CNT 4714: Enterprise Computing

Introduction To GUIs and Event-Driven Programming In Java – Part 1

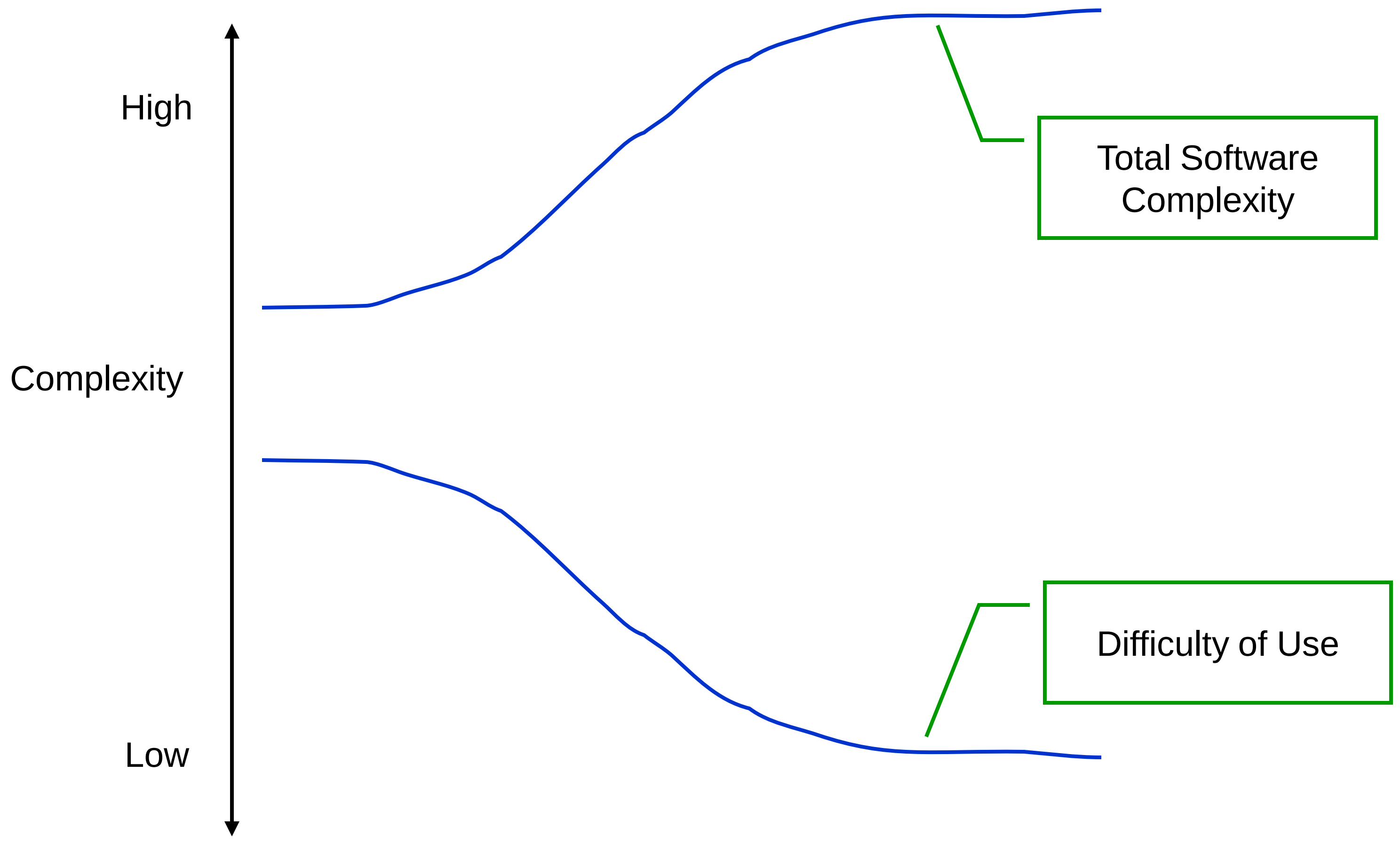
Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
http://www.cs.ucf.edu/courses/cnt4714/spr2010

School of Electrical Engineering and Computer Science
University of Central Florida

GUI and Event-Driven Programming

- Most users of software will prefer a graphical user-interface (**GUI**) -based program over a console-based program any day of the week.
- A GUI gives an application a distinctive “look” and “feel”.
- Providing different applications with consistent, intuitive user interface components allows users to be somewhat familiar with an application, so that they can learn it more quickly and use it more productively.
- Studies have found that users find GUIs easier to manipulate and more forgiving when misused.
- The GUI ease of functionality comes at a programming price – GUI-based programs are more complex in their structure than console-based programs.

The Trade-off Between Ease of Use and Software Complexity



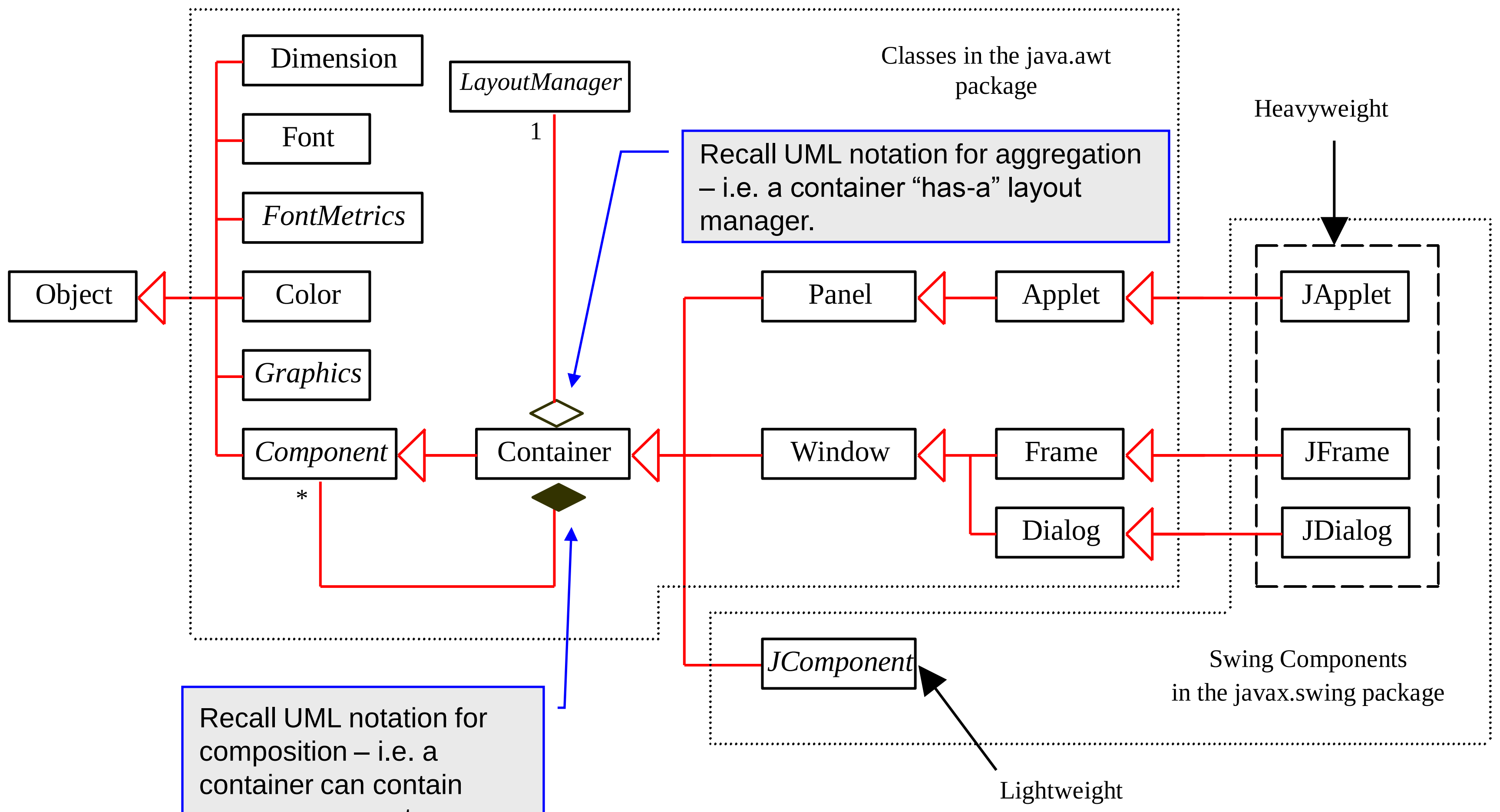
Popularity of GUIs

- Despite the complexity of GUI programming, its dominance in real-world software development makes it imperative that GUI programming be considered.
- Do not confuse GUI-based programming with applets. Although some of the features of the first few GUIs that we look at will be similar to those used in applet programs, notice that we are developing application programs here not applets.
  - The execution of a GUI-based application also begins in its method main(). However, method main() is normally responsible only for creating an *instance* of the GUI.
  - After creating the GUI, the flow of control will shift from the main() method to an event-dispatching loop that will repeatedly check for user interactions with the GUI.

Java GUIs (Graphical User Interfaces)

- The overall design of the API for Java GUI programming is an excellent example of how the object-oriented design principle can be applied.
- You can clearly see the inheritance hierarchy displayed in the GUI API in the UML diagram on the next page.

The Java GUI Hierarchy



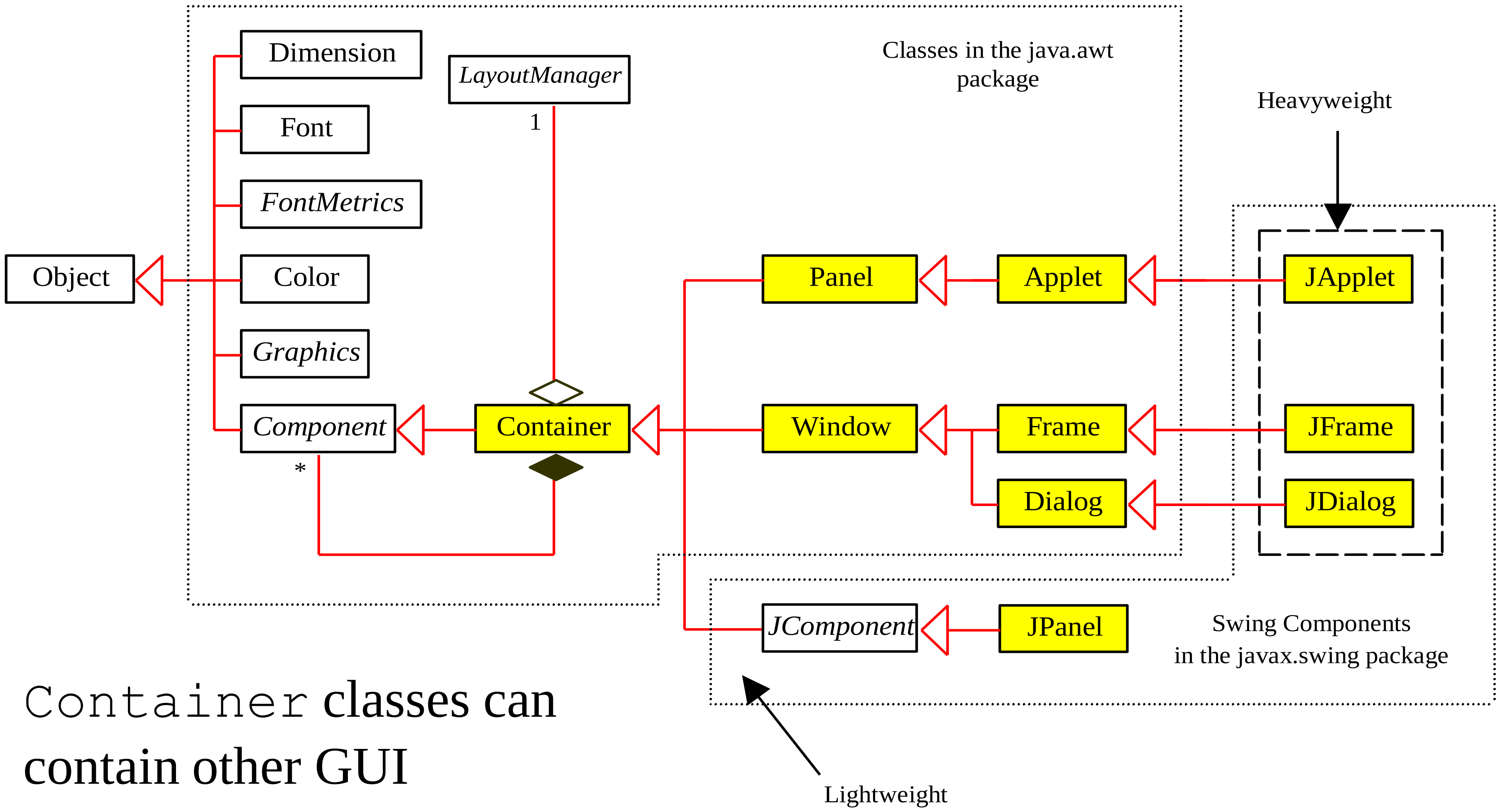
Recall UML notation for aggregation – i.e. a container "has-a" layout manager.

Recall UML notation for composition – i.e. a container can contain many components.

Java GUIs (Graphical User Interfaces)

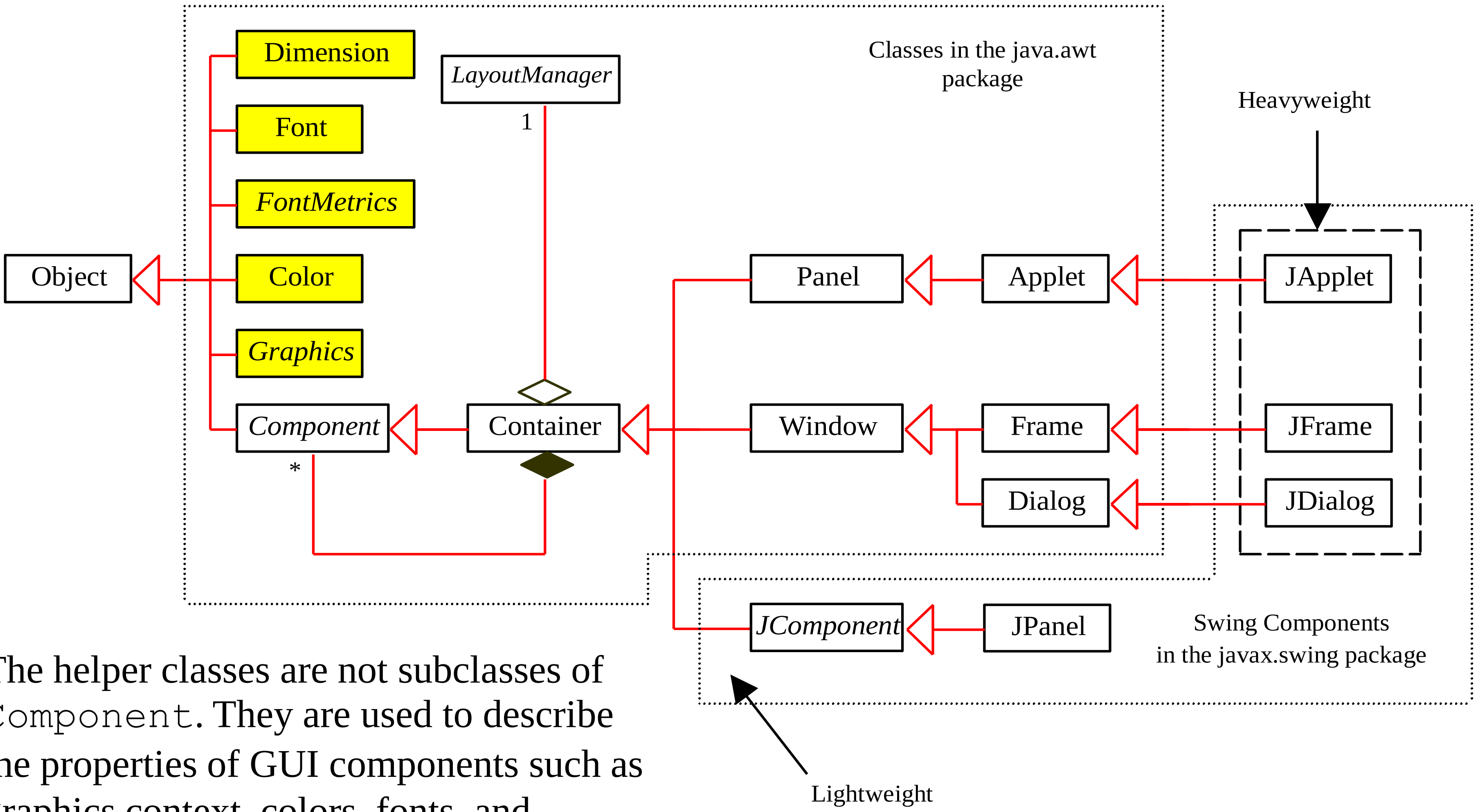
- The GUI classes can be classified into three groups: container classes, component classes, and helper classes.
- The container classes, such as `JFrame`, `JPanel`, and `JApplet`, are used to contained other components.
- The GUI component classes, such as `JButton`, `JTextField`, `JTextArea`, etc., are subclasses of `JComponent`.
- The GUI helper classes, such as `Graphics`, `Color`, `Font`, etc., are used to support GUI components.
- These are illustrated with UML class diagrams on the next two pages.

The Container Classes



`Container` classes can contain other GUI components.

The GUI Helper Classes



The helper classes are not subclasses of `Component`. They are used to describe the properties of GUI components such as graphics context, colors, fonts, and dimension.

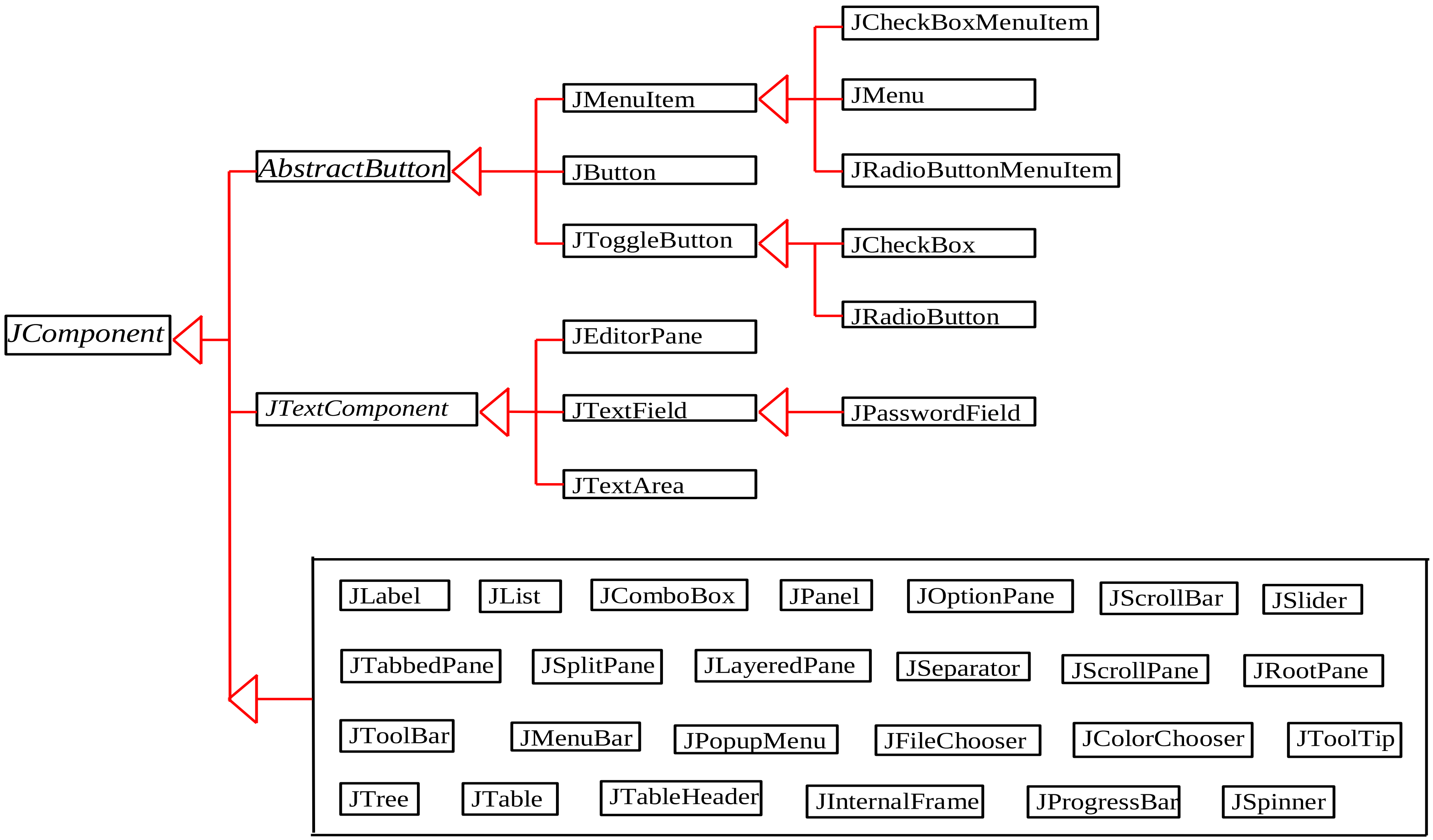
Swing vs. AWT (Abstract Windows Toolkit)

- When Java was introduced, the GUI classes were bundled into a library known as the Abstract Windows Toolkit (AWT).
- For every platform on which Java runs, the AWT components are automatically mapped into platform-specific components through their respective agents, known as peers.
- AWT is fine for developing simple GUIs, but not for developing comprehensive GUI projects. In addition, AWT is prone to platform-specific bugs, because its peer-based approach relies heavily on the underlying platform.
- With the release of Java 2, the AWT user-interface components were replaced by a more robust, versatile, and flexible library known as the Swing components.
- Swing components are painted directly on canvases using Java code, except for classes that are subclasses of `java.awt.Window`, or `java.awt.Panel`, which must be drawn using native GUI on a specific platform.

Swing vs. AWT (Abstract Windows Toolkit)

- Swing components are less dependent on the target platform and use less of the native GUI resource.
- For this reason, Swing components that don’t rely on the native GUI resource are referred to as lightweight components, and AWT components are referred to as heavyweight components.
- To distinguish new Swing component classes from their AWT counterparts, the names of Swing GUI components begin with a prefixed J.
- Although AWT components are still supported in Java 2, it is better to learn how to program using Swing components, because the AWT user-interface components will eventually fade away.
- We will look at Swing components exclusively from here on.
- The UML class hierarchy for the JComponent class is shown on the next page.

JComponent Hierarchy



Swing GUI Components

- `Component` is a superclass of all user-interface classes, and `JComponent` is a superclass of all the lightweight Swing components.
- Since `JComponent` is an abstract class, you cannot use `new JComponent()` to create an instance of `JComponent`. However, you can use the constructors of concrete subclasses of `JComponent` to create `JComponent` instances.
- It is important, if you are going to do any serious GUI programming in Java, to become familiar with the class inheritance hierarchy. For example, the following statements will all display true.

```
Button jbtOK = new JButton(“OK”);
System.out.println(jbtOK instanceof JButton);  //true
System.out.println(jbtOK instanceof AbstractButton);  //true
System.out.println(jbtOK instanceof JComponent);  //true
System.out.println(jbtOK instanceof Container);  //true
System.out.println(jbtOK instanceof Component);  //true
System.out.println(jbtOK instanceof Object);  //true
```

GUI Container Classes

Container Class	Description
`java.awt.Container`	Used to group components. Frames, panels, and applets are its subclasses.
`java.swing.JFrame`	A window not contained inside another window. It is the container that holds other Swing user-interface components in Java GUI applications.
`java.swing.JPanel`	An invisible container that holds user-interface components. Panels can be nested. You can place panels inside a container that includes a panel. `JPanel` is often used as a canvas to draw graphics.
`java.swing.JApplet`	A subclass of `Applet`. You must extend `JApplet` to create a Swing-based Java applet.
`java.swing.JDialog`	A pop-up window or message box generally used as a temporary window to receive additional information (input) from the user or to provide notification that an event has occurred. We've used these from time to time so far this semester and you will be using them in your third programming assignment.

GUI Helper Classes

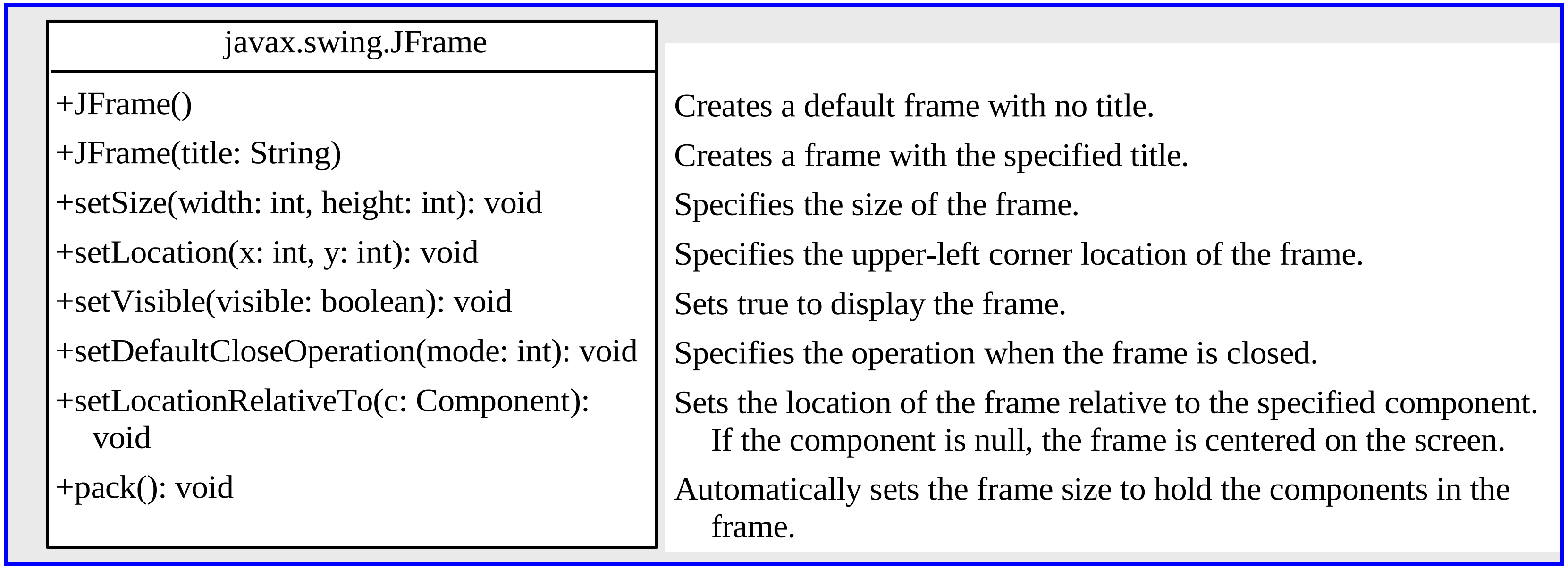
Helper Class	Description
`java.awt.Graphics`	An abstract class that provides a graphical context for drawing strings, lines, and simple shapes.
`java.awt.Color`	Deals with the colors of GUI components. For example, you can specify background or foreground colors in components like JFrame and JPanel, or you can specify colors of lines, shapes, and strings in drawings.
`java.awt.Font`	Specifies fonts for the text and drawings on GUI components. For example, you can specify the font type (e.g., SansSerif), style (e.g., bold), and size (e.g., 24 pt) for the text on a button.
`java.awt.FontMetrics`	An abstract class used to get the properties of the fonts.
`java.awt.Dimension`	Encapsulates the width and height of a component (in integer precision) in a single object.
`java.awt.LayoutManager`	An interface whose instances specify how components are arranged in a container.

NOTE

- The helper classes are in the `java.awt` package.
- The Swing components do not replace all the classes in AWT, only the AWT GUI component classes (e.g., `Button`, `TextField`, `TextArea`).
- The AWT helper classes remain unchanged.

Frames

- To create a user interface, you need to create either a frame or an applet to hold the user-interface components.
- We’re not concerned with applets here, so we’ll be creating a frame.
- To create a frame, use the `JFrame` class.



javax.swing.JFrame	
+JFrame()	Creates a default frame with no title.
+JFrame(title: String)	Creates a frame with the specified title.
+setSize(width: int, height: int): void	Specifies the size of the frame.
+setLocation(x: int, y: int): void	Specifies the upper-left corner location of the frame.
+setVisible(visible: boolean): void	Sets true to display the frame.
+setDefaultCloseOperation(mode: int): void	Specifies the operation when the frame is closed.
+setLocationRelativeTo(c: Component): void	Sets the location of the frame relative to the specified component. If the component is null, the frame is centered on the screen.
+pack(): void	Automatically sets the frame size to hold the components in the frame.

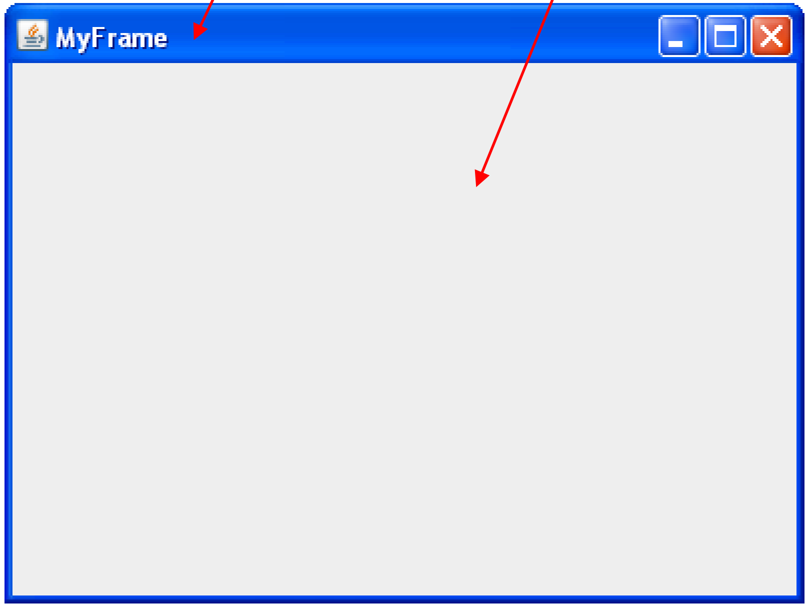
Frames

```
import javax.swing.*;

public class MyFrame {
  public static void main(String[] args) {
   //create a frame
    JFrame frame = new JFrame("MyFrame");
   //set the frame size
    frame.setSize(400, 300);
   //new method since JDK 1.4
    frame.setLocationRelativeTo(null);
frame.setDefaultCloseOperation(JFrame.EXIT_ON
_CLOSE);
   //display the frame
    frame.setVisible(true);  }
}
```

Title bar

Content pane



The EXIT_ON_CLOSE parameter tells the program to terminate when the frame is closed. If this statement is not used, the program does not terminate when the frame is closed and you kill the program externally.

This frame is 400 pixels wide (x-axis) and 300 pixels high (y-axis). The frame is not displayed until the `frame.setVisible(true)` method is invoked. If the `setSize` method is not used, the frame will be sized to display just the title bar. Using the method `setLocationRelativeTo(null)` centers the frame on the screen.
setSize should be invoked before setLocationRelativeTo(null).

Adding Components To A Frame

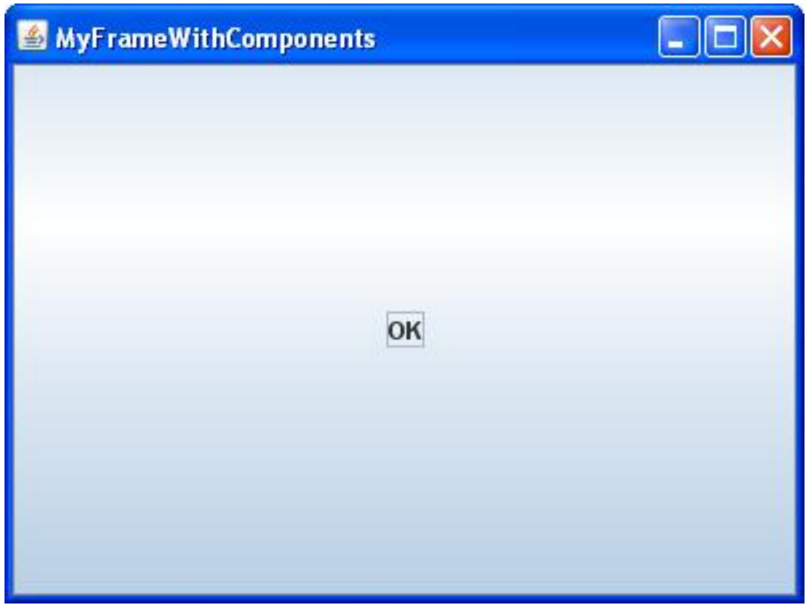
```
import javax.swing.*;

public class MyFrameWithComponents {
  public static void main(String[] args) {
    JFrame frame = new
        JFrame("MyFrameWithComponents");

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

    frame.setSize(400, 300);

frame.setDefaultCloseOperation(JFrame.EXIT_ON
_CLOSE);
    frame.setLocationRelativeTo(null); //
Center the frame
    frame.setVisible(true);
  }
}
```



Since JDK 1.5 you can place components into the content pane by invoking a frame's `add` method. This is called content pane delegation and strictly speaking, it adds a component to the content pane of a frame. Typically, we'll just say that it adds a component to a frame (that it goes into the content pane is implied). To remove a component from a frame use the `remove` method.

NOTE

- Run the `MyFrameWithComponents` program and resize the frame to various sizes.
- What happens to the OK button? It always remains centered in the frame regardless of its size.
- This is because components are placed into a frame by the content panes' layout manager, and the default layout manager for the content pane places the button in the center.
- We'll deal with layout managers next, but try this program before you go any further.

Layout Managers

- In many window-based systems, the user-interface components are arranges by using hard-coded pixel measurements. For example, put a button at location (10,10) in the window. Using hard-coded pixel measurements, the GUI might look fine on one system but be virtually unusable on another.
- Java’s layout managers provide a level of abstraction that automatically maps your GUI on all windows-based systems.
- The Java GUI components are placed in containers, where they are arranged by the container’s layout manager. In the previous example, we did not specify where to put the OK button, but Java knew where to put it because the layout manager works “behind the scenes” to place components in the correct locations.

Layout Managers

- A layout manager is created using a layout manager class.
- Every layout manager class implements the `LayoutManager` interface.
- Layout managers are set in containers using the `setLayout(LayoutManager)` method.
- For example, you can use the following statements to create an instance of XLayout and set it in a container:

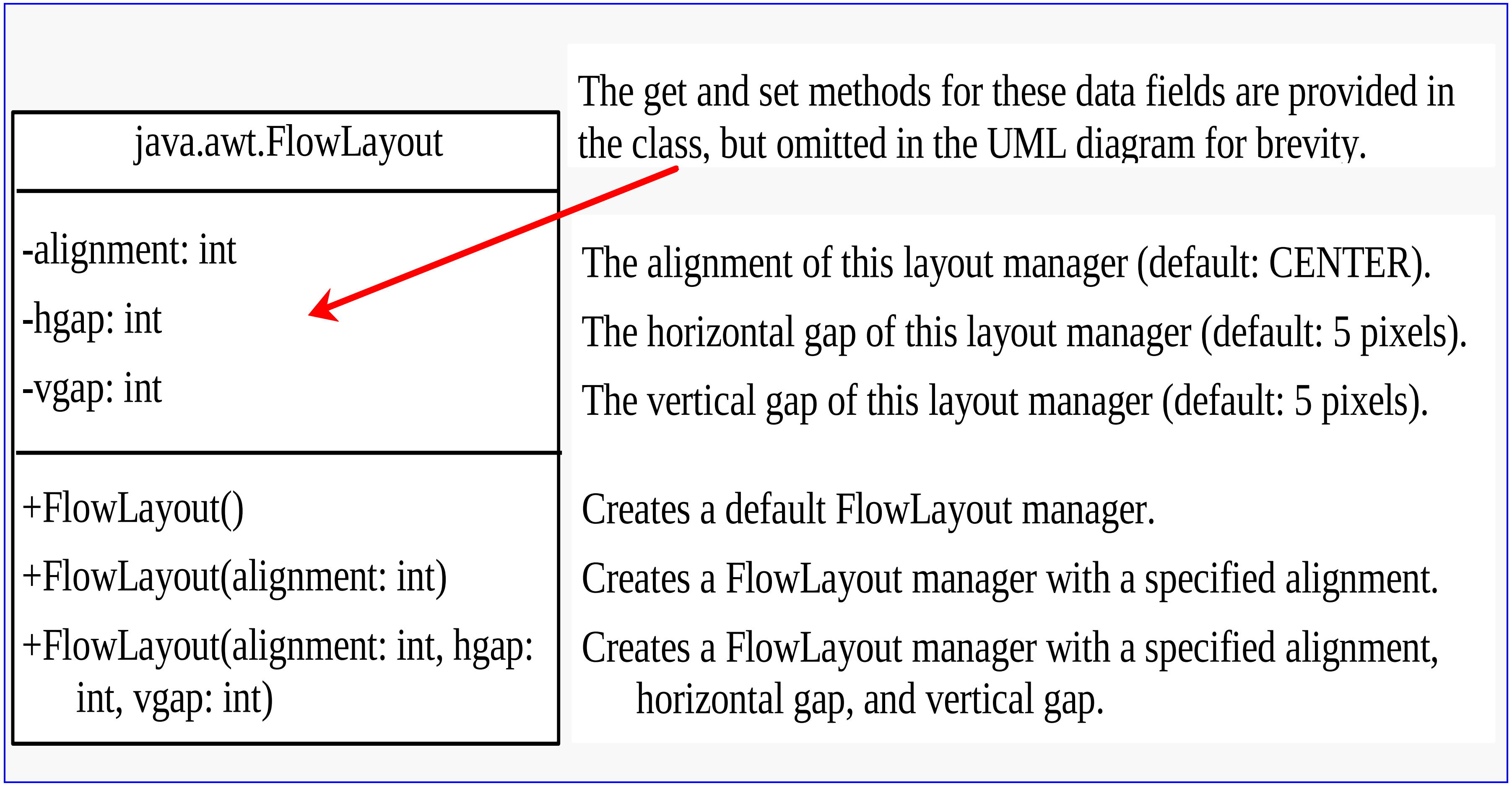
```
LayoutManager layoutManager = new XLayout();
container.setLayout(layoutManager);
```

- Java has several different layout managers, right now we'll focus on three basic layout managers: `FlowLayout`, `GridLayout`, and `BorderLayout`.

FlowLayout

- `FlowLayout` is the simplest layout manager in Java.
- The components are arranged in the container from left to right in the order in which they were added.
- When one row is filled, a new row is started.
- You can specify the way the components are aligned by using one of three constants: `FlowLayout.RIGHT`, `FlowLayout.CENTER`, and `FlowLayout.LEFT`.
- You can also specify the gap between components in pixels.
- The constructors and methods in `FlowLayout` are shown on the next page.

FlowLayout



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.

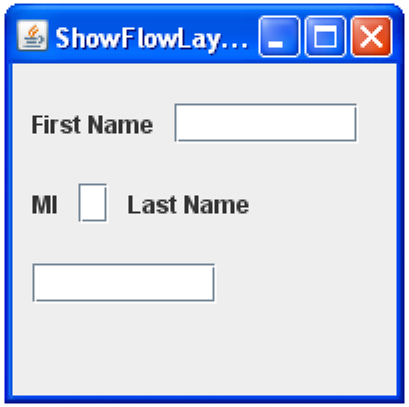
FlowLayout Example

```
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.setSize(200, 200);
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }
}
```



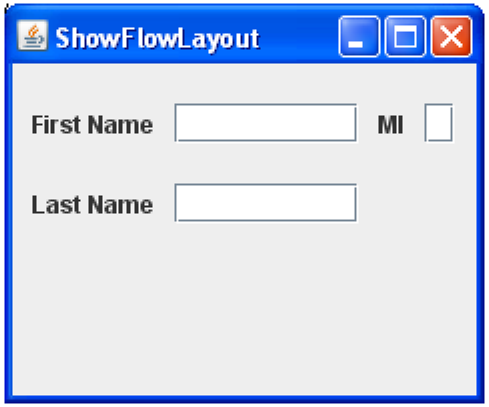
FlowLayout Example

```
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.setSize(240, 200);
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }
}
```



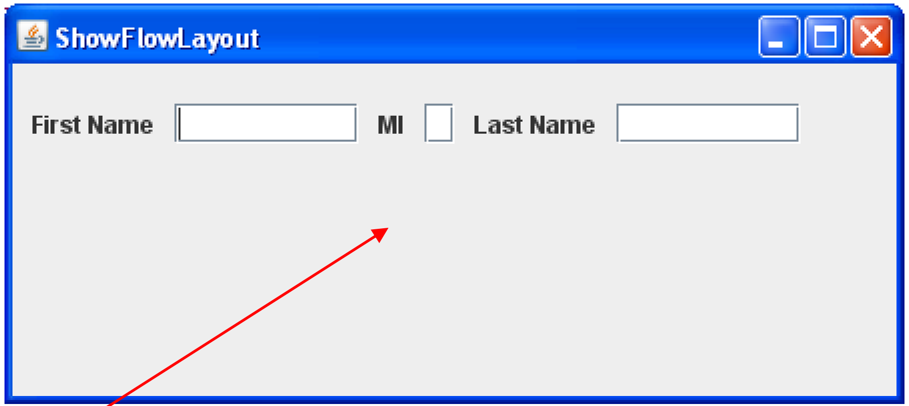
FlowLayout Example

```
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.setSize(450, 200);
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }
}
```



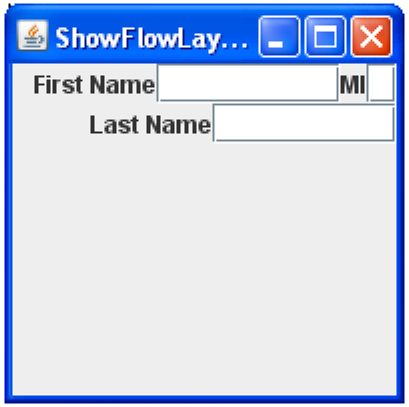
FlowLayout Example

```
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.RIGHT, 0, 0));

    // 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.setSize(200, 200);
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }
}
```



Comments On The `ShowFlowLayout` Program

- The `ShowFlowLayout` program is created in a different style than the `MyFrameWithComponents` program.
- The `MyFrameWithComponents` program created a frame using the `JFrame` class. The `ShowFlowLayout` program creates a class named `ShowFlowLayout` that extends the `JFrame` class. The main method in this program creates an instance of `ShowFlowLayout`. The constructor of `ShowFlowLayout` constructs and places the components in the frame.
- This is the preferred style for creating GUI applications for three reasons:
  1. Creating a GUI applications means creating a frame, so it is natural to define a frame to extend `JFrame`.
  2. The frame may be further extended to add new components or features.
  3. The class can be easily reused. For example, you can create multiple frames by creating multiple instances of the class.

Comments On The `ShowFlowLayout` Program

- In addition, using one style makes programs easier to read. From now on, most of the GUI classes that we construct will extend the `JFrame` class. The constructor of the main class constructs the user interface. The `main` method creates an instance of the main class and then displays the frame.

- In the `ShowFlowLayout` program, the `FlowLayout` manager is used to place components in a frame. If you resize the frame, the components are automatically rearranged to fit in the new size frame (see pages 22-24).

- The `setTitle` method is defined in the `java.awt.Frame` class. Since `JFrame` is a subclass of `Frame`, you can use it to set a title for an object of `JFrame`.

Comments On The `ShowFlowLayout` Program

- An anonymous `FlowLayout` object was created in the statement: `setLayout(new FlowLayout(FlowLayout.LEFT,10,20);` This code is equivalent to:

```
FlowLayout layout = new FlowLayout(FlowLayout.LEFT,10,10);

setLayout(layout);
```

This code creates an explicit reference to the object layout of the `FlowLayout` class. The explicit reference is not necessary, in this case, because the object is not directly referenced in the `ShowFlowLayout` class.

- Don't forget to put the new operator before a layout manager class when setting a layout style – for example, `setLayout(new FlowLayout()).`

- Notice that the constructor `ShowFlowLayout()` does not explicitly invoke the constructor `JFrame()`, but the constructor `JFrame()` is invoked implicitly – recall constructor chaining!

GridLayout

- The `GridLayout` manager arranges components in a grid (matrix) formation with the number of rows and columns defined by the constructor.
- The components are placed in the grid in a row major order (i.e., left to right beginning with row 1, then the second row, and so on), in the order in which they are added.
- The constructors and methods in `GridLayout` are shown on the next page.

GridLayout

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.

`GridLayout` Specifics

- You can specify the number of rows and columns in the grid following these basic rules:
  - The number of rows and columns can be zero, but not both. If one is zero and the other is nonzero, the nonzero dimension is fixed, while the zero dimension is determined dynamically by the layout manager.
    - For example, if you specify zero rows and three columns for a grid that has ten components, `GridLayout` creates three fixed columns of four rows, with the last row containing only one component. If you specify three rows and zero columns for a grid with ten components, `GridLayout` creates three fixed rows of four columns, with the last row containing two components.

`GridLayout` Specifics

- – If both the number of rows and columns are nonzero, the number of rows is the dominating parameter; that is, the number of rows is fixed, and the layout manager dynamically calculates the number of columns.
  - For example, if you specify three rows and three columns for a grid that has ten components, `GridLayout` creates three fixed columns of four rows, with the last row containing two components.

- The example on the next page illustrates the `GridLayout` manager.

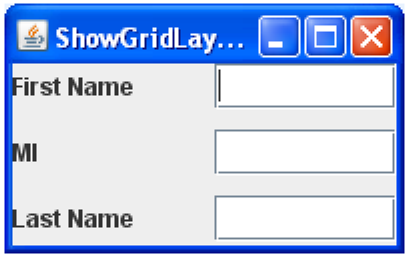
GridLayout Example

```java
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, 10, 10));

    // 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.setSize(200, 125);
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }
}
```



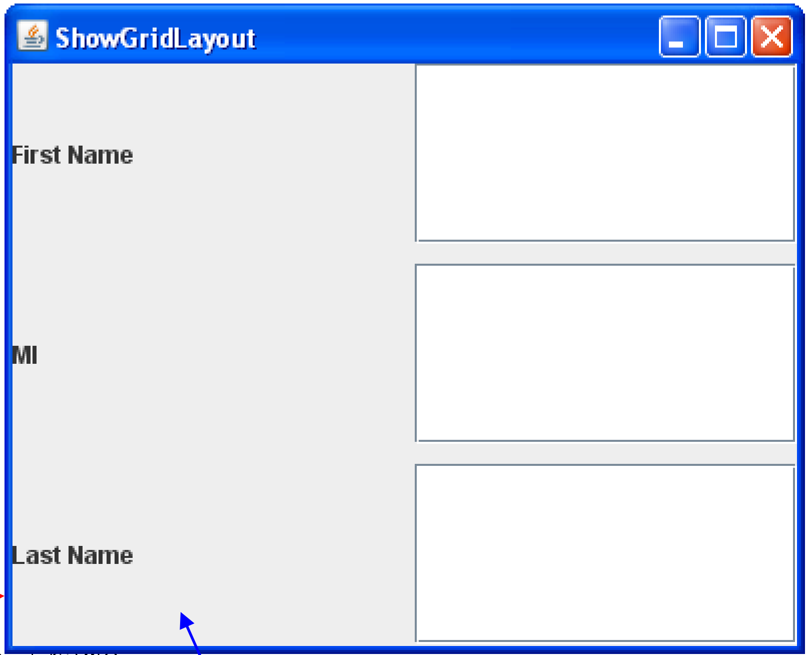
GridLayout Example

```
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, 10, 10));

    // 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.setSize(400, 325);
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }
}
```



All components are equal size, number of rows, number of columns and gap size remains unchanged in larger frame.

Comments On The `ShowGridLayout` Program

- What would happen if the `setLayout` where replaced with `setLayout(new GridLayout(4,2,10,10))`?
- How about if it were replaced with

  `setLayout(new GridLayout(2,2,10,10))`?

- Try both of these yourself to see the effect.
- NOTE: The order in which the components are added to the container is important in both `FlowLayout` and `GridLayout`. It determines the location of the components in the container. See the modified version of the `GridLayout` program on the next page to see this effect.

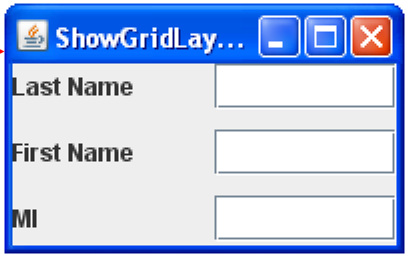
GridLayout Example

```
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, 10, 10));

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

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



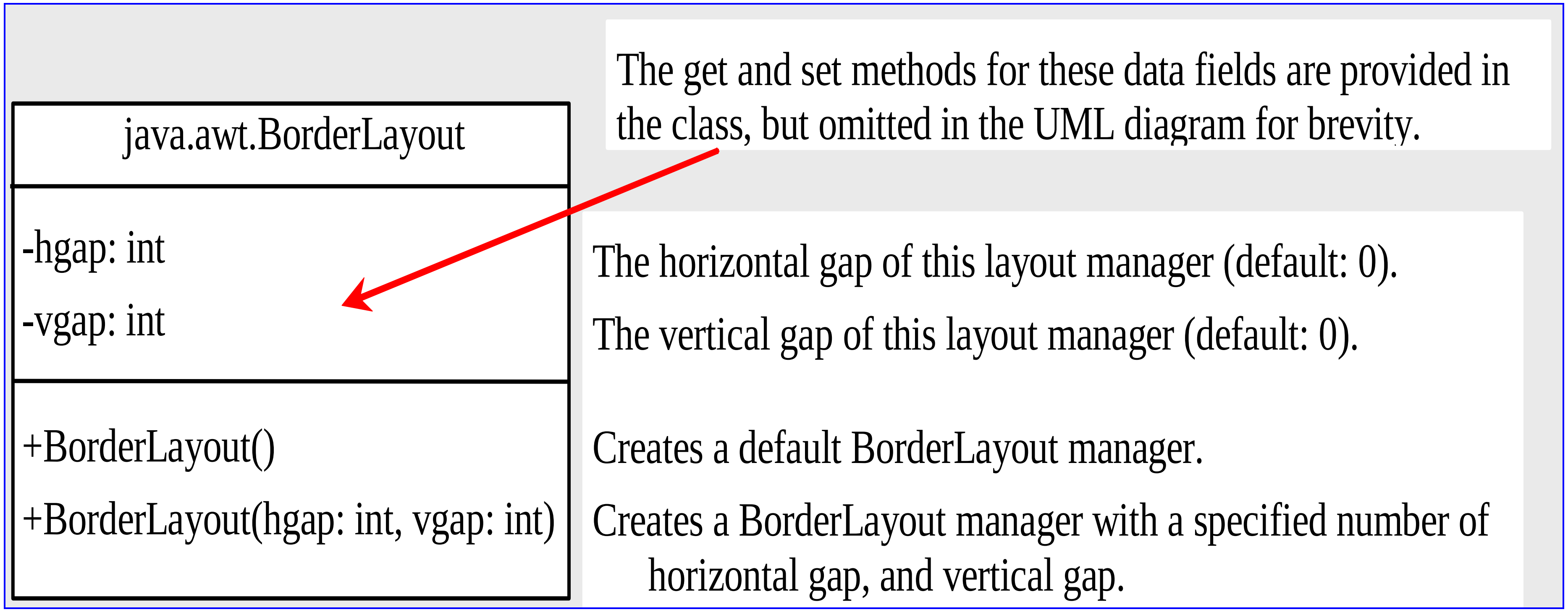
`BorderLayout`

- The `BorderLayout` manager divides the window into five areas: East, South, West, North, and Center.

- Components are added to a `BorderLayout` by using `add(Component, index)` where `index` is a constant `BorderLayout.EAST`, `BorderLayout.SOUTH`, `BorderLayout.WEST`, `BorderLayout.NORTH`, `BorderLayout.CENTER`.

- The constructors and methods in `BorderLayout` are shown on the next page.

BorderLayout



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

-vgap: int

+BorderLayout()

+BorderLayout(hgap: int, vgap: int)

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.

`BorderLayout`

- The components are laid out according to their preferred sizes and where they are placed in the container.
- The North and South components can stretch horizontally; the East and West components can stretch vertically, the Center component can stretch both horizontally and vertically to fill any empty space.
- The example program on the next page illustrates a border layout. The program adds five buttons labeled East, South, West, North, and Center into the frame using a `BorderLayout` manager.

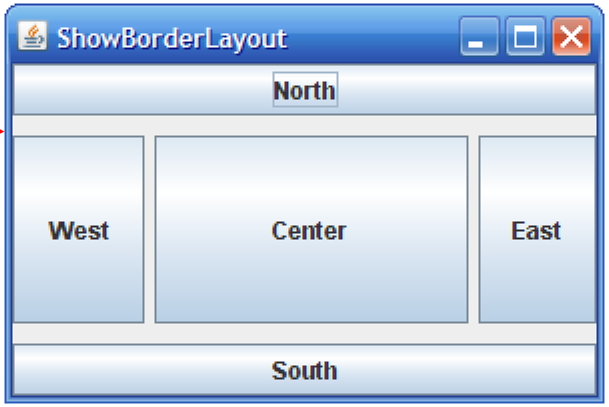
BorderLayout Example

```
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.setSize(300, 200);
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }
}
```



Properties of Layout Managers

- Layout managers have properties that can be changed dynamically. `FlowLayout` has `alignment`, `hgap`, and `vgap` properties (see page 21). You can use the `setAlignment`, `setHgap`, and `setVgap` methods to specify the alignment and the horizontal and vertical gaps.
- `GridLayout` has the rows, columns, hgap, and vgap properties (see page 30). You can use the `setRows`, `setColumns`, `setHgap`, and `setVgap` methods to specify the number of rows, the number of columns, and the horizontal and vertical gaps.
- `BorderLayout` has the `hgap`, and `vgap` properties (see page 38). You can use the `setHgap` and `setVgap` methods to specify the horizontal and vertical gaps.
- In the previous three examples for the three different layout managers, an anonymous layout manager was used because the properties of the layout manager did not need to change once it was created.

Properties of Layout Managers

- If you need to change the properties of a layout manager dynamically, the layout manager must be explicitly referenced by a variable.
- You can then change the properties of the layout manager through the variable.
- For example, the following code creates a layout manager and sets its properties.

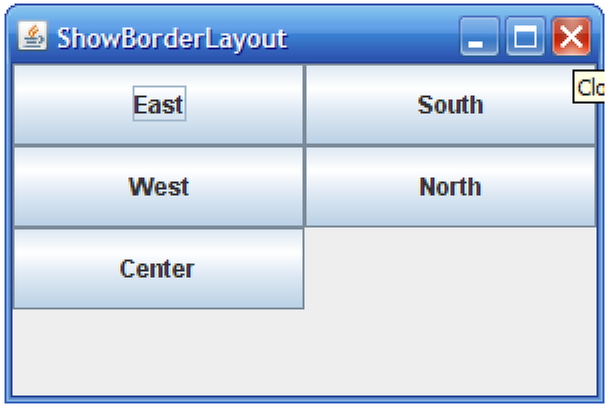
```
//create a layout manager
FlowLayout layout = new FlowLayout();
//set layout manager properties
layout.setAlignment(FlowLayout.RIGHT);
layout.setHgap(10);
layout.setVgap(20);
```

The `validate` Method

- A container can have only one layout manager at a time.
- You can change a container's layout manager by using the `setLayout(aNewLayout)` method and then use the `validate()` method to force the container to again lay out the components in the container using the new layout manager.
- The example on the next page illustrate this method.

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

public class ModifiedShowBorderLayout extends JFrame {
  public ModifiedShowBorderLayout() {
    // create explicit layout manager using BorderLayout
    BorderLayout layout = 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);
    //modify the layout manager from the initial one
    setLayout(new GridLayout(4,2));
    validate();
  }
  /** Main method */
  public static void main(String[] args) {
    ModifiedShowBorderLayout frame = new ModifiedShowBorderLayout();
    frame.setTitle("ShowBorderLayout");
    frame.setSize(300, 200);
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    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 (RGB model), each represented by an unsigned byte value that describes its intensity, ranging from 0 (darkest shade) to 255 (lightest shade).
- You can create a color using the following constructor: `public Color (int r, int g, int b);`

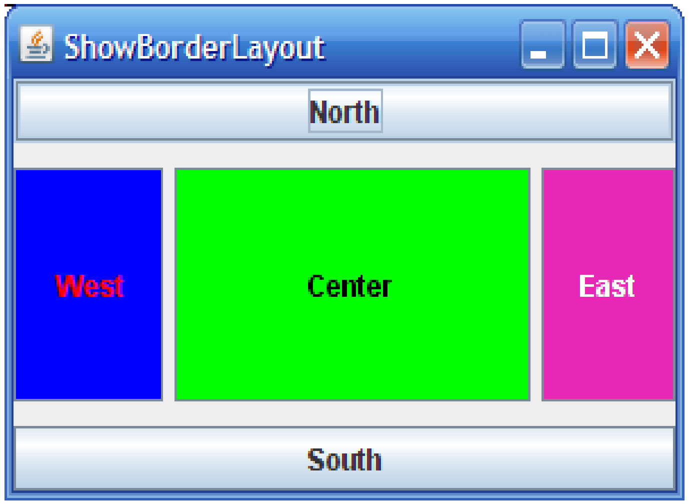
  where `r`, `g`, and `b` specify a color by its red, green, and blue components. For example:

```
Color color = new Color(128, 100, 100);
```

The `Color` Class

- The arguments `r`, `g`, and `b` in the `Color` constructor are between 0 and 255. If a value beyond this range is passed to the argument, an `IllegalArgumentException` will occur.
- You can use `setBackground(Color c)` and `setForeground(Color c)` methods defined in the `java.awt.Component` class to set a component's background and foreground colors.
- You can also use one of the 13 standard colors (`BLACK`, `BLUE`, `CYAN`, `DARKGRAY`, `GRAY`, `GREEN`, `LIGHTGRAY`, `MAGENTA`, `ORANGE`, `PINK`, `RED`, `WHITE`, `YELLOW`) defined as constants in `java.awt.Color`.
- The example on the next page modifies the `ShowBorderLayout` program to color some of the buttons. (NOTE: I've only included the modified code not the entire program this time.)

```
public class ColorShowBorderLayout extends JFrame {
  public ColorShowBorderLayout() {
    // Set BorderLayout with horizontal gap 5 and
vertical gap 10
    setLayout(new BorderLayout(5, 10));
    //Create and color buttons
    Color myColor = new Color(230,40,180);
    JButton east = new JButton("East");
    east.setBackground(myColor);
    east.setForeground(Color.WHITE);
    JButton west = new JButton("West");
    west.setBackground(Color.BLUE);
    west.setForeground(Color.RED);
    JButton center = new JButton("Center");
    center.setBackground(Color.GREEN);
    center.setForeground(Color.BLACK);
    // Add buttons to the frame
    add(east, BorderLayout.EAST);
    add(new JButton("South"), BorderLayout.SOUTH);
    add(west, BorderLayout.WEST);
    add(new JButton("North"), BorderLayout.NORTH);
    add(center, BorderLayout.CENTER);
  }
```



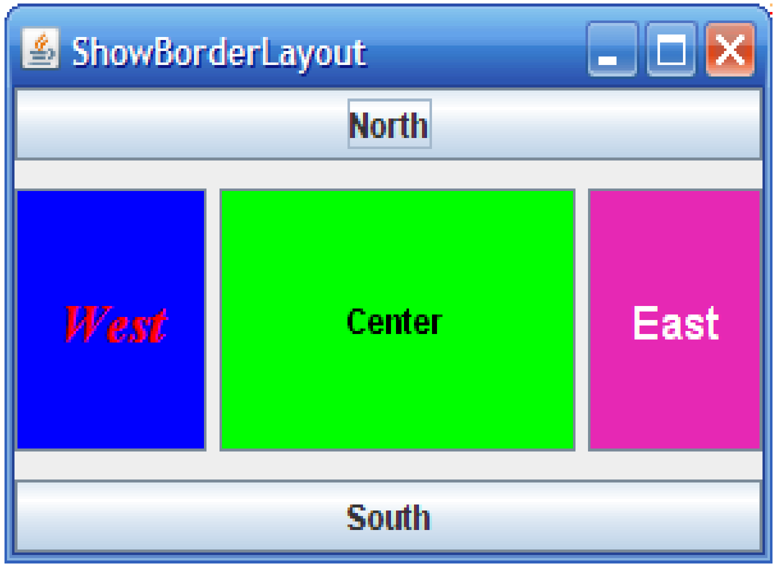
The `Font` Class

- You can create a font using the `java.awt.Font` class and set fonts for the components using the `setFont` method in the `Component` class.
- The constructor for `Font` is:

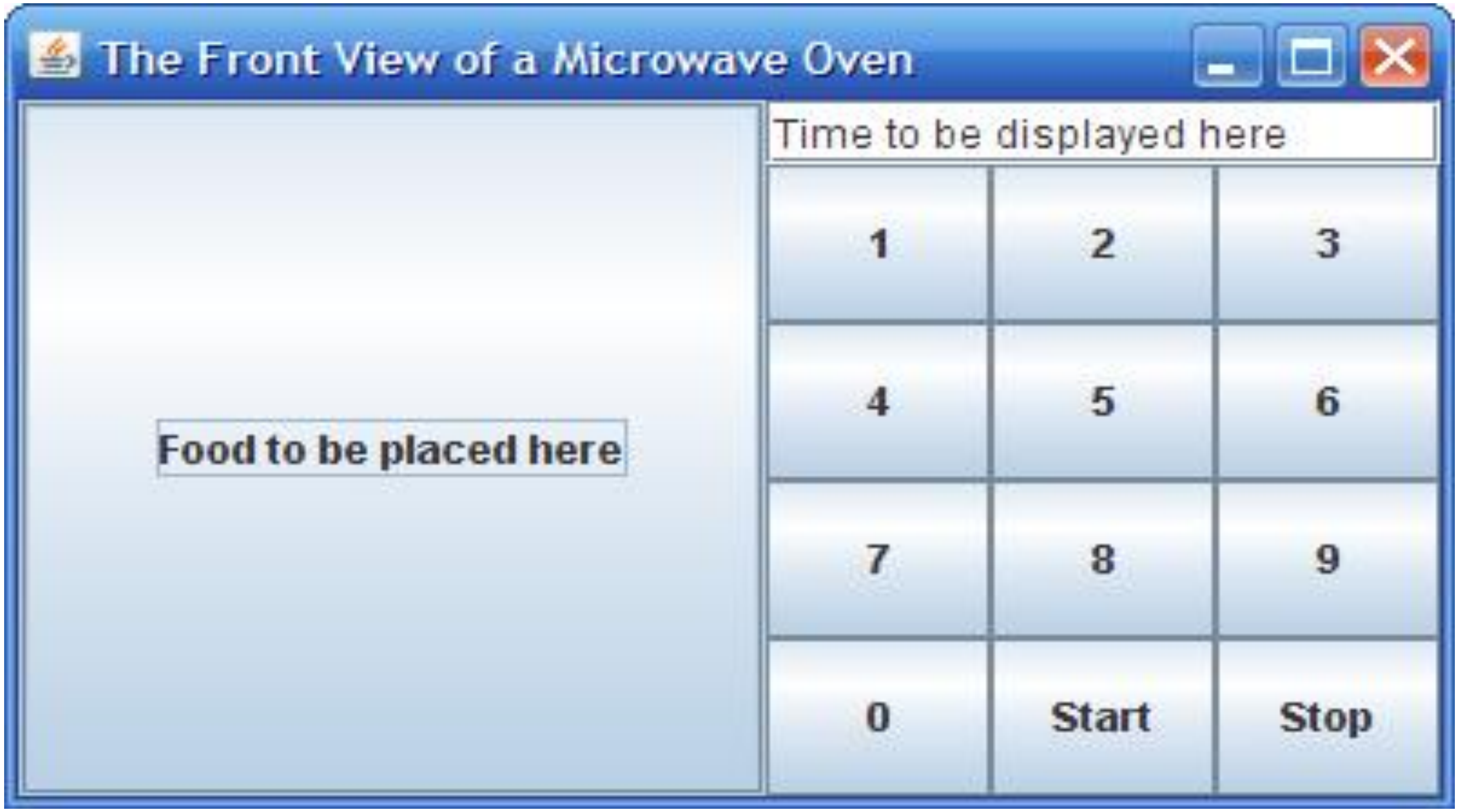
```
public Font(String name, int style, int size);
```

- You can choose a font name from `SanSerif`, `Serif`, `Monospaced`, `Dialog`, or `DialogInput`, choose a style from `Font.Plain(0)`, `Font.BOLD(1)`, `Font.ITALIC(2)`, and `Font.BOLD + Font.ITALIC(3)`, and specify a font size of any positive integer.
- The example on the following page, modifies some of the buttons in the `ShowBorderLayout` example, again, I've included only the modified code.

```
public class ColorShowBorderLayout extends JFrame {
  public ColorShowBorderLayout() {
    // Set BorderLayout with horizontal gap 5 and vertical gap
10
    setLayout(new BorderLayout(5, 10));
    //Create fonts and color buttons
    Font font1 = new Font("SansSerif", Font.BOLD, 16);
    Font font2 = new Font("Serif", Font.BOLD+Font.ITALIC, 20);
    Color myColor = new Color(230,40,180);
    JButton east = new JButton("East");
    east.setFont(font1);
    east.setBackground(myColor);
    east.setForeground(Color.WHITE);
    JButton west = new JButton("West");
    west.setBackground(Color.BLUE);
    west.setForeground(Color.RED);
    west.setFont(font2);
    JButton center = new JButton("Center");
    center.setBackground(Color.GREEN);
    center.setForeground(Color.BLACK);
    // Add buttons to the frame
    add(east, BorderLayout.EAST);
    add(new JButton("South"), BorderLayout.SOUTH);
    add(west, BorderLayout.WEST);
    add(new JButton("North"), BorderLayout.NORTH);
    add(center, BorderLayout.CENTER);
  }
```



Try To Create This GUI


The Front View of a Microwave Oven
Time to be displayed here
1
2
3
4
5
6
7
8
9
0
Start
Stop
Food to be placed here