# Graphical User Interfaces (GUI)

Hirra Anwar

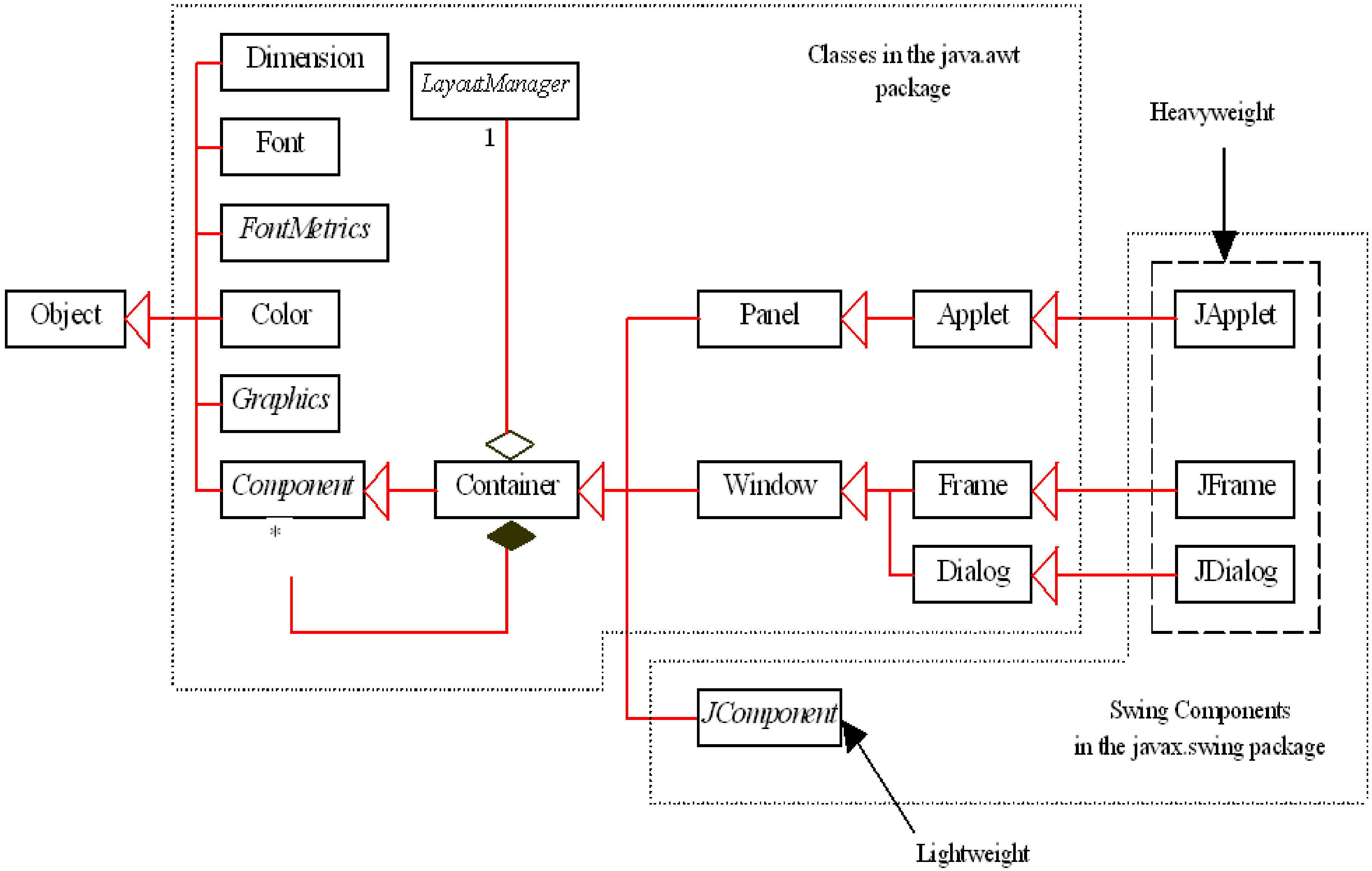
## Introduction

- Abstract Window Toolkit (AWT)
  - The java.awt package was the basic APIs that are used to create a GUI in java.
  - AWT components are heavyweight so many of the classes it defines have been superseded in Java 2 by javax.swing
- Swing (JFC)
  - Designed and developed using Java
  - Lightweight components with more features
  - However the Swing classes are generally derived from, and depend on, fundamental classes within java.awt, so these cant be ignored.

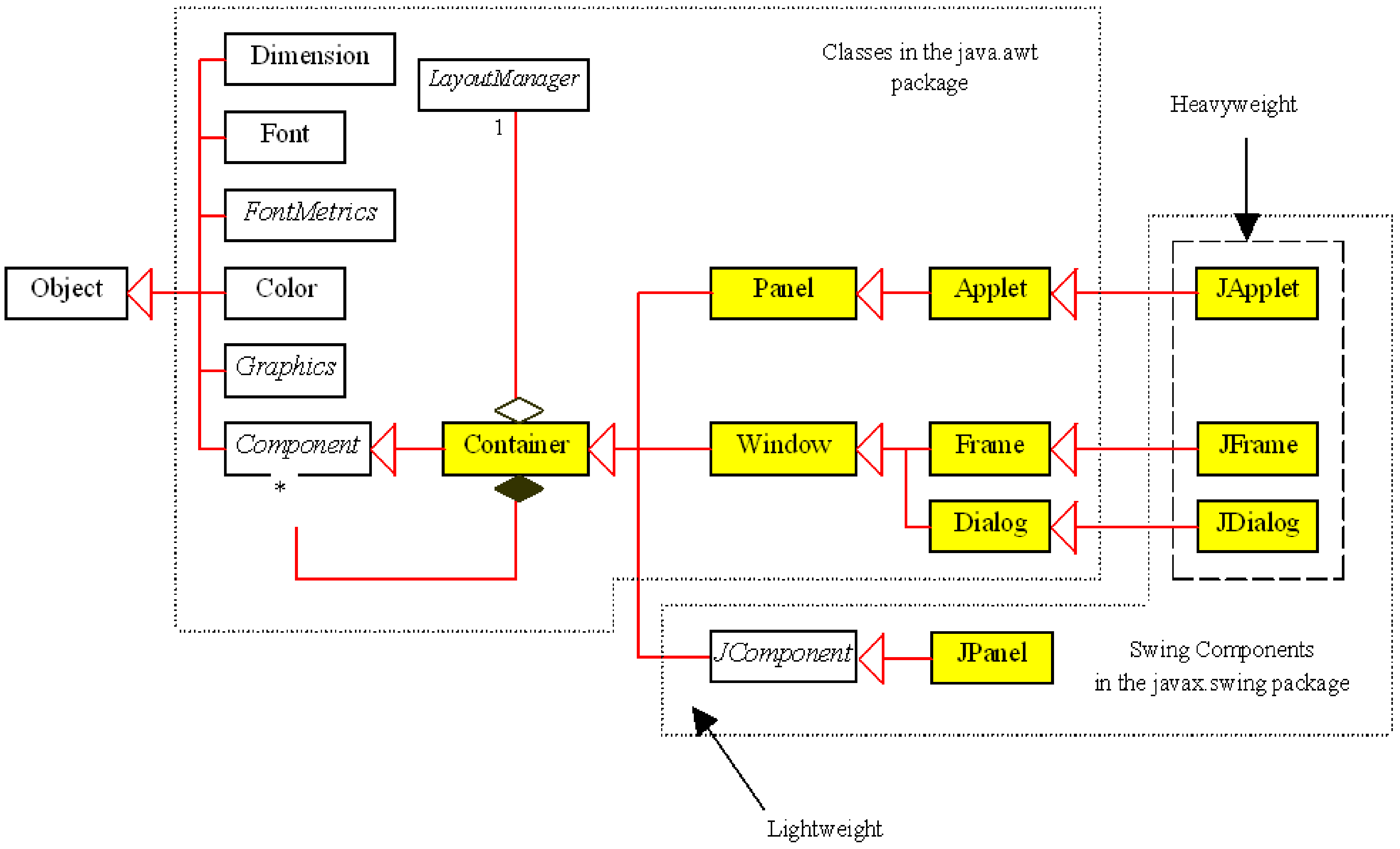
## Introduction : JFC

- Part of the Java Foundation Classes (JFC)
- Provides a rich set of GUI components
- Used to create a Java program with a graphical user interface (GUI)
- table controls, list controls, tree controls, buttons, and labels, and so on…

## GUI Class Hierarchy (Swing)



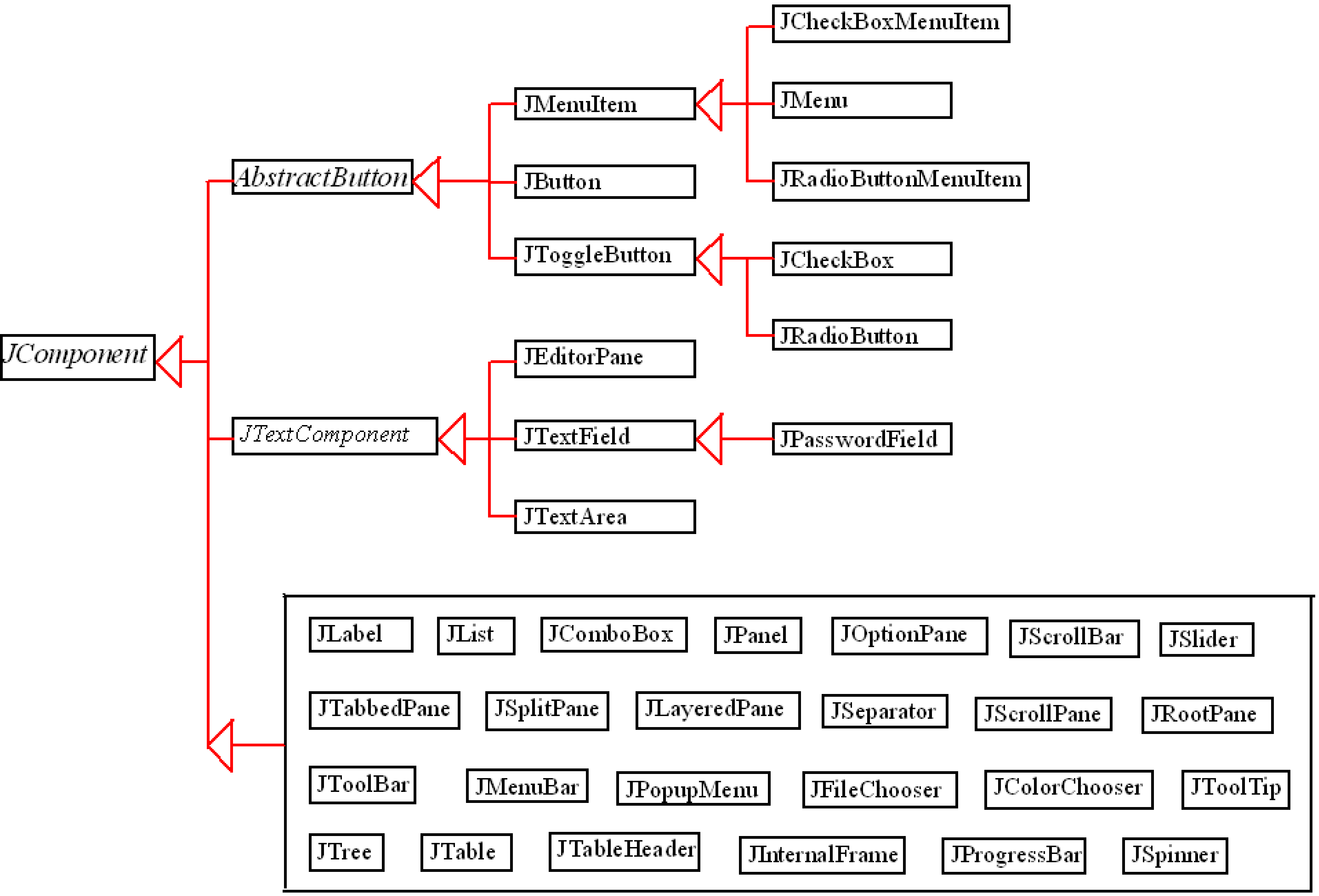
# Container Classes



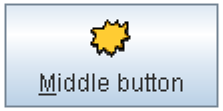
## Component

- A component represents a graphical entity of one kind or another that can be displayed on screen
- All container and visual components inherit from java.awt.Component
- Key Classes: Window, JFrame, JDialog, **JApplet** etc

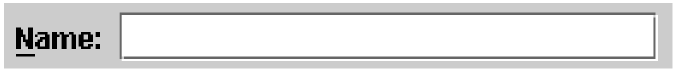
## Swing GUI Components



# Components

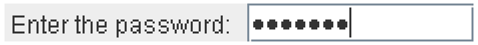


JButton

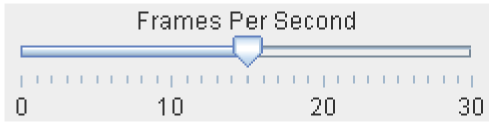


JLabel

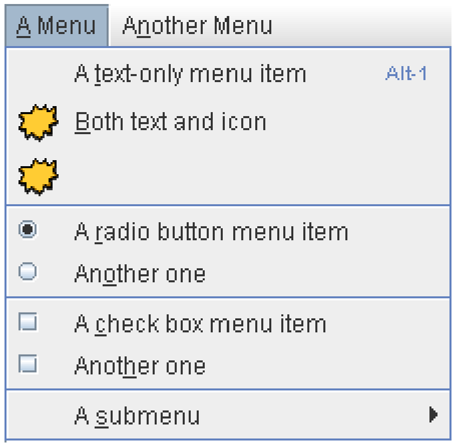
JTextField



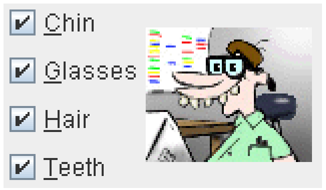
JPasswordField



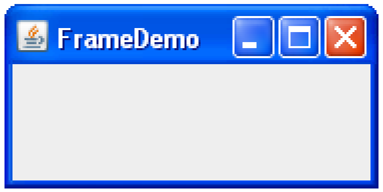
JSlider



JMenu

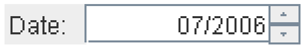


JCheckBox

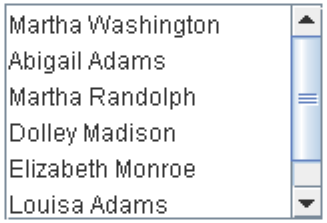


JFrame

JApplet

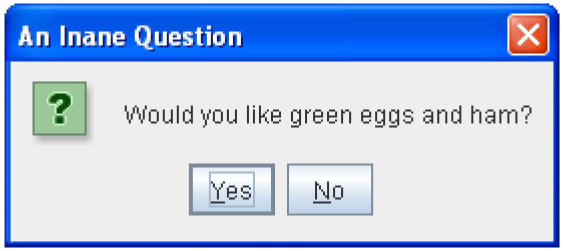


JSpinner



Martha Washington
Abigail Adams
Martha Randolph
Dolley Madison
Elizabeth Monroe
Louisa Adams

JList



Would you like green eggs and ham?

JDialog

# GUI Application

```
public class MyHelloGUI extends JFrame
{
      private JButton btn=new JButton("Click me");
      private JTextField jtf=new JTextField("value");
      private JLabel lab= new JLabel("Enter Value:");

      public MyHelloGUI()
      {
         Container con = this.getContentPane();
         this .setTitle("Title");
         con.setLayout(new FlowLayout());
         con.add(lab);
         con.add(jtf);
         con.add(btn);
         con.setSize(300,200);
   btn.addActionListener(new   MyActionListener());
        con.show();
        this.pack();
       this.setVisible(true);
   }
```

```
 public static void main(String[] args)
 {
      MyHelloGUI myHelloGUI = new MyHelloGUI();
 }
 class MyActionListener implements ActionListener
 {
   public void actionPerformed (ActionEvent e)
   {
         if(e.getSource() == btn)
         {
           JOptionPane.showMessageDialog(null,jtf.getText());
        }
     }
 }
}
```

- The Container class is an abstract class
- Most commonly used concrete subclasses are JApplet, JFrame, JDialog and Panel
- All these classes derived from the Container and can contain other objects of the classes derived from Component
- Swing provides **containers** such as
  - top level: frames, dialogs
  - intermediate level: panel, scroll pane, tabbed pane, ...
  - other Swing components: buttons, labels, ...

## GUI Application : Containers

- Descendants of the java.awt.Container class
- Use a layout manager to position and size the components contained in them.
- Components are added to a container using one of the various forms of its **add** method
  - Depending on which layout manager is used by the container

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

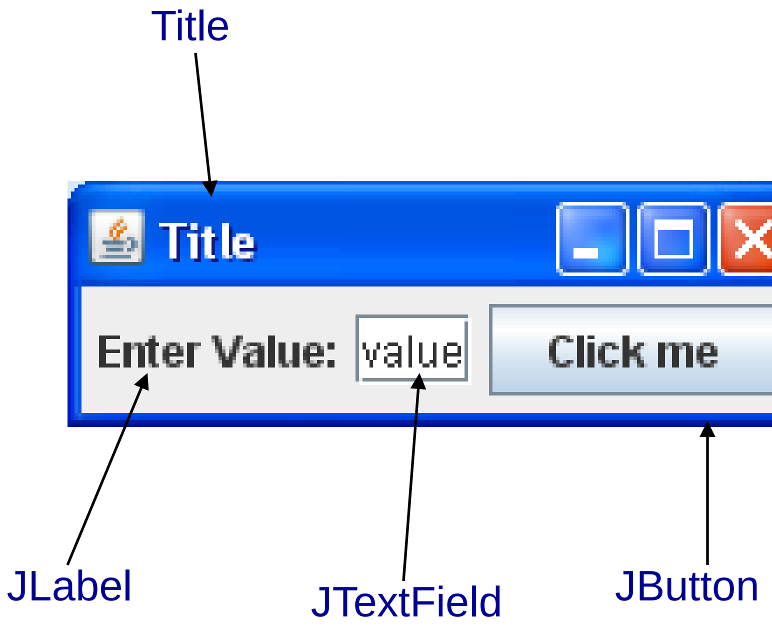
## GUI Application : Containers

- 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.
- Swing provides three top-level containers:
  - JFrame (Main window)
  - JDialog (Secondary window)
  - JApplet (An applet display area within a browser window)

## GUI Application : Components

```
// 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"});
```

# GUI Application



## Introduction

- Layout Managers
- JPanel

## Layout Managers

- Java’s layout managers provide a level of abstraction to automatically map your user interface on all window systems.

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

- Layout managers are set in containers using the setLayout(LayoutManager) method in a container.

## Layout Managers

- Java GUI reside in frames
- There are several layout manager classes in the AWT and swing
- java.awt.LayoutManager is an interface not a class

## Layout Managers

- The Java platform supplies five commonly used layout managers:
  – FlowLayout
  – BorderLayout
  – GridLayout
  – BoxLayout
  – NULL Layout

## Flow Layout Manager

- It is the default layout manager for panels
- It always arranges the components in horizontal rows while honoring each components preferred size
- Within every row the components are evenly spaced and the cluster of components is centered
- To change this default behavior , you can use setLayout( )
  - setLayout needs a parameter of type object of Layout Manager
  - setLayout(new FlowLayout(FlowLayout.RIGHT))
- By default it leaves a gap of 5 pixels between components in both horizontal and vertical directions

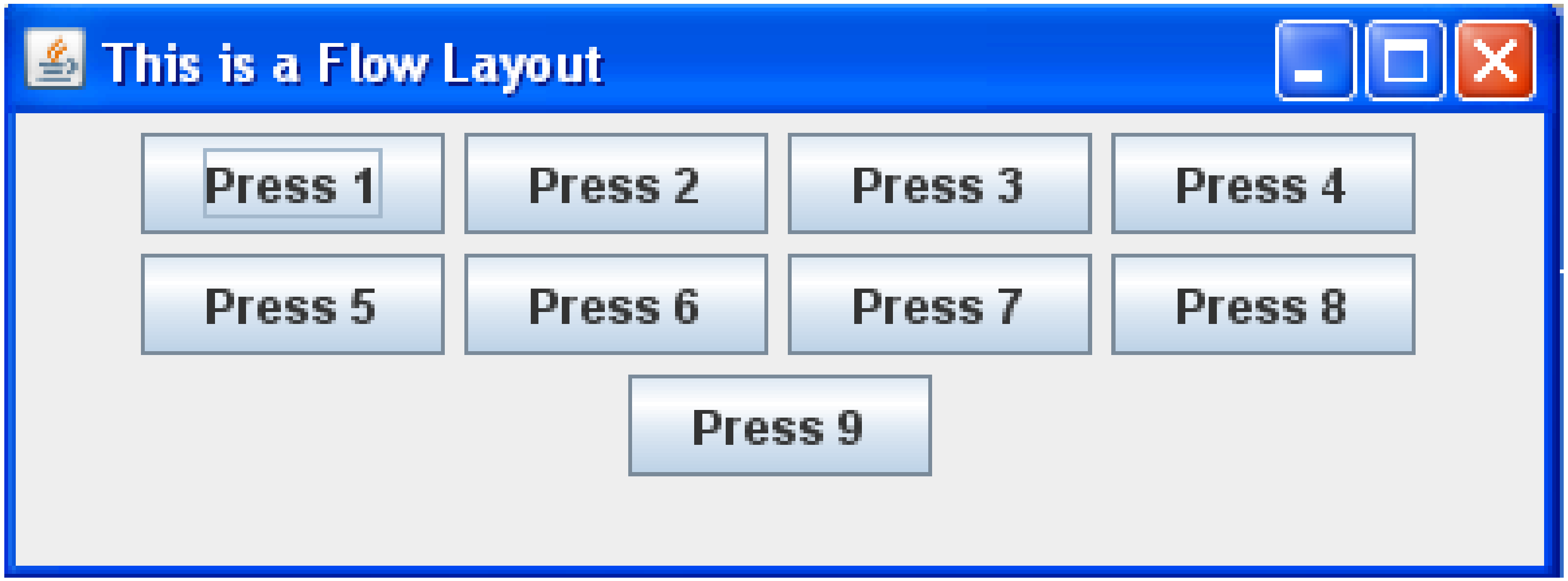
## FlowLayout Example

```
import java.awt.Container;
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;

public class TryFlowLayout
{
  // The window object
  static JFrame aWindow = new JFrame("This is a Flow Layout");
  public static void main(String[] args)
  {
    JButton btn[] = new JButton [9];
    int windowWidth = 400;                    // Window width in pixels
    int windowHeight = 150;                   // Window height in pixels

     aWindow.setBounds(100, 100,                 // Set position
                  windowWidth, windowHeight);     // and size
```

## FlowLayout Example

```
aWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   FlowLayout flow = new FlowLayout();           // Create a layout manager

   Container  content = aWindow.getContentPane();  // Get the content pane

   content.setLayout(flow);                        // Set the container layout mgr

      // Now add six button components
      for(int i = 1; i < 10; i++)
      {
    btn[i] = new JButton ("Press " + i);
        content.add(btn[i] );      // Add a Button to content pane
      }
      aWindow.setVisible(true);                   // Display the window
    }
  }
```

## FlowLayout Example



## The FlowLayout Class

The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

| java.awt.FlowLayout | |
|---|---|
| -alignment: int | The alignment of this layout manager (default: CENTER). |
| -hgap: int | The horizontal gap of this layout manager (default: 5 pixels). |
| -vgap: int | The vertical gap of this layout manager (default: 5 pixels). |
| +FlowLayout() | Creates a default FlowLayout manager. |
| +FlowLayout(alignment: int) | Creates a FlowLayout manager with a specified alignment. |
| +FlowLayout(alignment: int, hgap: int, vgap: int) | Creates a FlowLayout manager with a specified alignment, horizontal gap, and vertical gap. |

## Border Layout Manager

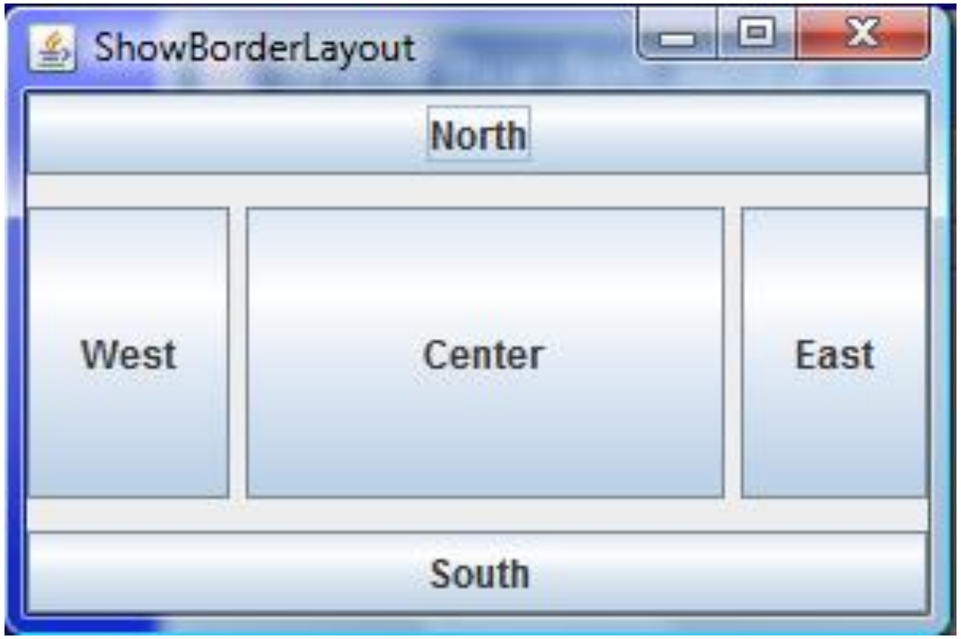
- is the default layout manager for frames
- It divides its territory in five regions, North, South, East, West and Center
- Each region can contain at the most one component, It may be empty though
- You can pass either a string like “North”, “Center” or you can use defined constants like BorderLayout.NORTH, BorderLayout.CENTER etc
- Constructors

```
p.setLayout(new BorderLayout()); // Default is no gaps
p.setLayout(new BorderLayout(hgap, vgap);
```

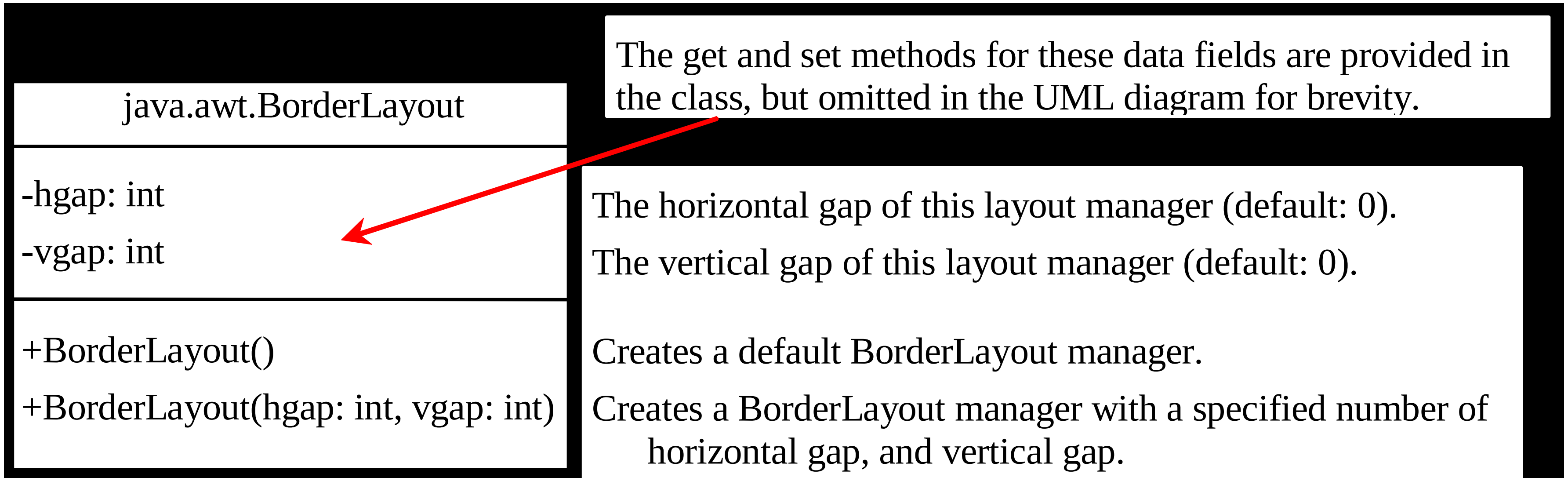
## Border Layout Manager Example

```
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);  }

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

## Border Layout Manager Example



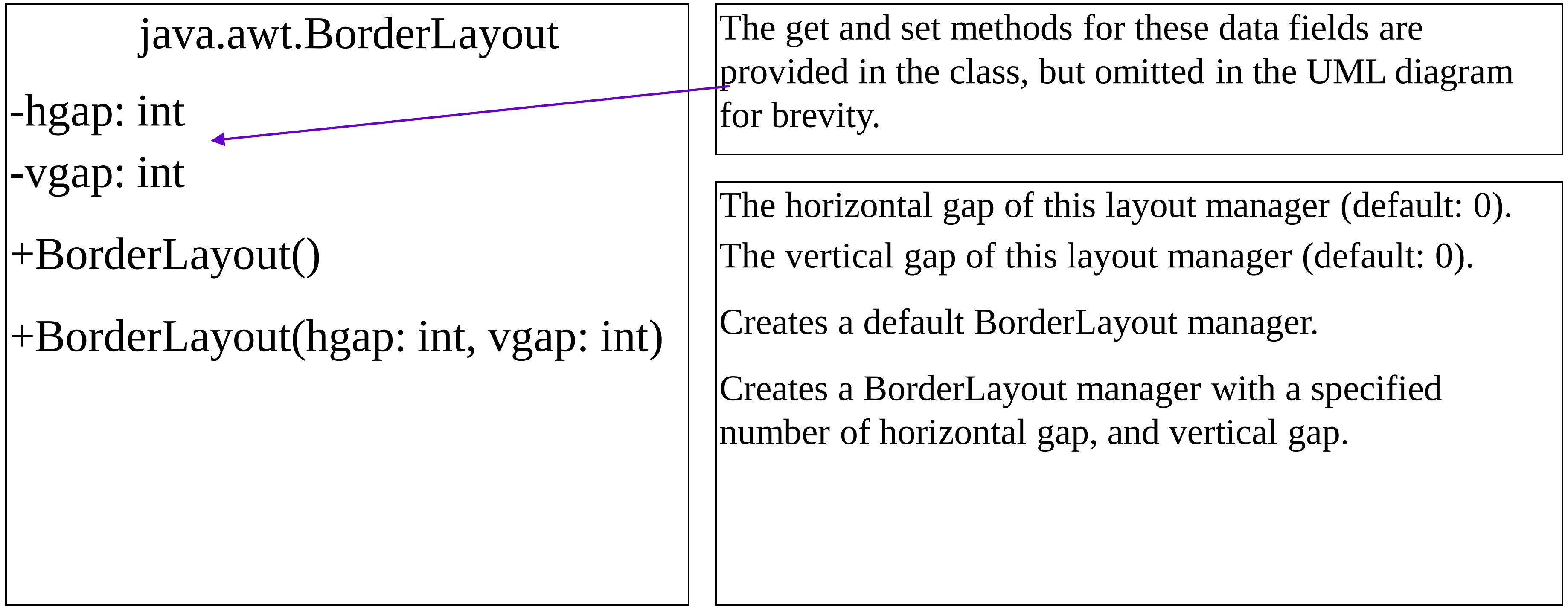
# The BorderLayout Class



The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

| java.awt.BorderLayout | |
|---|---|
| -hgap: int | The horizontal gap of this layout manager (default: 0). |
| -vgap: int | The vertical gap of this layout manager (default: 0). |
| +BorderLayout() | Creates a default BorderLayout manager. |
| +BorderLayout(hgap: int, vgap: int) | Creates a BorderLayout manager with a specified number of horizontal gap, and vertical gap. |

# The BorderLayout Class



java.awt.BorderLayout

-hgap: int

-vgap: int

+BorderLayout()

+BorderLayout(hgap: int, vgap: int)

The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

The horizontal gap of this layout manager (default: 0).

The vertical gap of this layout manager (default: 0).

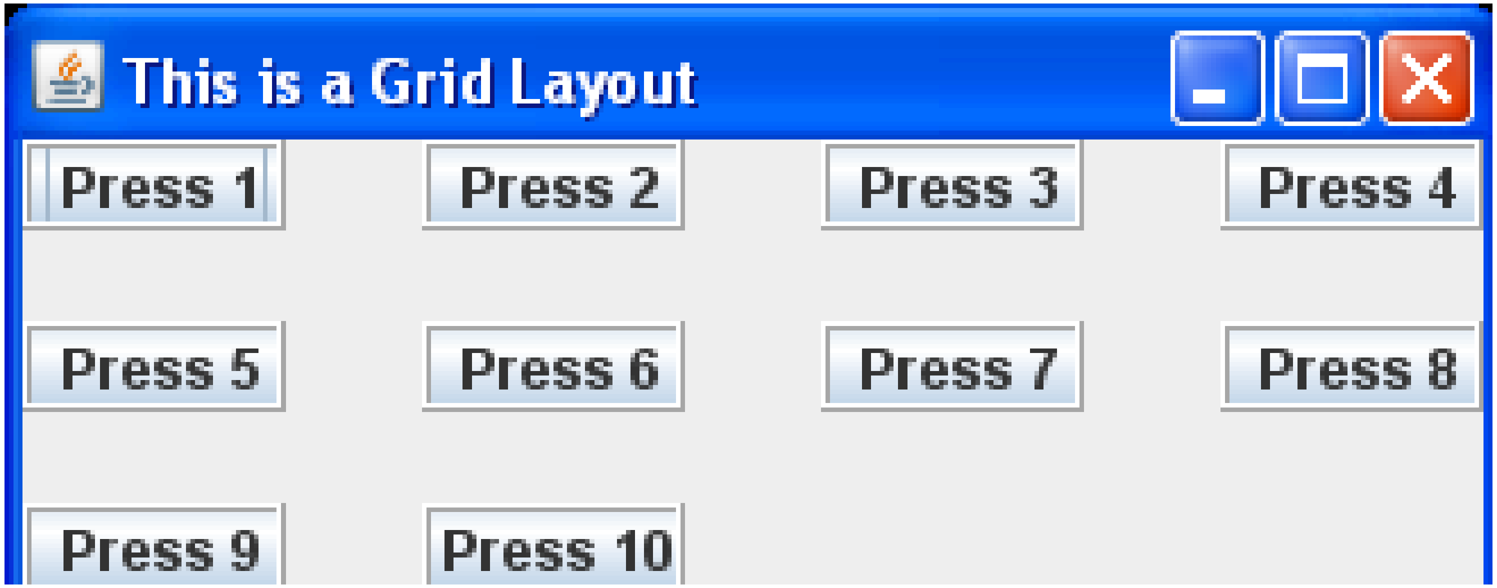
Creates a default BorderLayout manager.

Creates a BorderLayout manager with a specified number of horizontal gap, and vertical gap.

## Grid Layout Manager

- It always ignores a component’s preferred size
- It divides the whole region into a matrix of rows and columns
- Every component is exactly the same size and they appear in the order in which they are added from left to right row by row
- They behave strangely if you put lesser components than number of rows times number of columns or more components

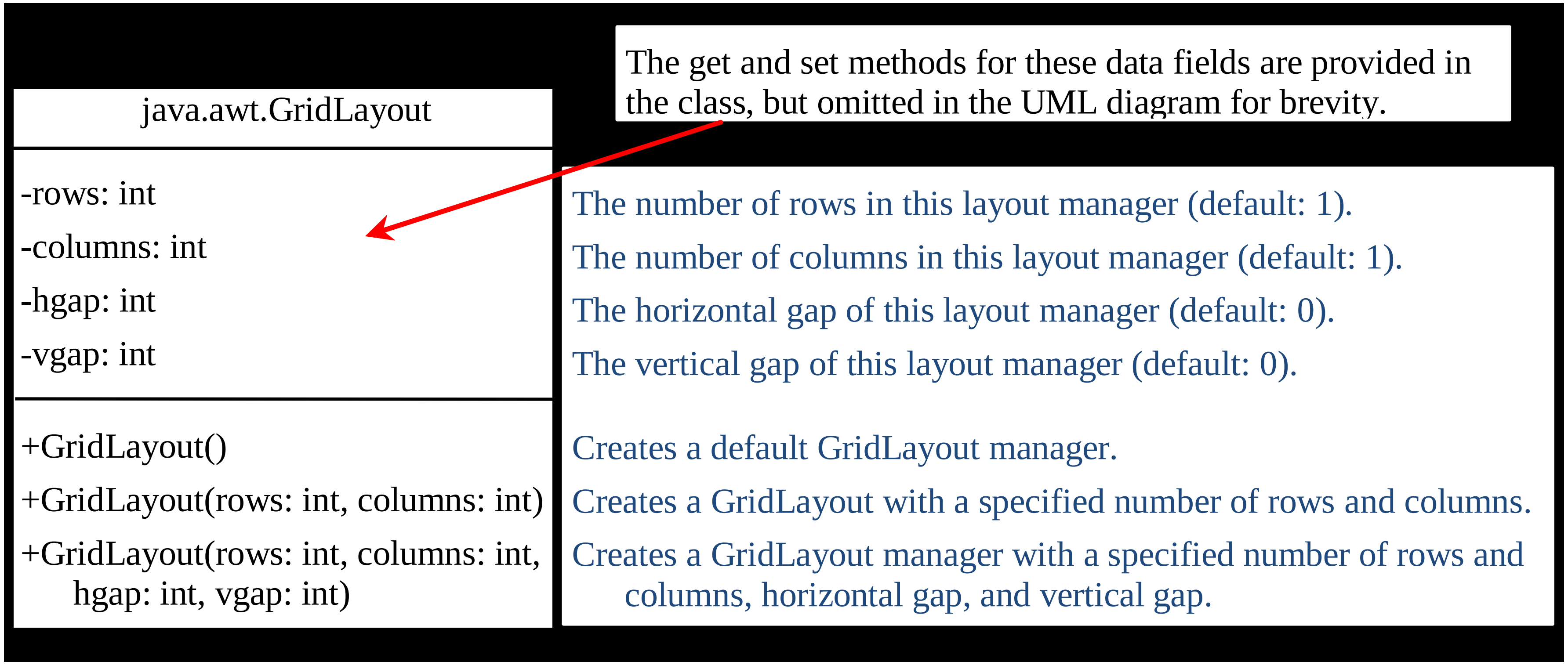
## Grid Layout Manager Example



## Grid Layout Manager Example

```
GridLayout grid = new GridLayout(3,4,30,20);          // Create a layout manager



 Container content = aWindow.getContentPane();  // Get the content pane
    content.setLayout(grid);                          // Set the container layout mgr

JButton button;                                       // Stores a button
    for(int i = 1; i <= 10; i++) {
      content.add(button = new JButton(" Press " + i));    // Add a Button
      button.setBorder(edge);                              // Set the border
    }
```

# The GridLayout Class



The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

| java.awt.GridLayout | |
|---|---|
| -rows: int | The number of rows in this layout manager (default: 1). |
| -columns: int | The number of columns in this layout manager (default: 1). |
| -hgap: int | The horizontal gap of this layout manager (default: 0). |
| -vgap: int | The vertical gap of this layout manager (default: 0). |
| +GridLayout() | Creates a default GridLayout manager. |
| +GridLayout(rows: int, columns: int) | Creates a GridLayout with a specified number of rows and columns. |
| +GridLayout(rows: int, columns: int, hgap: int, vgap: int) | Creates a GridLayout manager with a specified number of rows and columns, horizontal gap, and vertical gap. |

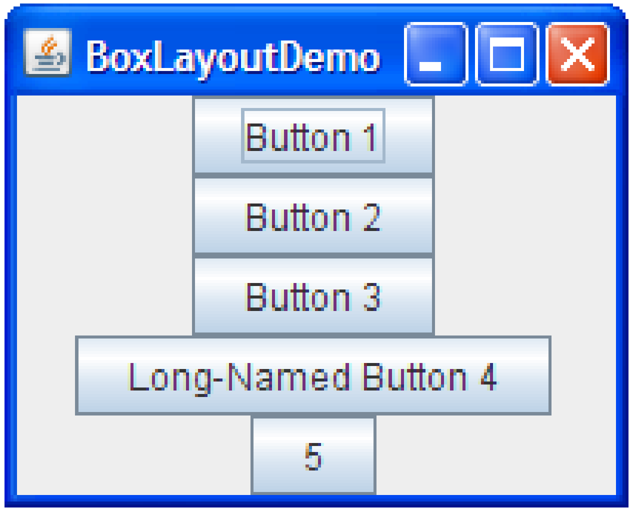
## The GridLayout Class

The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

| java.awt.GridLayout | |
|---|---|
| -rows: int | The number of rows in this layout manager (default: 1). |
| -columns: int | The number of columns in this layout manager (default: 1). |
| -hgap: int | The horizontal gap of this layout manager (default: 0). |
| -vgap: int | The vertical gap of this layout manager (default: 0). |
| +GridLayout() | Creates a default GridLayout manager. |
| +GridLayout(rows: int, columns: int) | Creates a GridLayout with a specified number of rows and columns. |
| +GridLayout(rows: int, columns: int, hgap: int, vgap: int) | Creates a GridLayout manager with a specified number of rows and columns, horizontal gap, and vertical gap. |

## Box Layout Manager

- A general purpose layout manager
- BoxLayout either stacks its components on top of each other or places them in a row



## NULL Layout Manager

- Customized Layout
- Use setBounds (int x, int y, int width, int height) method to set the position/location of the component

## Using Panels as Sub-Containers

- Panels act as sub-containers for grouping user interface components.
- It is recommended that you place the user interface components in panels and place the panels in a frame. You can also place panels in a panel.
- To add a component to JFrame, you actually add it to the content pane of JFrame. To add a component to a panel, you add it directly to the panel using the add method.

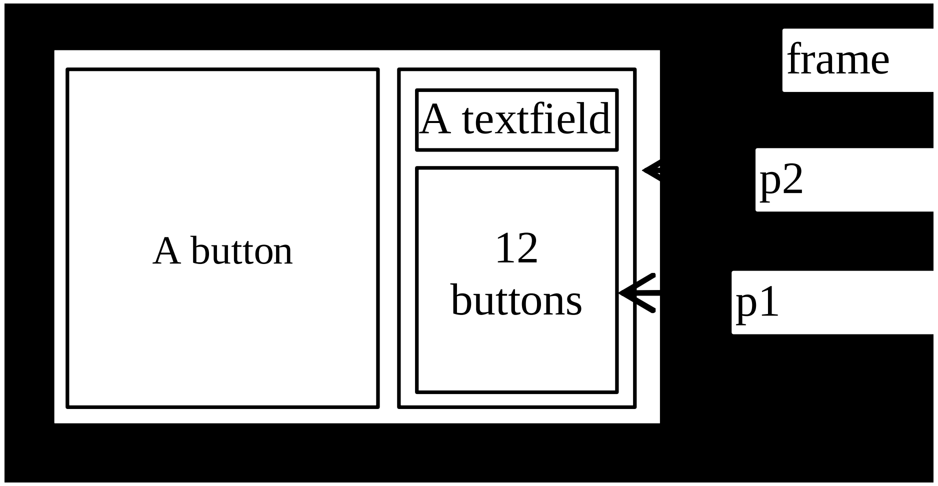
## Creating a JPanel

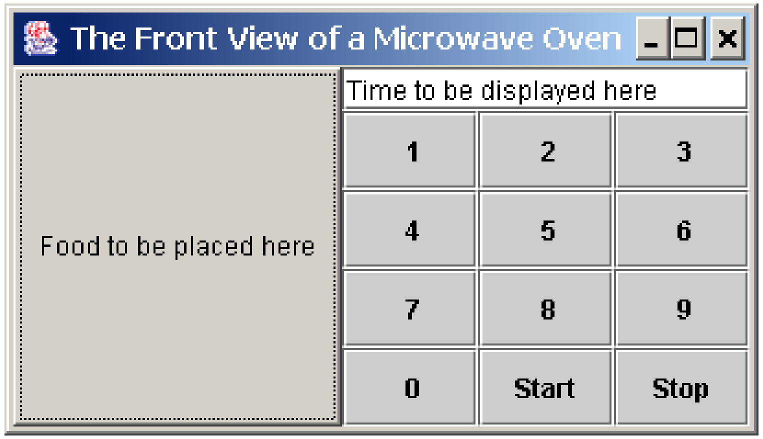
You can use new JPanel() to create a panel with a default FlowLayout manager or new JPanel(LayoutManager) to create a panel with the specified layout manager. Use the add(Component) method to add a component to the panel. For example,

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

## Testing Panels Example

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





Events - - >

## Reading Material

- Reading Material:
  - Chapter 14