Programming in Java – Part 13
Basics of Swing

Paolo Vercesi

ESTECO SpA

Agenda

Hello, World!

The rules of the game

Hello, World!

Graphical user interfaces (GUI) and OOP

Object-oriented programming is very well suited for GUI programming

GUI components or controls are natural objects: windows, buttons, labels, text fields, etc., GUI programming is naturally asynchronous and event oriented

In a GUI application, the main method is responsible to initialize and assemble the GUI and the application logic, and then to make the GUI visible

GUI libraries for Java

- *Swing* 
  - *Abstract Widget Toolkit (AWT)*
  - *Part of Java SE*
- *JavaFX*
- *Standard Widget Toolkit (SWT)*
- *All available for Windows, Linux, and MacOS*

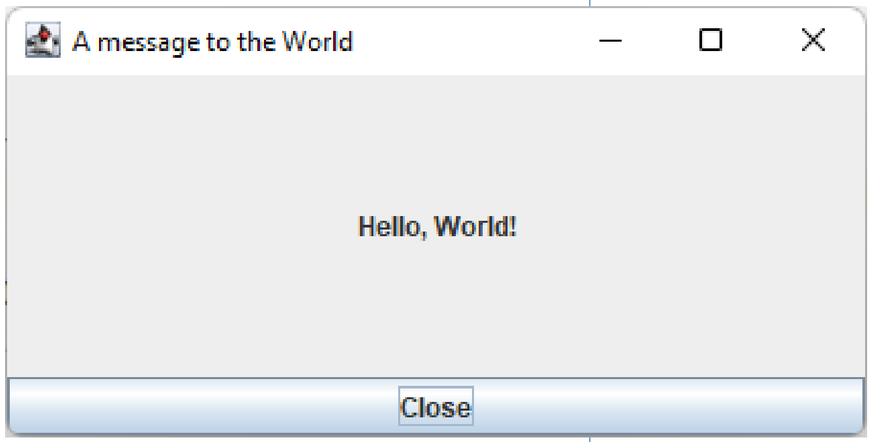
How to learn Java Swing

- *Official tutorial https://docs.oracle.com/javase/tutorial/uiswing/index.html*
  - *be aware that it is based on Java 8*
  - *released in 2014*
  - *the API hasn't changed in the meanwhile*
  - *good to understand how the components work*
- *Study the Java documentation*
- *Look at the source code*
- *Attend this introduction to Java Swing*
- *Do a lot of experiments*

Hello, World!

HelloWorld.java

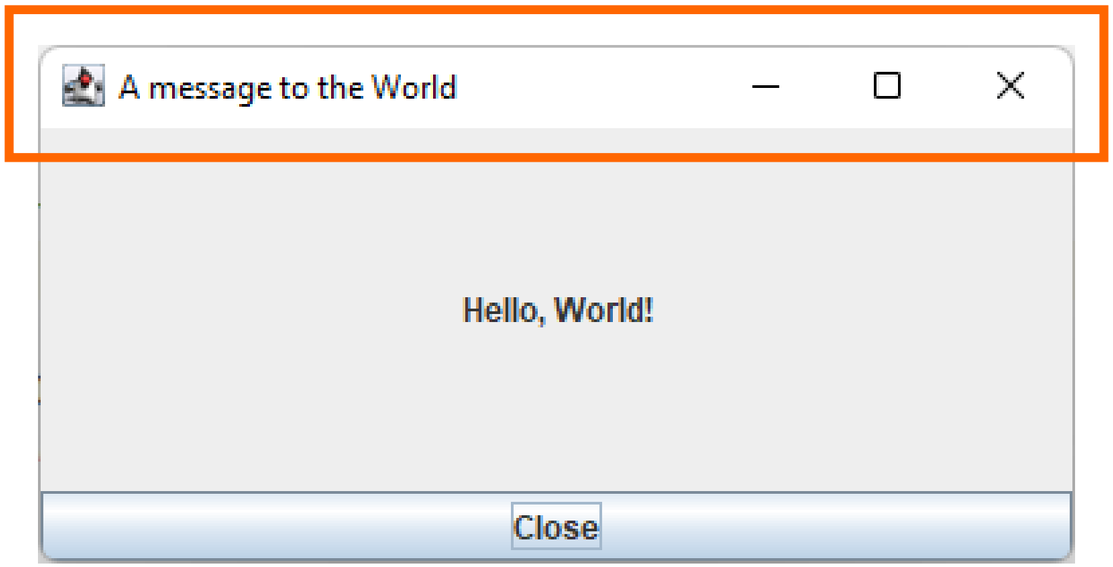
```
public class HelloWorld {

    static void main() {
        SwingUtilities.invokeLater(HelloWorld::helloWorld);
    }

    private static void helloWorld() {
        JFrame frame = new JFrame("A message to the World");
        frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);

        JLabel label = new JLabel("Hello, World!");
        label.setHorizontalAlignment(SwingConstants.CENTER);
        frame.getContentPane().add(label, BorderLayout.CENTER);

        JButton closeButton = new JButton("Close");
        closeButton.addActionListener(x -> frame.dispose());
        frame.getContentPane().add(closeButton, BorderLayout.SOUTH);

        frame.setSize(400, 200);
        frame.setVisible(true);
    }
}
```



Analysis of HelloWorld.java 1/5

HelloWorld.java

```
JFrame frame = new JFrame("A message to the World");
frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
```



A `JFrame` represents a window with all the decorations: icon, title, and buttons to minimize, maximize, and close

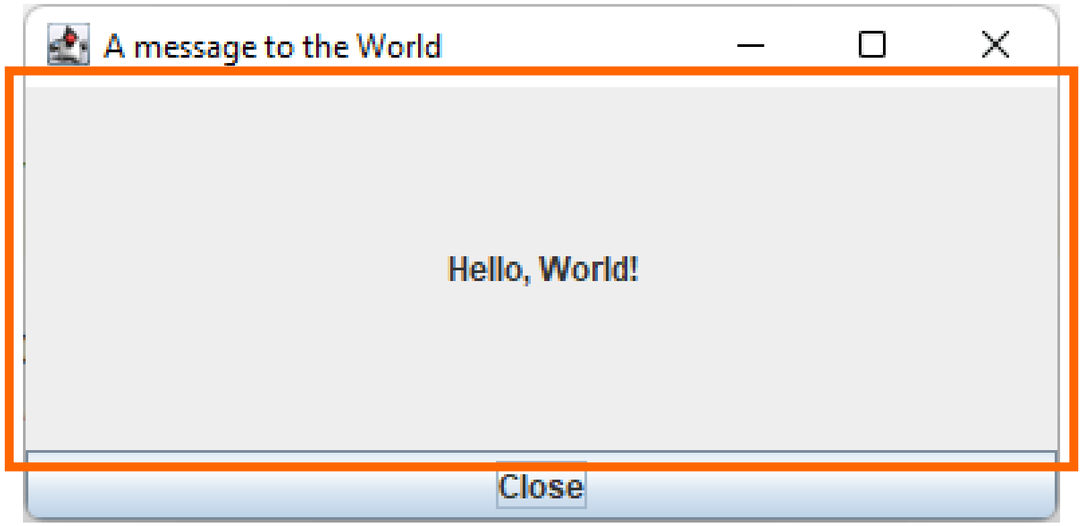
The behavior of the close button can be customized, for example to dispose the `JFrame`

By disposing a `JFrame`, you close the `JFrame` window if open, and you release all the resources associated to this `JFrame`

Analysis of HelloWorld.java 2/5

HelloWorld.java

```
JLabel label = new JLabel("Hello, World!");
label.setHorizontalAlignment(SwingConstants.CENTER);
frame.getContentPane().add(label, BorderLayout.CENTER);
```



The content pane of a `JFrame` *uses the* `BoderLayout` *manager by default*

A `JLabel` *is a Swing component used to represents a piece of text with an icon*

To make a Swing component visible, we must add it to a container, if there are no intermediate containers, we can add it to the content pane of the `JFrame` *directly*

A container uses a layout manager to layout the components it contains, when adding a component to a container we can specify a layout constraint

Analysis of HelloWorld.java 3/5

HelloWorld.java

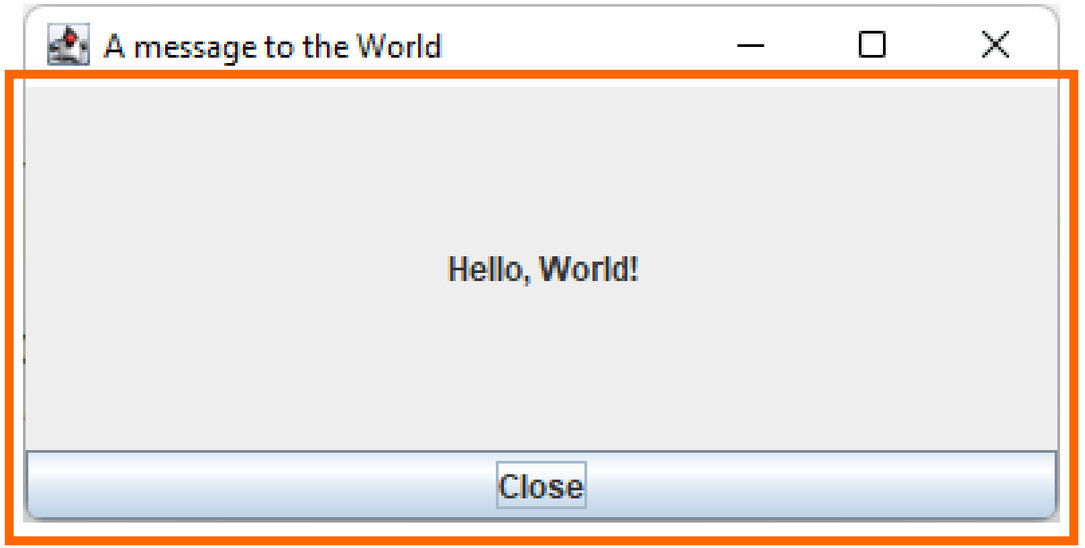
```
JButton closeButton = new JButton("Close");
closeButton.addActionListener(x -> frame.dispose());
frame.getContentPane().add(closeButton, BorderLayout.SOUTH);
```

A `JButton` is a Swing component able to respond to user actions, for example, when you click on the button, it triggers an action listener

Analysis of HelloWorld.java 4/5

HelloWorld.java

```
frame.setSize(400, 200);
frame.setVisible(true);
```



A `JFrame` and its content pane are shown in the screen when we make the frame visible

Analysis of HelloWorld.java 5/5

HelloWorld.java

```
static void main() {
    SwingUtilities.invokeLater(HelloWorld::helloWorld);
}
```

Almost all GUI code **MUST** *run on the Event Dispatch Thread by using either the static methods* `invokeLater` or `invokeAndWait` *of* `SwingUtilities`

`static void` **`invokeAndWait`**`(`**`Runnable`** `doRun)`	Causes *doRun.run()* to be executed synchronously on the AWT event dispatching thread.
`static void` **`invokeLater`**`(`**`Runnable`** `doRun)`	Causes *doRun.run()* to be executed asynchronously on the AWT event dispatching thread.

More on this topic in the next lecture!

Take aways

- *Swing is a library used to develop a graphical user interface (GUI) for Java programs*
- *Swing is part of the "The Java Platform, Standard Edition (Java SE) APIs"*

The rules of the game

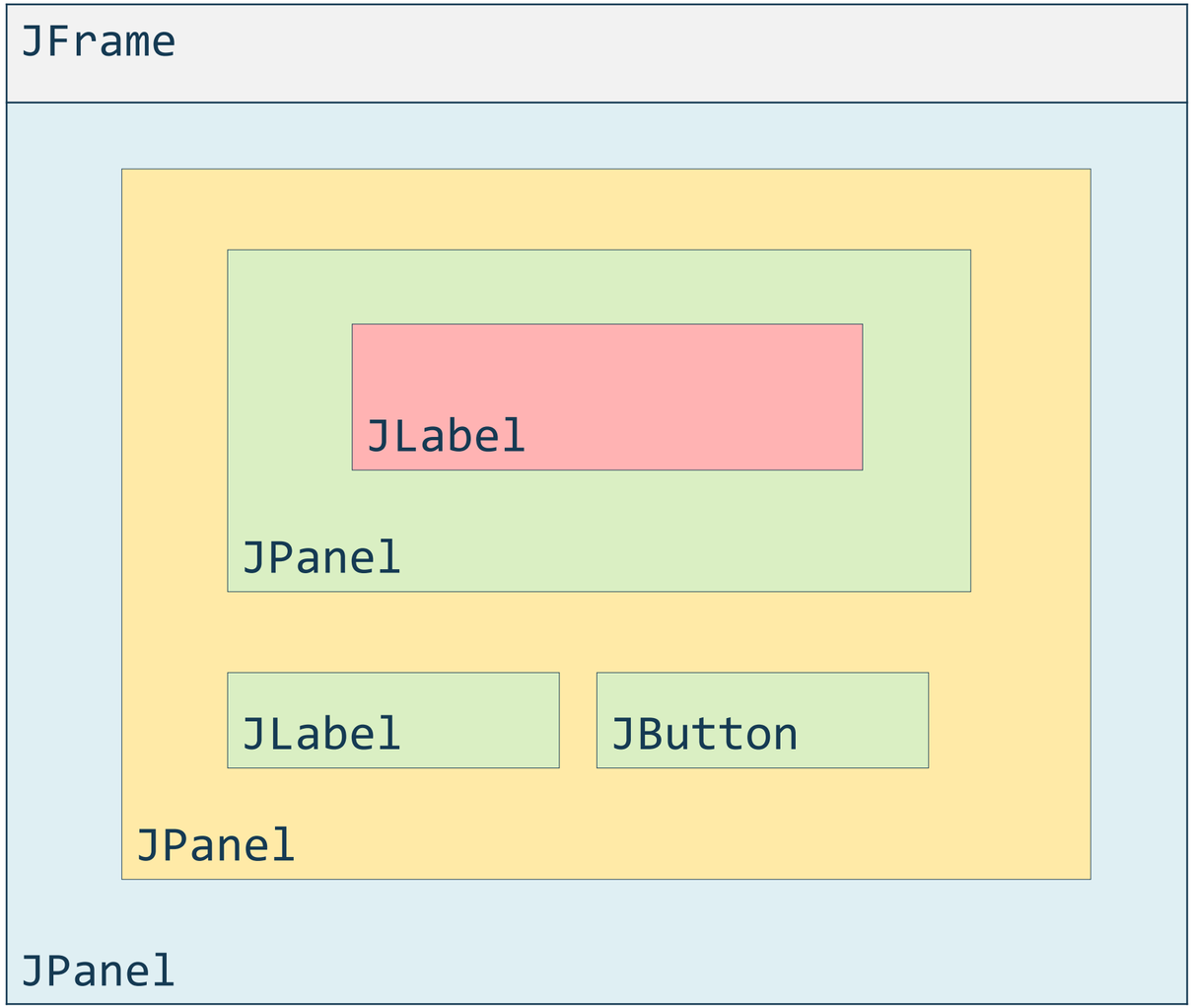
Containment hierarchy

To make a component visible, its containment hierarchy must be included into a `JFrame` *o another window object*

`JPanels` *are containers to which usually we add components*

Each component can belong to just one container

Other containers to which we add components are `JToolBar`, `JMenu`, *and* `JPopupMenu`



Swing windows

	JFrame	JDialog	JWindow
Title bar	*Yes*	*Yes*	*No*
Window buttons	*Minimize, maximize, and close*	*Close*	*None*
Border	*Yes*	*Yes*	*No*
Modal	*No*	*Yes*	*No*
Independent	*Yes*	*No*	*No*

A GUI application usually visualizes just one `JFrame` *instance*

- *When a frame is minimized, all the child dialogs and windows are minimized*
- *When a frame is disposed, all the child dialogs and windows are disposed*

Disposing windows

Windows (`JFrame`, `JDialog`, and `JWindow`) must be disposed after usage

```
public void dispose()
```

Releases all of the native screen resources used by this `Window`, its subcomponents, and all of its owned children. That is, the resources for these `Components` will be destroyed, any memory they consume will be returned to the OS, and they will be marked as undisplayable.

The `Window` and its subcomponents can be made displayable again by rebuilding the native resources with a subsequent call to `pack` or `show`. The states of the recreated `Window` and its subcomponents will be identical to the states of these objects at the point where the `Window` was disposed (not accounting for additional modifications between those actions).

Note: When the last displayable window within the Java virtual machine (VM) is disposed of, the VM may terminate. See AWT Threading Issues for more information.

Dispose vs hide

DisposedFrame.java

```
public class DisposedFrame {
    static void main() {
        SwingUtilities.invokeLater(DisposedFrame::disposeFrame);
    }

    private static void disposeFrame() {
        JFrame frame = new JFrame("A frame that will be disposed");
        frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        frame.setSize(400, 200);
        frame.setVisible(true);
    }
}
```

This program terminates

This program doesn't terminate

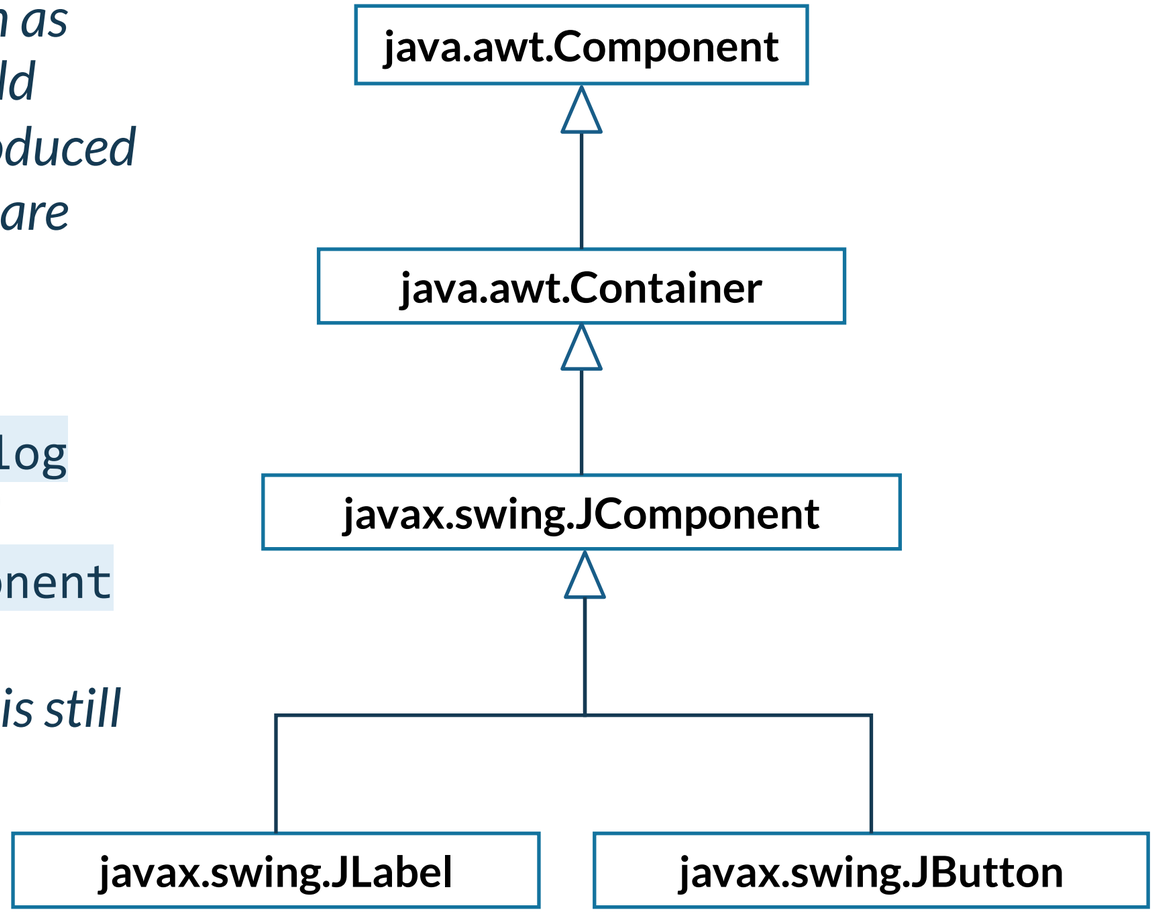
HiddenFrame.java

```
public class HiddenFrame {

    static void main() {
        SwingUtilities.invokeLater(HiddenFrame::hideFrame);
    }

    private static void hideFrame() {
        JFrame frame = new JFrame("A frame that will be hidden");
        frame.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
        frame.setSize(400, 200);
        frame.setVisible(true);
    }
}
```

Swing components and AWT

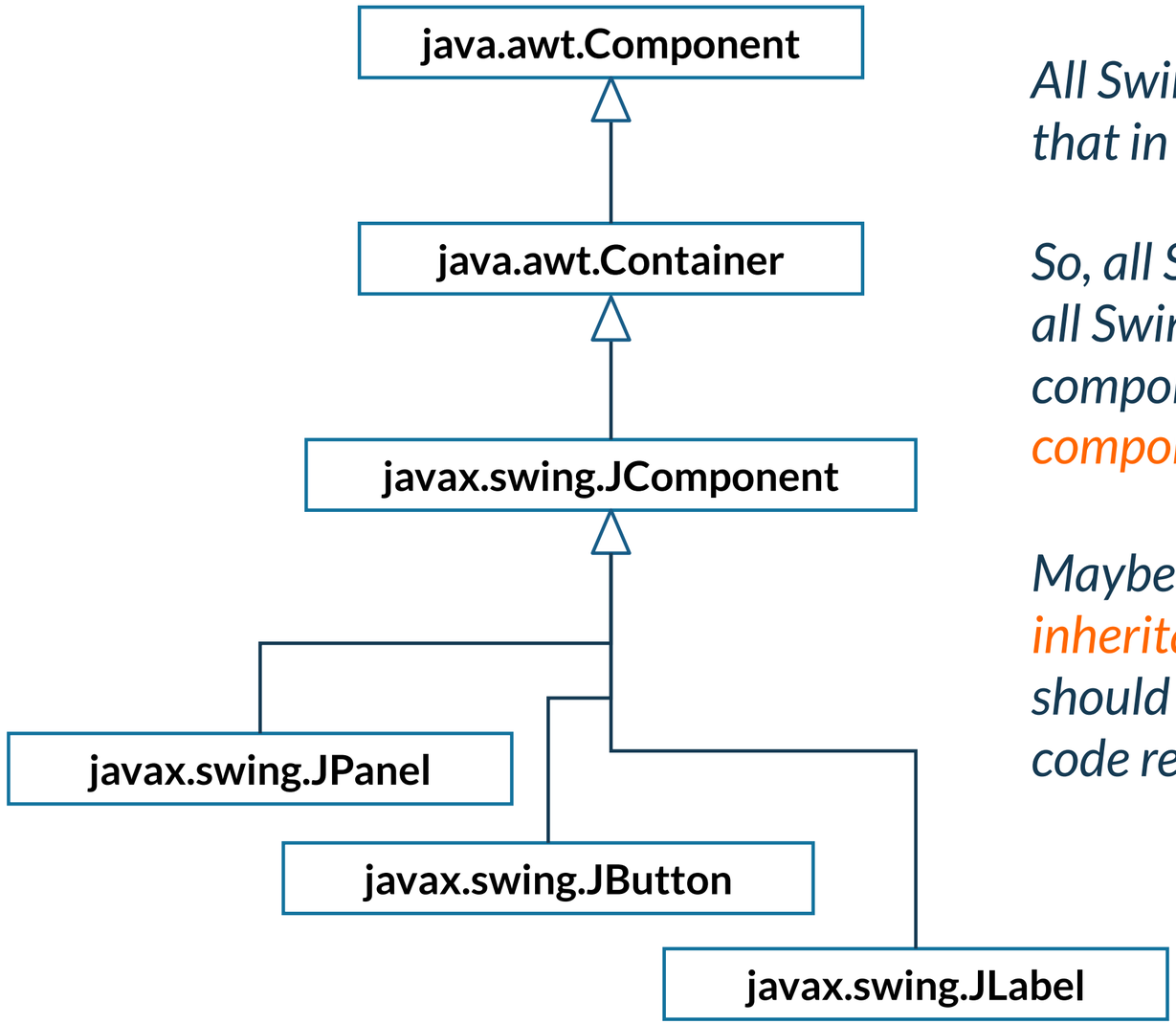
In Java Swing there are other windows classes, such as `Frame`, `Dialog`, *and* `Window`*; these are part of the old AWT library available since Java 1; Swing was introduced since Java 2; graphic classes without the 'J' in front are usually part of AWT and you should not use them*

Some Swing classes, like for example `JFrame`, `JDialog` *and* `JWindow` *still inherits from* `Frame`, `Dialog`, *and* `Window`*; All Swing components inherit from* `JComponent` *that inherit from* `Container` *that inherit from* `Component`*; the API of* `Container` *and* `Component` *is still widely used*



Assignment: explore the API of `Container`, `Component` *and* `JComponent`.

Inheritance hierarchy



All Swing components, inherits from `JComponent` *that in turn inherits from* `Container`

So, all Swing components are containers, but not all Swing components are meant to contain other components, e.g., is not appropriate to add a component to a `JButton`

Maybe this is not a very appropriate use of inheritance, but sometimes software engineers should accept trade-offs, in this case they traded code reuse with a "misuse" of inheritance

Remember - When to use inheritance

Both classes are in the same logical domain

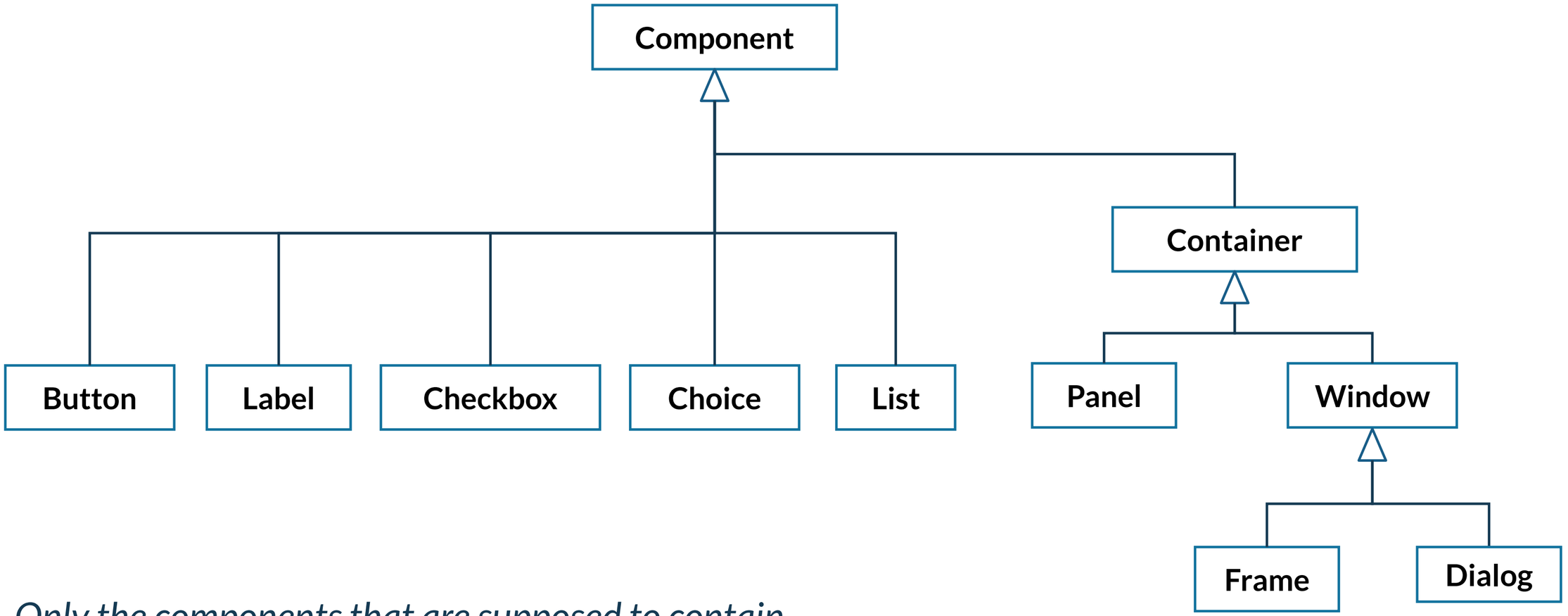
The subclass is a proper subtype of the superclass

The implementation of the superclass is necessary or appropriate for the subclass

The enhancements made by the subclass are primarily additive

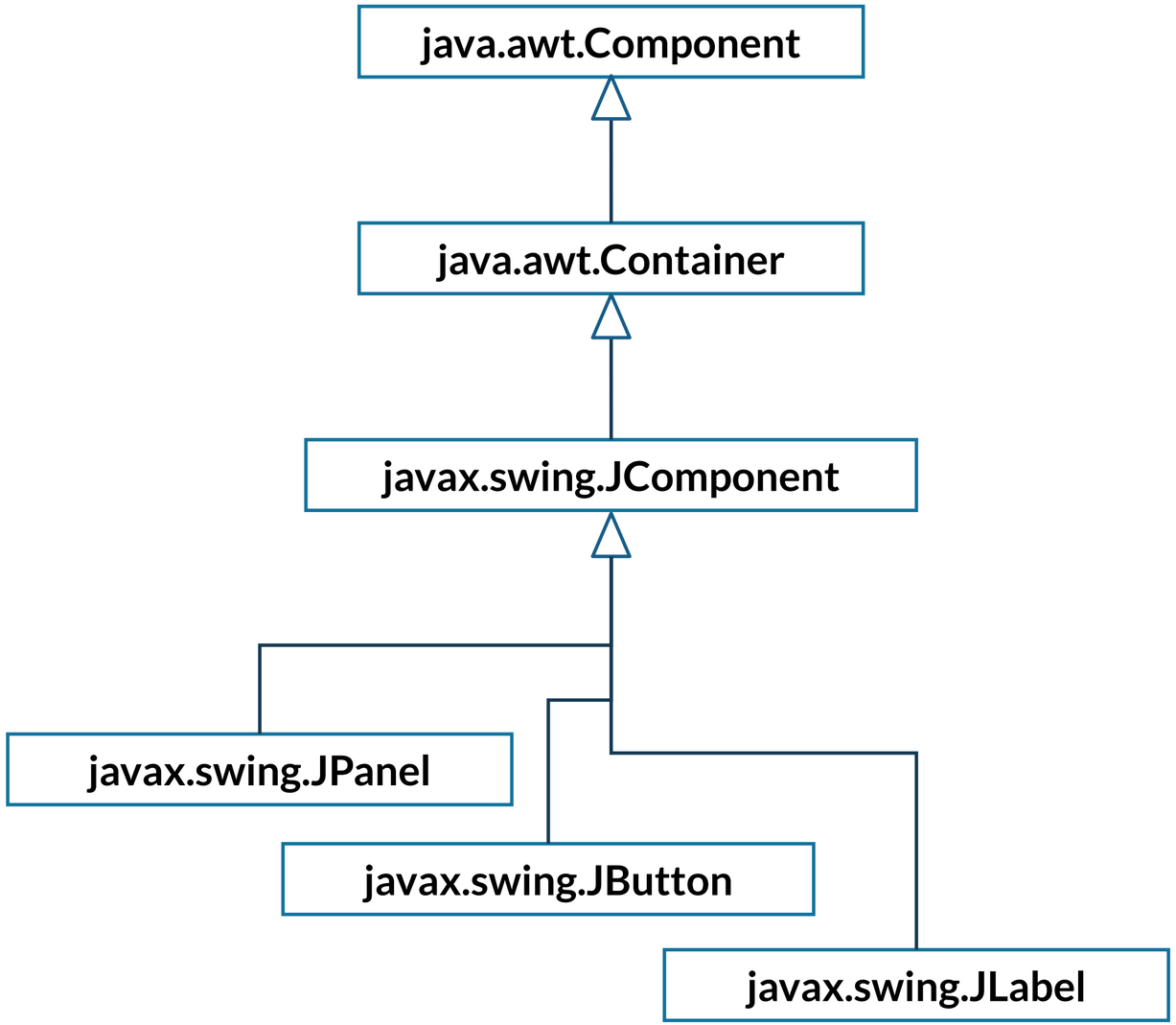
https://www.thoughtworks.com/insights/blog/composition-vs-inheritance-how-choose

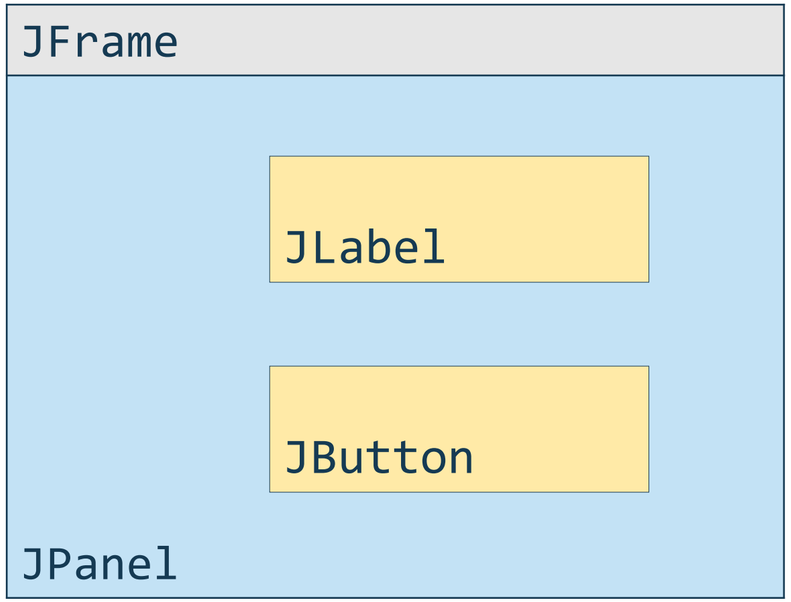
Digression - AWT inheritance hierarchy



Only the components that are supposed to contain other components are subclasses of `Container`

Inheritance vs containment hierarchy





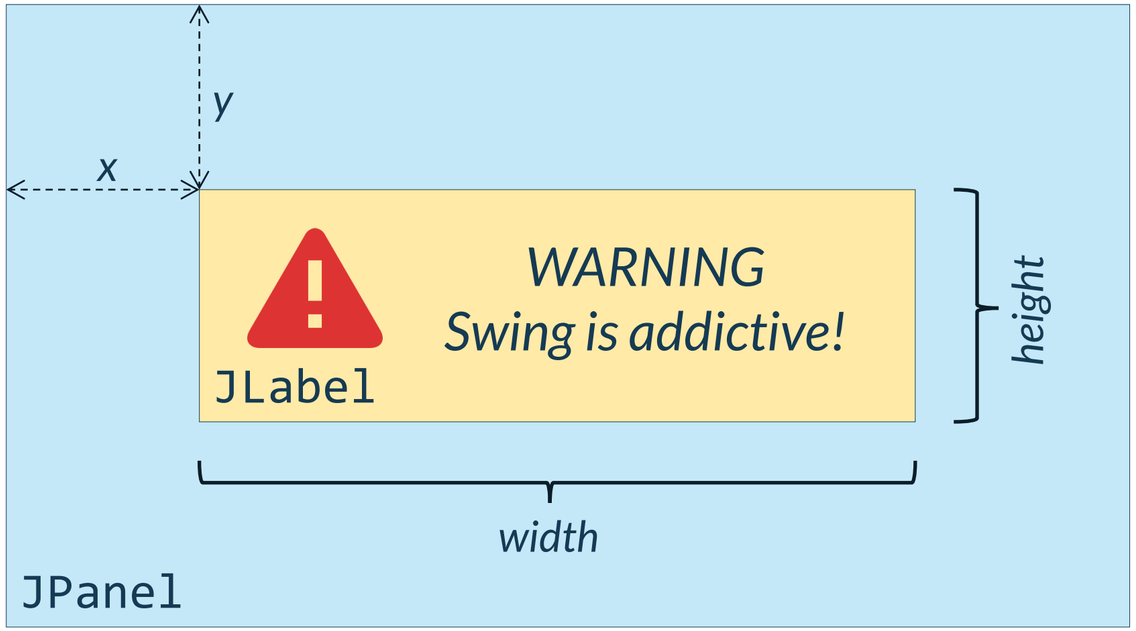
Label and button are child components of a panel

Don't get confused!

Exercises

1. *Modify the HelloWorld example to use a* `JDialog` *and a* `JWindow` *instead of a* `JFrame`
    1. *Explore how window closing works*
    2. *Explore how program termination works*
2. *Modify the Hello World example to open an "Hello, World!" popup (use both* `JDialog` *and* `JWindow`*) when pressing the button*
    1. *Explore how modality of* `JDialog` *works*
    2. *Explore window closing and program termination*

Almost “NO” fixed layout



The position, size and location, of a component is decided by the layout manager of its container

Each component is responsible to indicate its preferred, minimum and maximum sizes

Each Swing component knows how to calculate its preferred, minimum and maximum sizes

Each container has its own layout manager

A layout manager has two main responsibilities

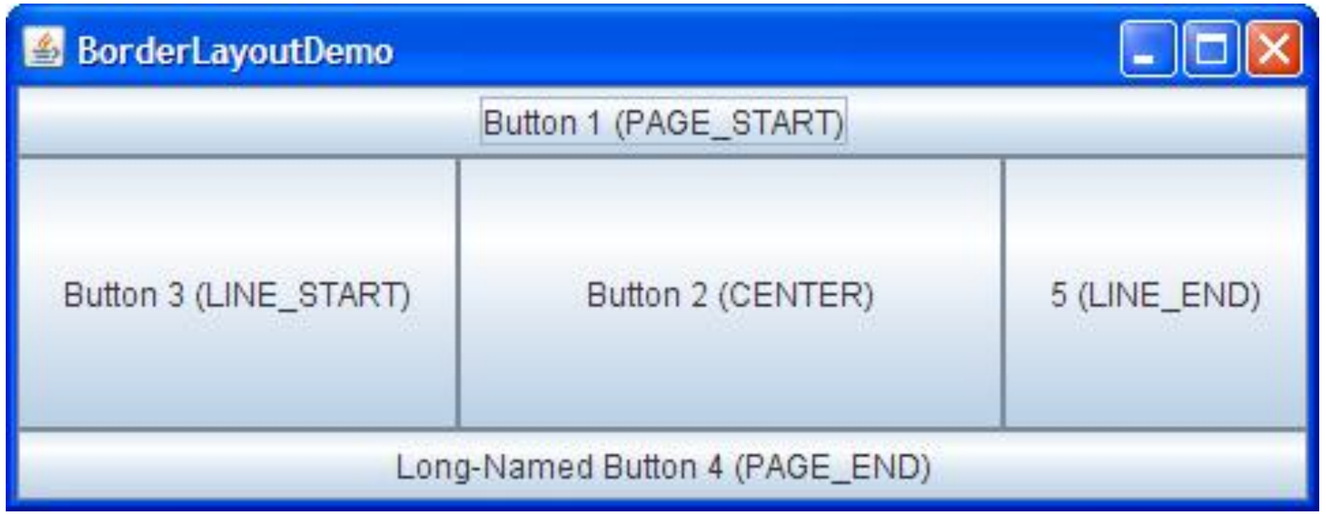
1. *layout the child components given their preferences and eventually a set of constraints*
2. *calculate the container preferred, minimum, and maximum sizes*

Since each container has its own layout manager, the process is “recursive”

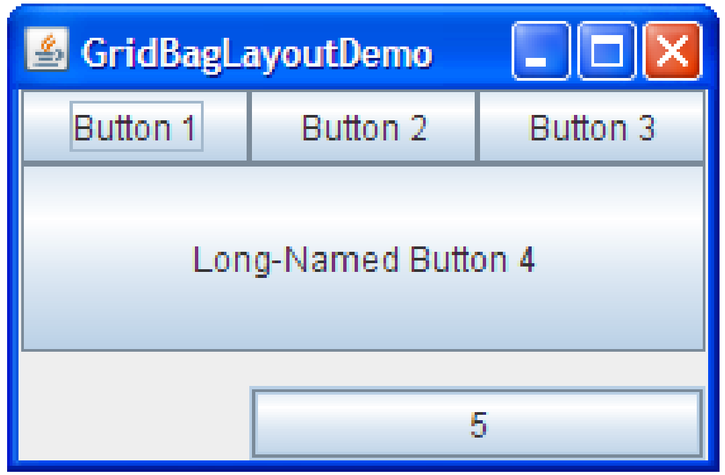
Layout managers

Common (my favorites) layout managers

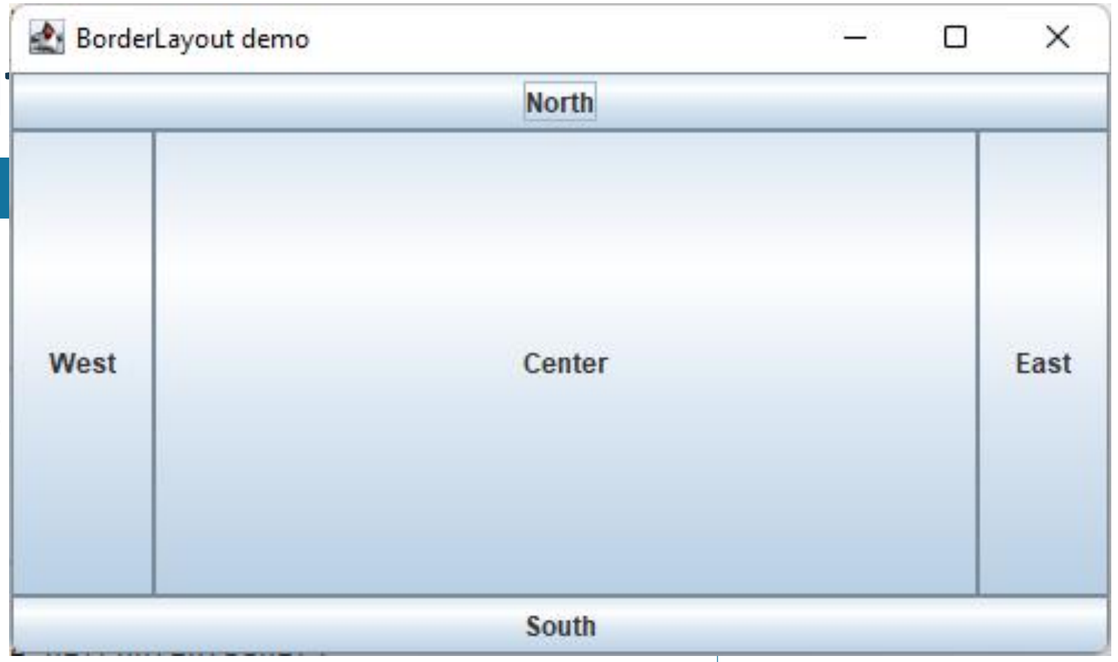
`BorderLayout`



`GridBagLayout`



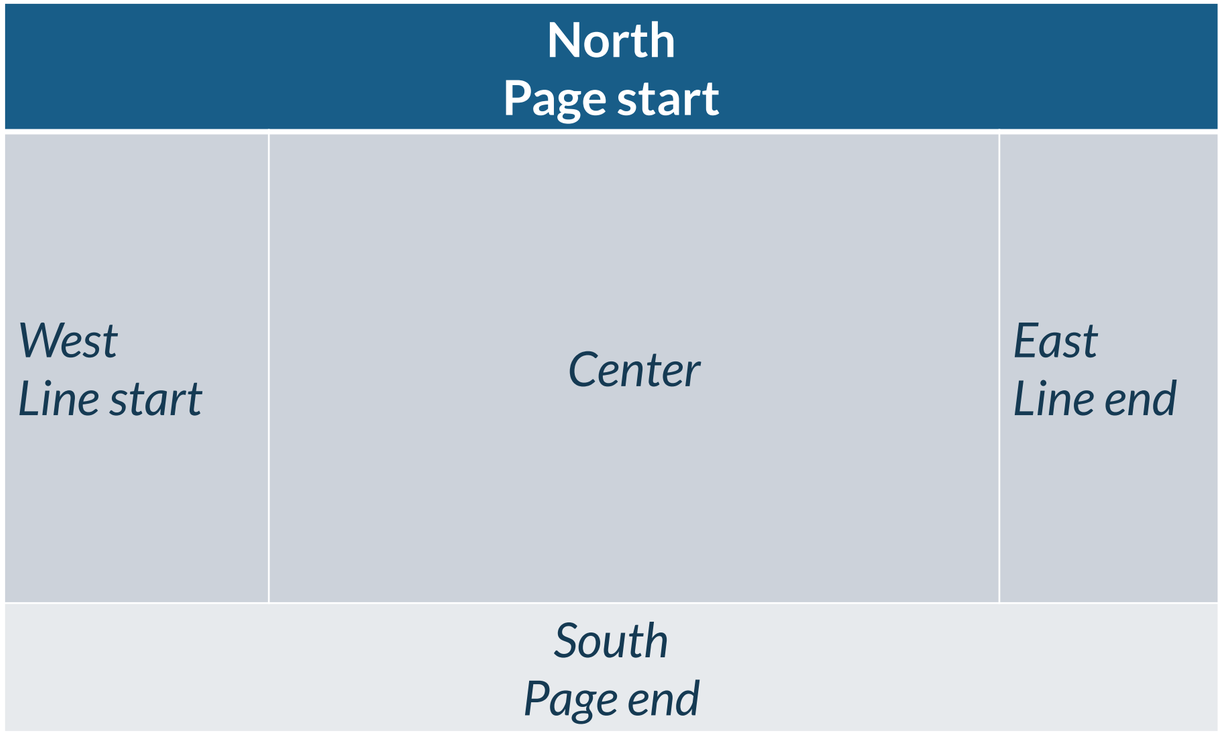
BorderLayou



BorderLayoutDemo.java

```
public class BorderLayoutDemo {

    static void main() {
        SwingUtilities.invokeLater(BorderLayoutDemo::run)
    }

    private static void run() {
        JFrame frame = new JFrame("BorderLayout demo");
        frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
        Container cp = frame.getContentPane();
        cp.setLayout(new BorderLayout());
        cp.add(new JButton("North"), BorderLayout.NORTH);
        cp.add(new JButton("South"), BorderLayout.SOUTH);
        cp.add(new JButton("East"), BorderLayout.EAST);
        cp.add(new JButton("West"), BorderLayout.WEST);
        cp.add(new JButton("Center"), BorderLayout.CENTER);
        frame.setSize(500, 400);
        frame.setVisible(true);
    }
}
```

BorderLayout



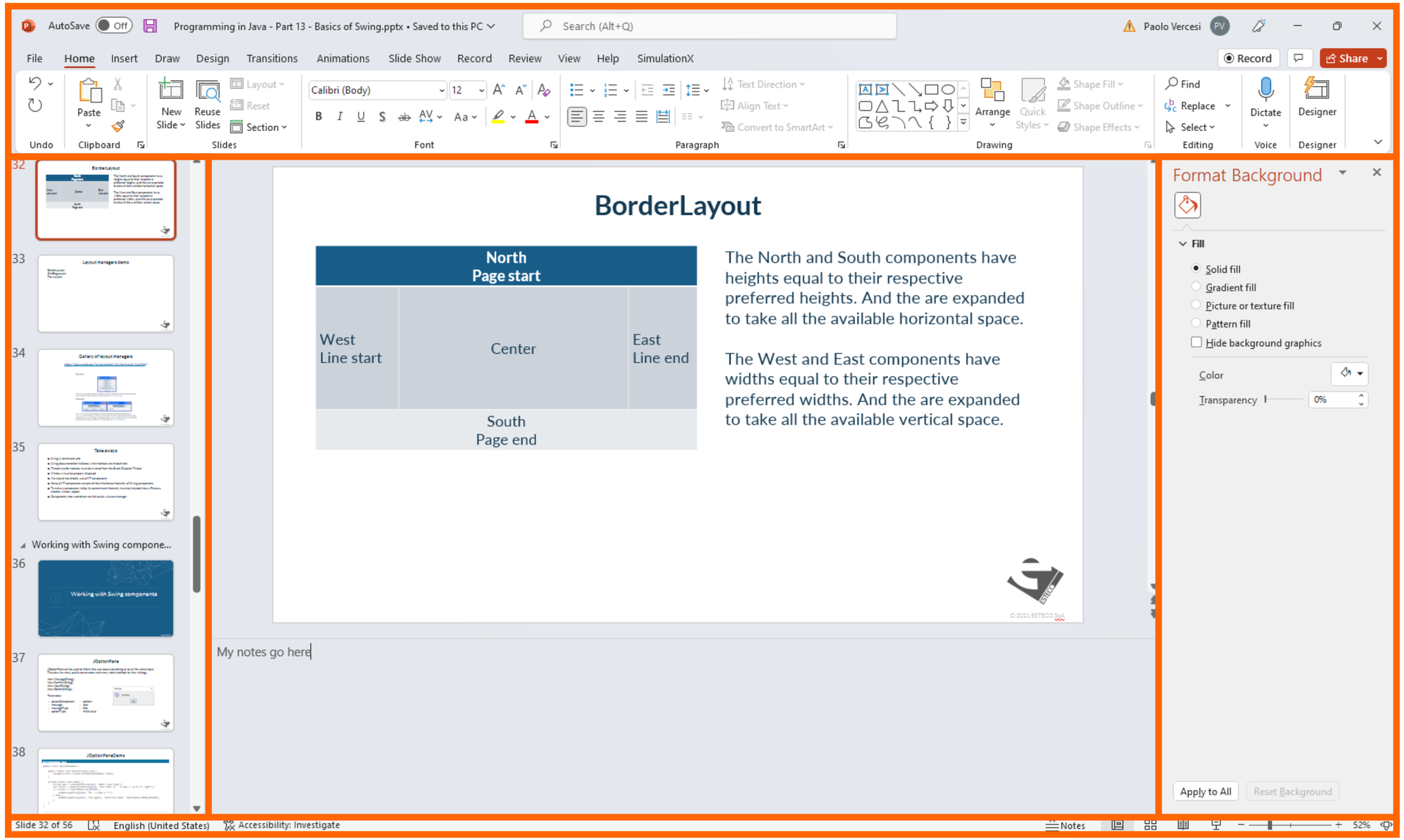
The maximum number of components is 5

The position of the component in the layout defines the constraints to which a component is subject

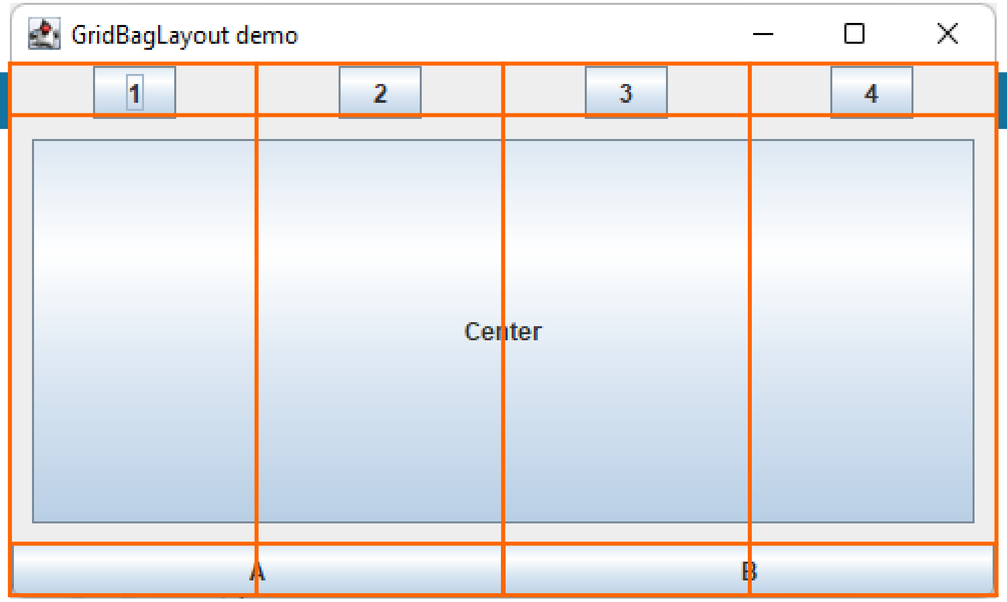
When using the `BorderLayout`

- *The North and South components have heights equal to their respective preferred heights; and they are expanded to take all the available horizontal space*
- *The West and East components have widths equal to their respective preferred widths; and they are expanded to take all the available vertical space*
- *The Center component takes all the available horizontal and vertical space*

Familiar enough!



GridBagLayout demo

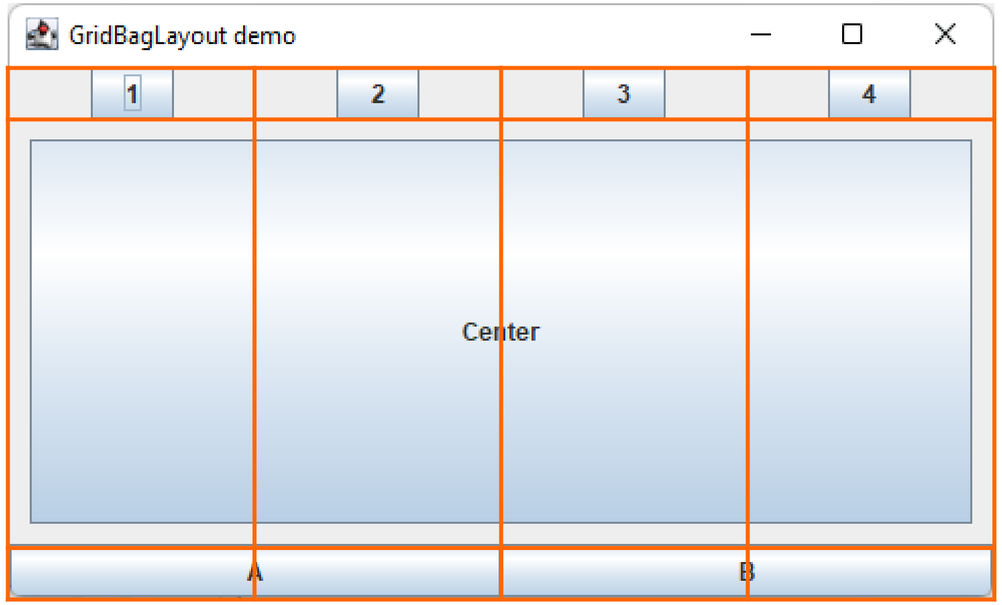


GridBagLayoutDemo.java

```
public class GridBagLayoutDemo {

  static void main() {
    SwingUtilities.invokeLater(GridBagLayoutDemo::run);
  }

  private static void run() {
    JFrame frame = new JFrame("GridBagLayout demo");
    frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
    Container cp = frame.getContentPane();
    cp.setLayout(new GridBagLayout());
    cp.add(new JButton("1"), new GridBagConstraints(0, 0, 1, 1, 1.0, 0.0, CENTER, NONE, new Insets(0, 0, 0, 0), 0, 0 ));
    cp.add(new JButton("2"), new GridBagConstraints(1, 0, 1, 1, 1.0, 0.0, CENTER, NONE, new Insets(0, 0, 0, 0), 0, 0 ));
    cp.add(new JButton("3"), new GridBagConstraints(2, 0, 1, 1, 1.0, 0.0, CENTER, NONE, new Insets(0, 0, 0, 0), 0, 0 ));
    cp.add(new JButton("4"), new GridBagConstraints(3, 0, 1, 1, 1.0, 0.0, CENTER, NONE, new Insets(0, 0, 0, 0), 0, 0 ));
    cp.add(new JButton("Center"), new GridBagConstraints(0, 1, 4, 1, 1, 1, CENTER, BOTH, new Insets(10, 10, 10, 10), 0, 0 ));
    cp.add(new JButton("A"), new GridBagConstraints(0, 2, 2, 1, 1.0, 0.0, CENTER, HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0 ));
    cp.add(new JButton("B"), new GridBagConstraints(2, 2, 2, 1, 1.0, 0.0, CENTER, HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0 ));
    frame.setSize(500, 300);
    frame.setVisible(true);
  }
}
```

x, y, width, height, weightx, weighty, anchor, fill, insets, padx, pady

GridBagLayout



The `GridBagLayout` *creates a "virtual" grid that can be extended indefinitely*

Each components is subject to many constraints

- `x`, `y` *position in the grid*
- `width`, `height` *horizontal and vertical span*
- `weightx`, `weighty` *define the weight of the corresponding columns (rows), Horizontal (vertical) extra space is assigned based to the column (row) weight. Define also how much horizontal (vertical) extra space is given to the component*
- `anchor` *how to position the component in the cell*
- `fill` *how to resize the component in the cell, depending on its weight*
- `insets` *how much space we should put around the component*
- `padx`, `pady` *internal padding of the component*

Exercise 1

Define the `GridBagConstraints` *that, when used with a* `GridBagLayout`*, produce the same effects of the five constraints of the* `BorderLayout`, `NORTH`, `WEST`, `CENTER`, `EAST`, `SOUTH`.

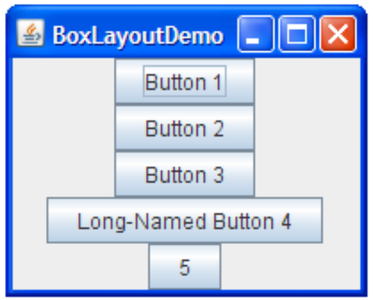
Exercise 2

Implement a “fixed” layout manager

Gallery of layout managers

https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html

BoxLayout



The `BoxLayout` class puts components in a single row or column. It respects the components' requested maximum sizes and also lets you align components. For further details, see How to Use BoxLayout.

CardLayout

The `CardLayout` class lets you implement an area that contains different components at different times. A `CardLayout` is often controlled by a combo box, with the state of the combo box determining which panel (group of components) the `CardLayout` displays. An alternative to using `CardLayout` is using a tabbed pane, which provides similar functionality but with a pre-defined GUI. For further details, see How to Use CardLayout.

Take aways

- *To make a component visible, its containment hierarchy must be included into a visible* `JFrame` *o another visible window object*
- *Swing provides three types of windows*
- *In general, an application has just one JFrame and it can have more instances of* `JDialog` *or* `JWindow`
- *We should not directly use AWT components, even if we still use AWT classes*
- *Windows must be properly disposed*
- *Most Swing components are subclasses of AWT components*
- *Components into a container are laid out by a layout manager*

Self assessment

Quiz 1: Hello, World!

1. **Why is object-oriented programming (OOP) well suited for GUI development?**
   A. *Because GUIs are purely procedural*
   B. *Because GUI components are natural objects*
   C. *Because Swing doesn't use objects*
   D. *Because GUIs don't need inheritance*
2. **Which Java library provides the standard GUI framework discussed in this lecture?**
   A. *JavaFX*
   B. *AWT*
   C. *SWT*
   D. *Swing*
3. **What is the responsibility of the main method in a GUI program?**
   A. *Initialize and assemble the GUI and application logic*
   B. *Handle all events*
   C. *Create only the application logic*
   D. *Control operating system windows*

Quiz 2: The rules of the game

1. **Which class represents a top-level window in Swing?**
   A. `JPanel`
   B. `JFrame`
   C. `JLabel`
   D. `JButton`

2. **How do you make a Swing component visible?**
   A. Call `setVisible(true)` on it directly
   B. Register an `ActionListener`
   C. Add it to a visible container such as the content pane of a `JFrame`
   D. Compile the program

3. **What is a containment hierarchy?**
   A. The tree of components and containers that define what's visible
   B. A list of all imports
   C. A thread pool for events
   D. A debugging structure

Quiz 2: The rules of the game

4. **Which statement about Swing windows is true?**
   A. *A program can have many visible JFrames simultaneously by default*
   B. *Typically a GUI app has one main* `JFrame` *and multiple dialogs*
   C. `JFrame` *cannot have child dialogs*
   D. *Disposing a* `JFrame` *doesn't affect child windows*

5. **What is the difference between the** `dispose()` **and** `hide()` **methods of Window?**
   A. *None*
   B. `hide()` *releases native resources,* `dispose()` *just makes invisible*
   C. `dispose()` *pauses the program,* `hide()` *terminates the JVM*
   D. `dispose()` *releases native resources,* `hide()` *just makes invisible*

6. **Which is the direct superclass all Swing components inherit from?**
   A. `Container`
   B. `JComponent`
   C. `Window`
   D. `Component`

Quiz 2: The rules of the game

7. **Which is not recommended to use directly in Swing apps?**
   A. *AWT components (without the "J" prefix)*
   B. `JPanel`
   C. `JLabel`
   D. `JButton`

8. **What is the main responsibility of a layout manager?**
   A. *Handle user input to layout child components*
   B. *Dispatch events*
   C. *Manage multithreading*
   D. *Arrange child components based on constraints and preferences*

9. **Which layout expands the center component to fill all available space?**
   A. `FlowLayout`
   B. `GridLayout`
   C. `BorderLayout`
   D. `BoxLayout`

Quiz 2: The rules of the game

10. Why must GUI updates run on the Event Dispatch Thread (EDT)?

A. *To ensure correct event ordering and thread safety*
B. *Because most Swing methods are not thread safe*
C. *Because it improves rendering speed*
D. *Both A and B*

11. What happens if long tasks run on the EDT?

A. *The JVM crashes*
B. *The UI becomes unresponsive, and events queue up*
C. *Nothing — Swing handles it automatically*
D. *Only repainting stops temporarily*

Correct answers

Quiz 1

1. *B*
2. *D*
3. *A*
9. *C*
10. *D*
11. *B*

Quiz 2

1. *B*
2. *C*
3. *A*
4. *A*
5. *D*
6. *D*
7. *A*
8. *D*

Thank you!

esteco.com