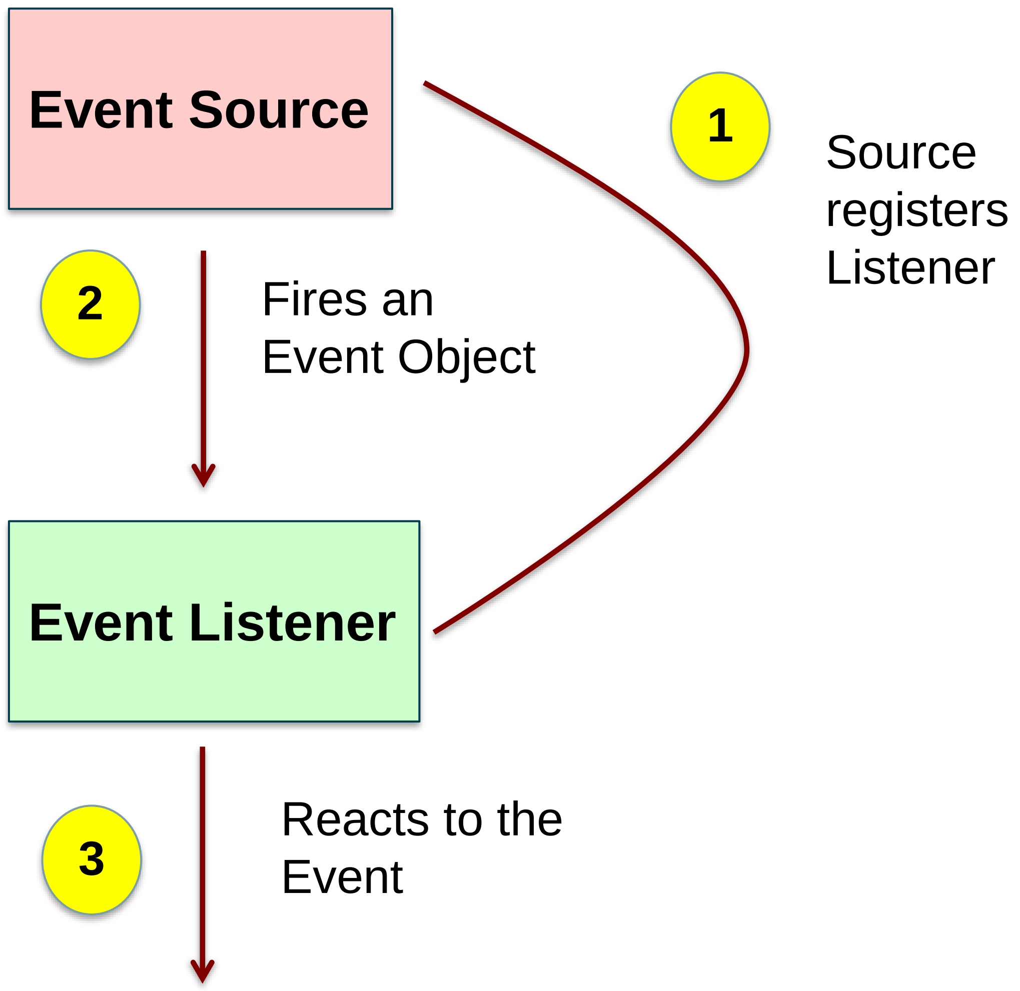
What is an Event Listener?

- An event listener is an object
  - It "listens" for events from a specific GUI component (itself an object)
- When an event is generated by the GUI component a method in the listener object is invoked to respond to the event

What if there is no event listener?

- When there is no event listener for an event
  - A program can ignore events
  - If there is no listener for an event, the event is just ignored

Recap the Event Handling process



UNIVERSITY OF WESTMINSTER

What do we need to implement an Event Handler?

- It just looks for 3 pieces of code!

- **First**: **Implement an event handler Class**. In the declaration of the class, one line of code must specify that the class **implements** either a **listener interface** or extends a class that implements a listener interface.

```
public class DemoEventHandlerClass implements
ActionListener {
```

What do we need to implement an Event Handler?

- **Second**: Register the listener to the component. There will be a **line of code *which registers an instance* of the event handler class** as a listener of one or more components because, as mentioned earlier, the object must be registered as an event listener.

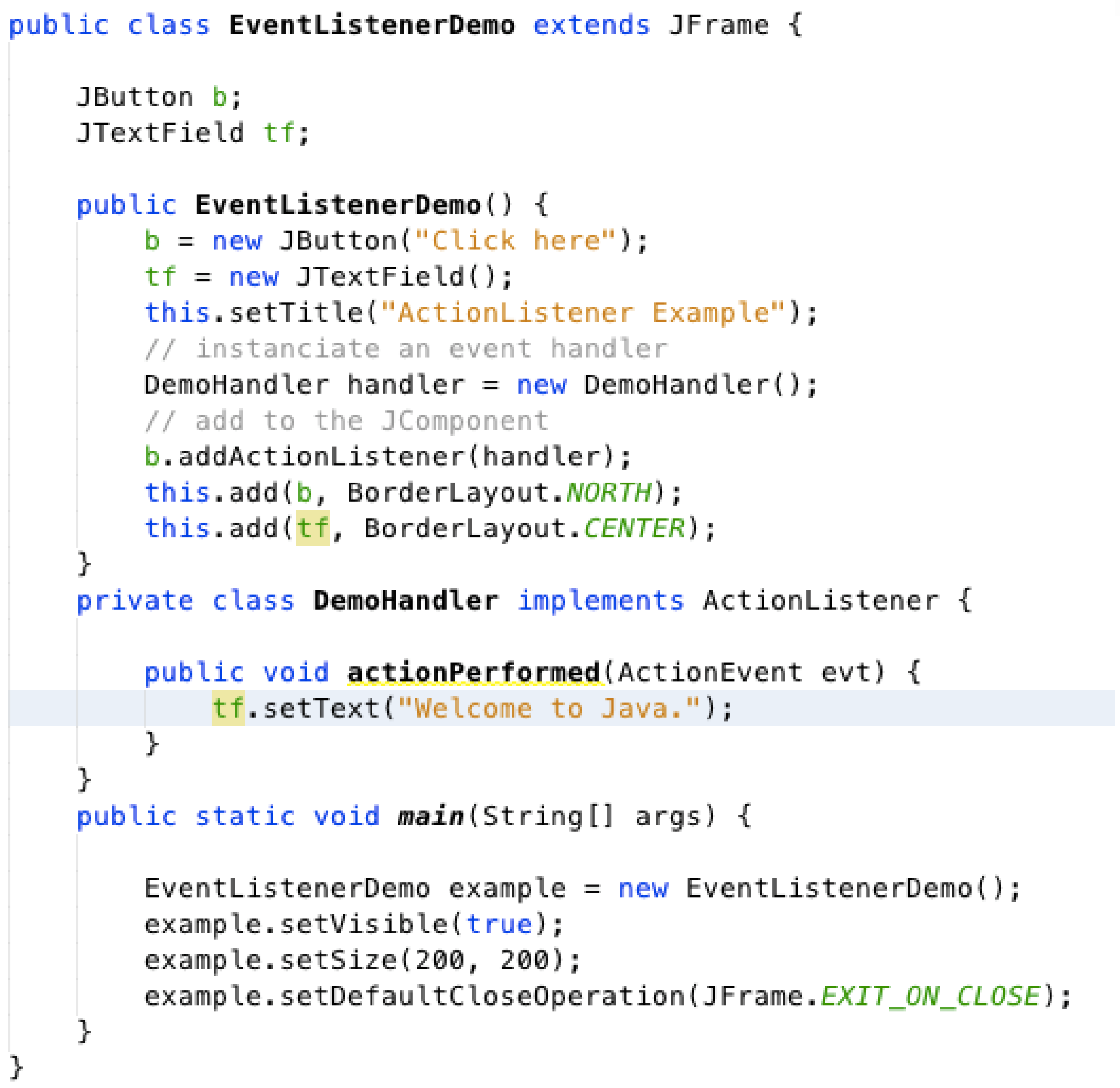
```
anyComponent.addActionListener(instance of
DemoEventHandlerClass);
```

What do we need to implement an Event Handler?

- **Third**: the event handler must have a piece of code that implements the methods in the listener interface.

```
public void actionPerformed(ActionEvent e) {
    ...//code that reacts to the action...
  }
```

Example



```java
public class EventListenerDemo extends JFrame {

    JButton b;
    JTextField tf;

    public EventListenerDemo() {
        b = new JButton("Click here");
        tf = new JTextField();
        this.setTitle("ActionListener Example");
        // instanciate an event handler
        DemoHandler handler = new DemoHandler();
        // add to the JComponent
        b.addActionListener(handler);
        this.add(b, BorderLayout.NORTH);
        this.add(tf, BorderLayout.CENTER);
    }
    private class DemoHandler implements ActionListener {

        public void actionPerformed(ActionEvent evt) {
            tf.setText("Welcome to Java.");
        }
    }
    public static void main(String[] args) {

        EventListenerDemo example = new EventListenerDemo();
        example.setVisible(true);
        example.setSize(200, 200);
        example.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

References

- http://java.sun.com/docs/books/tutorial/uiswing/events/index.html
- http://java.sun.com/docs/books/tutorial/uiswing/learn/example2.html#handlingEvents