Unit 10:

GUI 1 - Composition and Layout

Object-Oriented Programming (OOP)
CCIT 4023, 2025-2026

U10: GUI 1 - Composition and Layout

- Introduction to Graphical User Interface (GUI)
- GUI Composition and Layout
  - Constructing GUI Window – **`JFrame`**
  - Adding GUI Components on Containers – **`JComponent`**
- Layout Manager – Flow, Border, Grid
- Common GUI Components
  - A Useful Container: **`Jpanel`**
  - **`JTextField`**, **`JLabel`** and **`JTextArea`**

Introduction to Graphical User Interface (GUI)

- Design of the API for Java GUI (Graphical User Interface) programming is an excellent example of how the object-oriented principle is applied
- We use the GUI components to develop user-friendly interfaces for applications and applets, compared to the CLI (Command Line Interface)
  - A Java applet is a special kind of Java “mini” application program that is typically embedded inside a web page and runs in the context of a web browser
- Since the GUI components in Java are very flexible and versatile, you can create a wide assortment of useful user interfaces

Introduction to Graphical User Interface (GUI)
(Swing and AWT packages in Java)

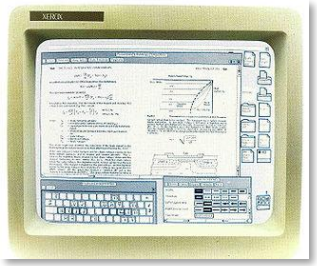
- In Java, GUI-based programs could be implemented by using classes from `javax.swing` and `java.awt` packages, often work in a concurrent way
  - Many of the original AWT classes are replaced by Swing classes
- Swing classes provide greater compatibility across different operating systems
  - They are fully implemented in Java, and behave the same on different operating systems
  - They support many new functionalities not supported by the *AWT*

* Remarks: In versions JDK 8,9,10, JavaFX is included for supporting better GUI programming. However, starting with JDK 11 JavaFX is decoupled from the JDK.

Introduction to Graphical User Interface (GUI)
(A Brief History)

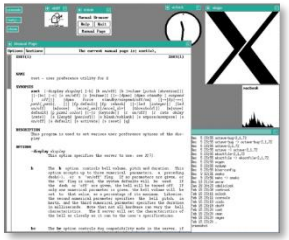
Reference Only

Early computer was operated with CLI (Command Line Interface)



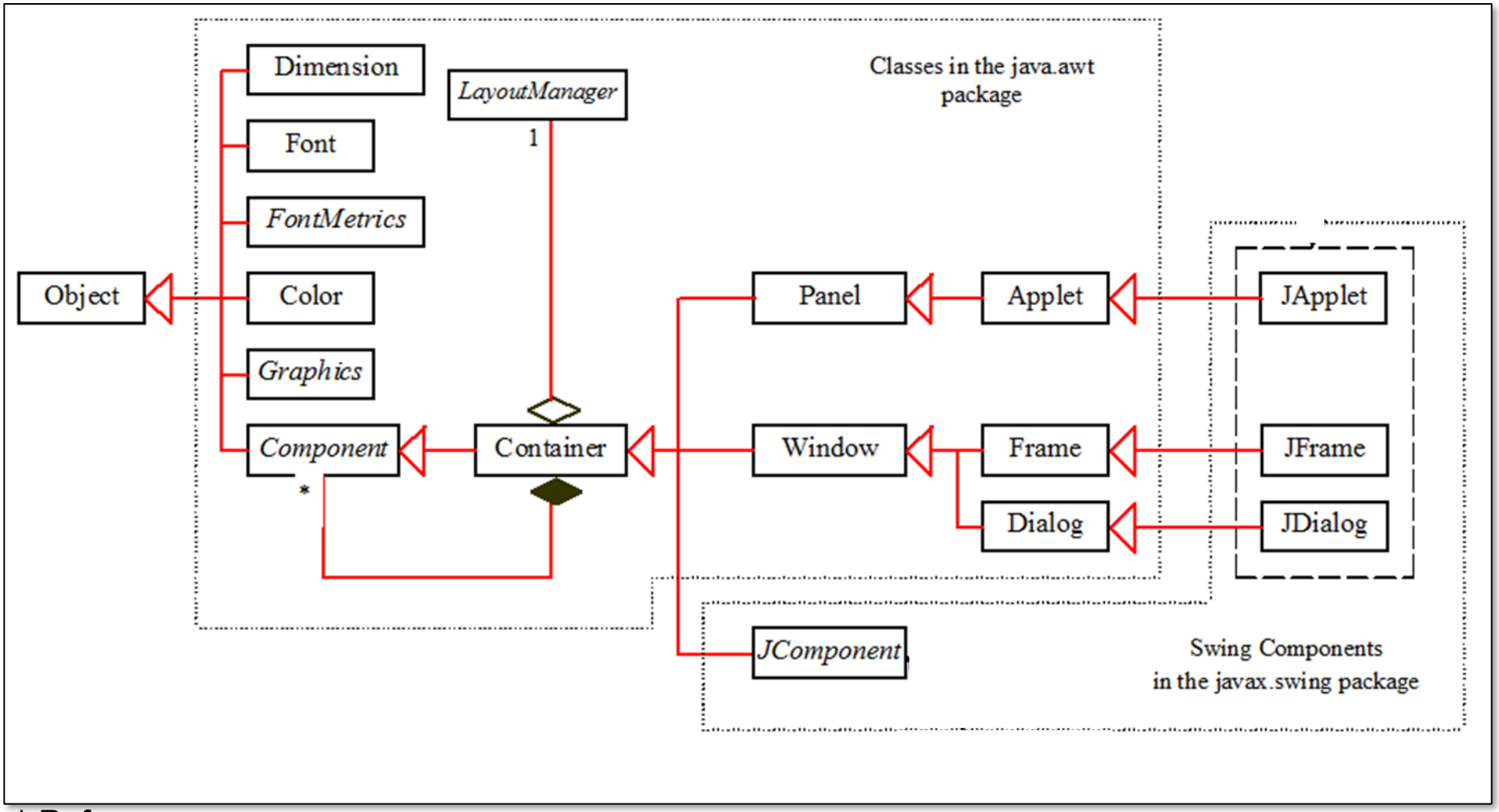
Xerox Star Workstation : the very early commercial GUI-based operating systems (~1981)

An early Unix desktop running the GUI-based X Window System (~1984)



Macintosh 128K was the first commercially successful personal computer to use a GUI (~1984)

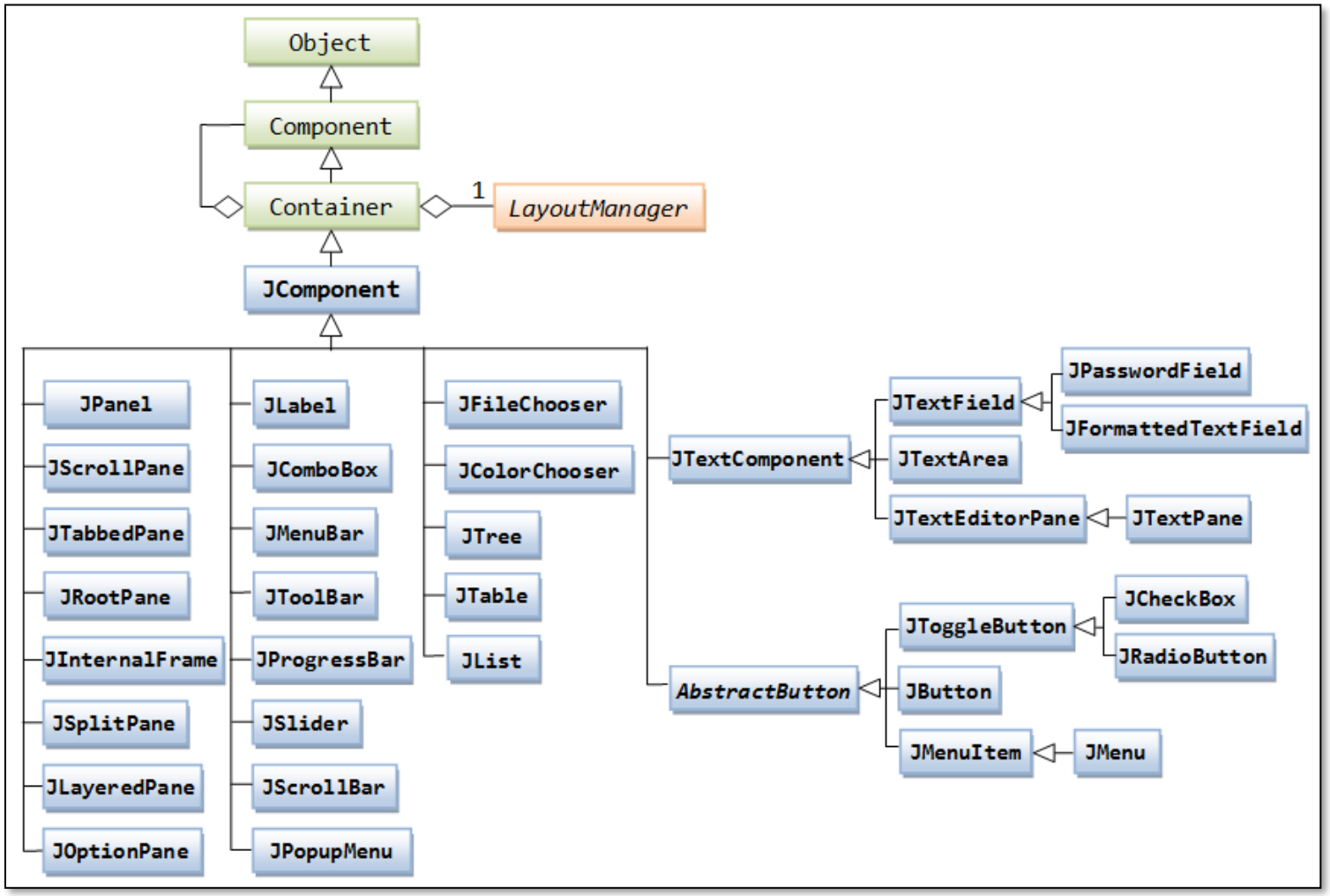
GUI Class Hierarchy



** Reference:*

- *Swing application (e.g. with `JFrame`) has its own single thread, called Event Dispatch Thread.*
- *That is why the application may not terminate, even the `main()` method ends.*

Subclasses of `JComponent` Class



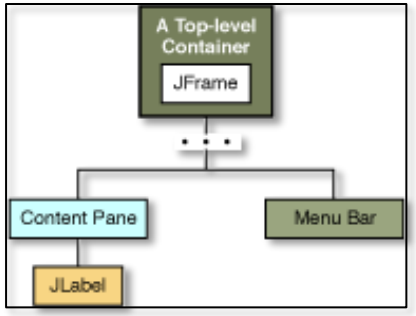
** Reference: Useful websites about Swing and JComponents*

- https://docs.oracle.com/javase/tutorial/uiswing/index.html
- https://docs.oracle.com/javase/tutorial/uiswing/components/index.html

GUI Composition and Layout

- We could only use `JOptionPane` dialogs for a very simple GUI application
- More often we need to develop an application as an integrated workspace GUI window
- In general, designing and developing GUI applications involves issues on:
  - visual design, including what GUI components we use and how to compose them with proper layout
    - We often use the components in `javax.swing` and `java.awt` packages
  - user interaction design, including how components respond to different user actions
    - Event Handling: handling user action event of some GUI components for user input

GUI Composition and Layout



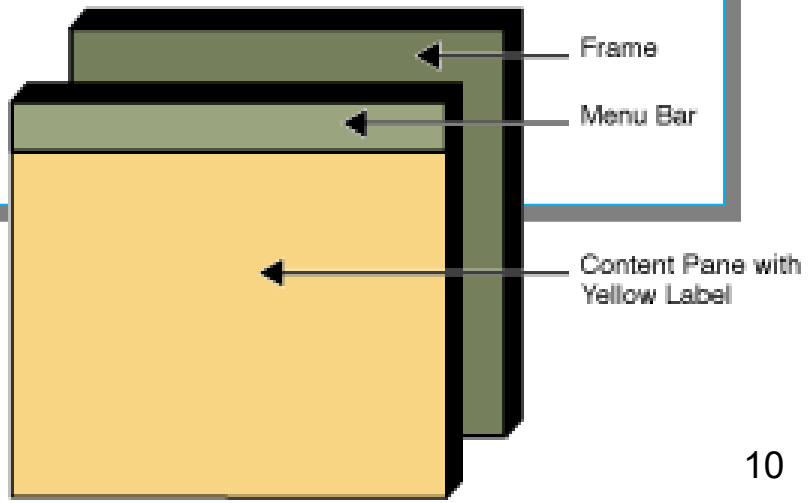
Typical GUI application includes

- A Top-Level Container (e.g. `JFrame`),
  - Sets its basic properties (e.g. Title, size)
  - Sets with particular layout manager (e.g. `FlowLayout`), to specify how to add/lay other GUI components on
  - Adds other containers (e.g. `JPanel`) or components (e.g. `JLabel`, `JButton`, some respond to user input)
- A set of components (`JComponents`), which
  - Are added in the Top-Level Container, or other containers (e.g. `JPanel`) in the "containment hierarchy"
  - Register event listeners, if they are event sources such as `JButton` responding to user input (button click)

Constructing GUI Windows

- Class `JFrame` of the `javax.swing` package is an indirect subclass of class `java.awt.Window` that is most often used to provide basic attributes and behaviors of an application window
  - a title bar at the top, buttons to minimize, maximize and close the window

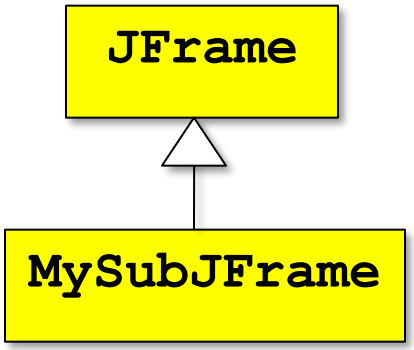
```
import javax.swing.JFrame;
public class MyJFrameApp {
    public static void main( String[] args ) {
        JFrame appJFrame;
        appJFrame = new JFrame("A JFrame App");
        // . . .
    }
}
```



Creating a Subclass of `JFrame`

- We seldom directly create an instance of `JFrame` in most applications, instead we often define a subclass of `JFrame` for specific tasks
- It is more often to define a *subclass* of the `JFrame` class, as the example below
- To define a subclass of another class, we declare the subclass with the reserved word **`extends`**

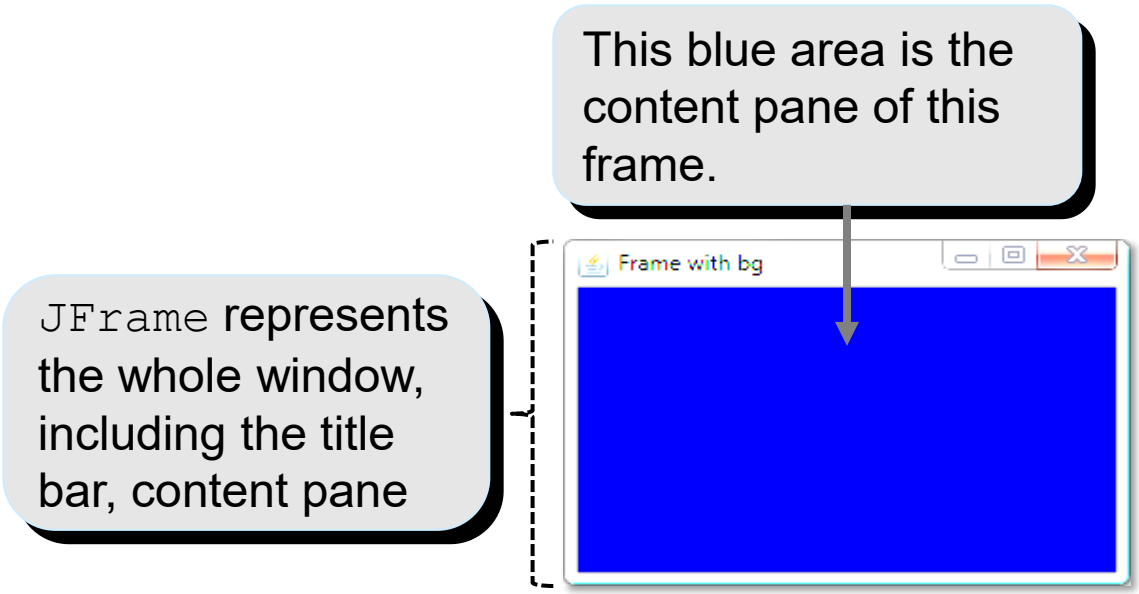
```
import javax.swing.*;
public class MySubJFrame extends JFrame {
 // Fields
    JButton okButton, cancelButton;
    JLabel statusLabel;
    // ...
}
```



Adding GUI Components on Containers

- The *content pane* (a `Container` object) is where we add/ put GUI objects such as buttons, labels.
  - We do not directly add our GUI objects to `JFrame`
- Access the content pane by calling the frame's `getContentPane()` method to return an object of the `Container` class
  - E.g. set its background color:

```
getContentPane().setBackground(java.awt.Color.BLUE);
```

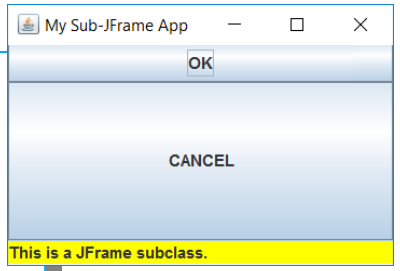


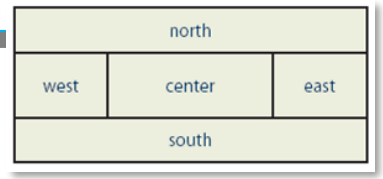
Adding Components: Button and Label

- Add Component to content pane of `JFrame` with method **`add()`**. E.g.
  - Add two **`JButton`** objects, GUI component that represents push-buttons
    - One on top and one on center of the `JFrame`
  - Add a JLable object, GUI component showing information (e.g. text)
  - Here's an example of how we place two buttons and one label to `JFrame`, with the support of its default `java.awt.BorderLayout` manager

* Remark: As a conveniance `add` and its variants, `remove` and `setLayout` have been overridden to forward to the `contentPane` as necessary. This means you can write `frame.add(child);` and the child will be added to its `contentPane`.

```
okButton = new JButton("OK");
cancelButton = new JButton("CANCEL");
statusLabel = new JLabel("This is a JFrame subclass.");
getContentPane().add(okButton, BorderLayout.NORTH);
getContentPane().add(cancelButton, BorderLayout.CENTER);
getContentPane().add(statusLabel, BorderLayout.SOUTH);
```



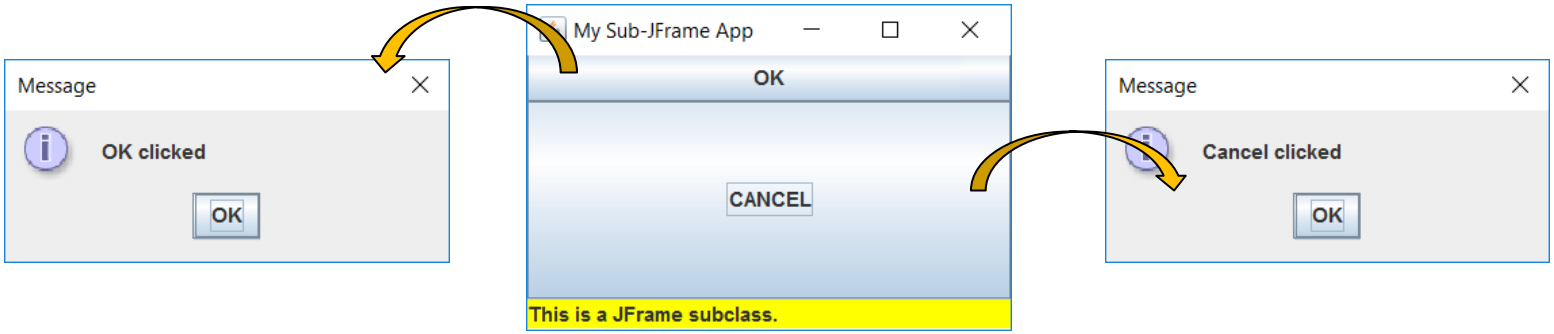


Simple Button Event Handling

- For GUI components with user interaction, such as buttons in our example, we may write codes to respond user action (event handling)
- The following shows simple Button action event handling (with Lambda Expression), showing text with `JOptionPane` *only for demonstration*:

```
okButton.addActionListener( e -> { //button click jumps here
        ////// code to let button-click respond HERE  //////
    JOptionPane.showMessageDialog(null, "OK clicked");
} );
cancelButton.addActionListener( e -> { //button click jumps here
        ////// code to let button-click respond HERE  //////
    JOptionPane.showMessageDialog(null, "Cancel clicked");
} );
```

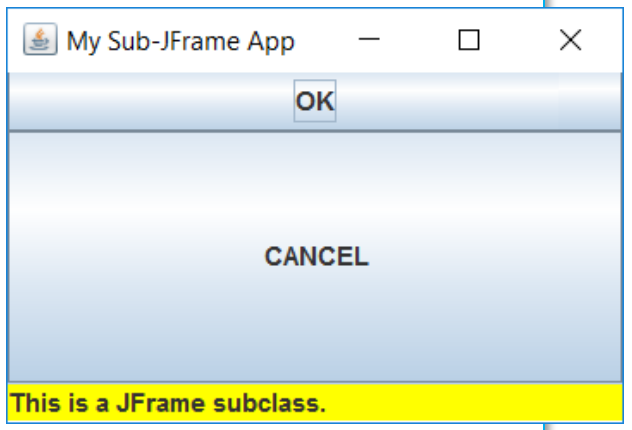
** Concept details of event handling will be discussed in later unit.*



Code Sample

MySubJFrame.java

```
// MySubJFrame.java; OOP
import javax.swing.*;
import java.awt.*;
public class MySubJFrame extends JFrame {
    JButton okButton, cancelButton;
    JLabel statusLabel;
    public MySubJFrame(String title) { // constructor
        super(title);
        setSize(300, 200);   setLocation(150, 250);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        compositeGUI();
    }
    public void compositeGUI(){ // method for composing GUI
        getContentPane().setBackground(Color.YELLOW);
        statusLabel = new JLabel("This is a JFrame subclass.");
        okButton = new JButton("OK");
        cancelButton = new JButton("CANCEL");
        getContentPane().add(okButton, BorderLayout.NORTH);
        getContentPane().add(cancelButton, BorderLayout.CENTER);
        getContentPane().add(statusLabel, BorderLayout.SOUTH);
        okButton.addActionListener( e -> { //button click jumps here
            JOptionPane.showMessageDialog(null, "OK clicked");
        } );
        cancelButton.addActionListener( e -> { //button click jumps here
            JOptionPane.showMessageDialog(null, "Cancel clicked");
        } );
    }
    public static void main(String[] args) {
        MySubJFrame mySF = new MySubJFrame("My Sub-JFrame App");
        mySF.setVisible(true);
    }
}
```



Layout Manager – Flow, Border, Grid

- When developing application with GUI interface, often we attach GUI components to a container, such as the content pane of `JFrame` or `JPanel`
- We also decide how to *position these* GUI components and specify the layout
- Java provides several layout managers (in package `java.awt`), e.g.
  - `FlowLayout`
  - `BorderLayout`
  - `GridLayout`

Setting Layout for ContentPane (of a Frame)

- Sample code of setting Layout Manager (often before adding elements), for `JFrame` (its content pane):

```
getContentPane().setLayout(new FlowLayout());

getContentPane().add(scrollText);
```

- Even though it is rare and a bit complicated, we may use *absolute positioning with* `null` layout manager

```
getContentPane().setLayout(null);
```

- Content panes (of `JFrame`) use `BorderLayout` by default, while `JPanel` object uses `FlowLayout` if not specified

FlowLayout

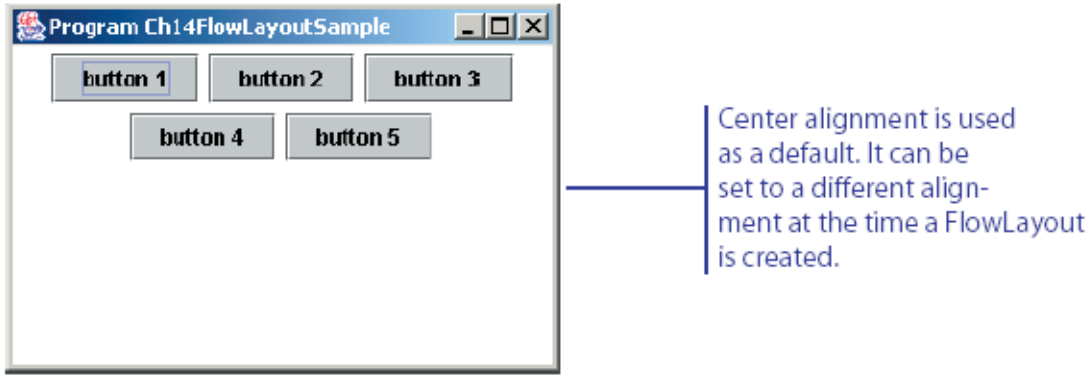
- In using this layout, GUI components are placed in left-to-right order
  - When the components do not fit on the same line, left-to-right placement continues on the next line
- By default, components on each line are centered
- When the frame containing the component is resized, the placement of components is adjusted accordingly
- Sample Code, in a `JFrame` class:

```
getContentPane().setLayout(new FlowLayout());
```

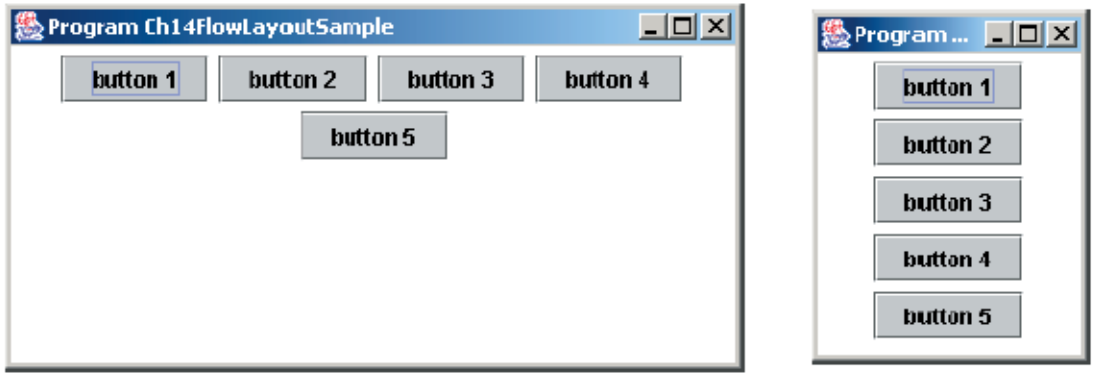
** By default, `JPanel` has a `FlowLayout` manager*

FlowLayout Sample

- The placement of five buttons by using `FlowLayout`



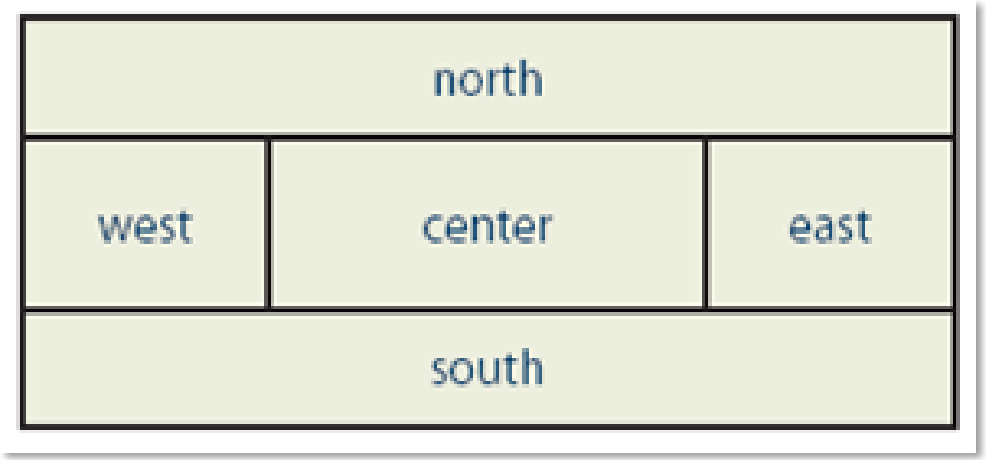
When the frame first appears on the screen.



After the frame's width is widened and shortened.

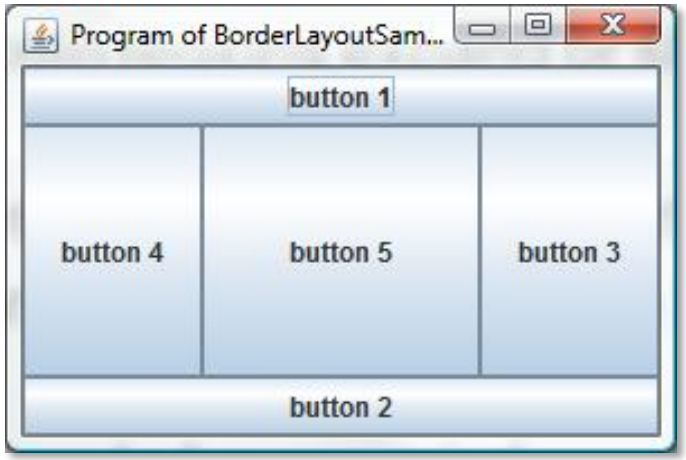
BorderLayout

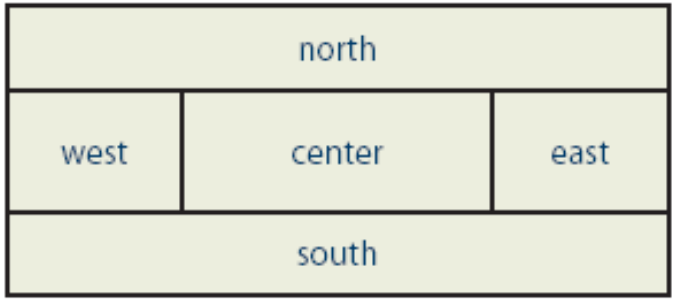
- This layout manager divides the container into five regions: center, north, south, east, and west
  - North and south regions expand or shrink in height only
  - East and west regions expand or shrink in width only
  - Center region expands or shrinks on both height & width
  - Not all regions have to be occupied
- By default, `JFrame` has a `BorderLayout` manager



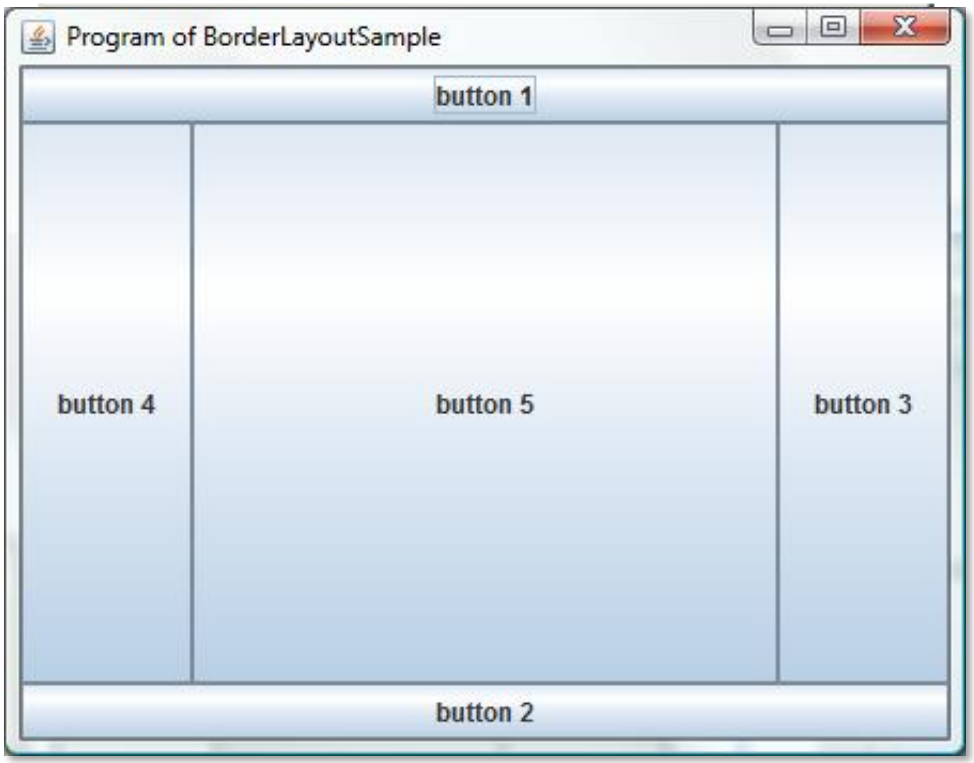
BorderLayout Sample

When the frame first appears on the screen.





After the frame is resized.



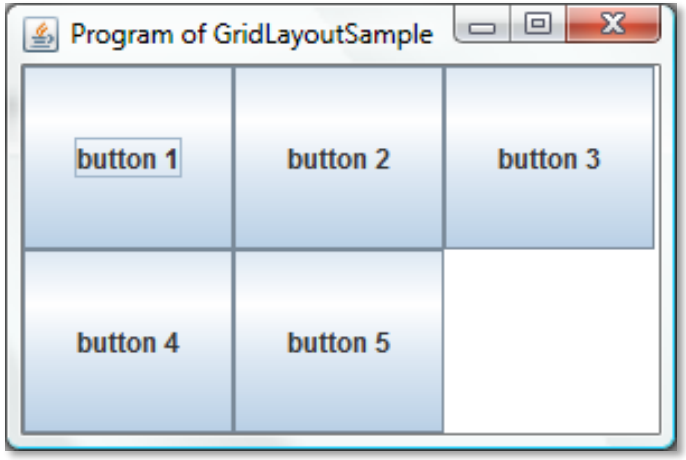
E.g.: `contentPane.add(button1, BorderLayout.NORTH);`

GridLayout

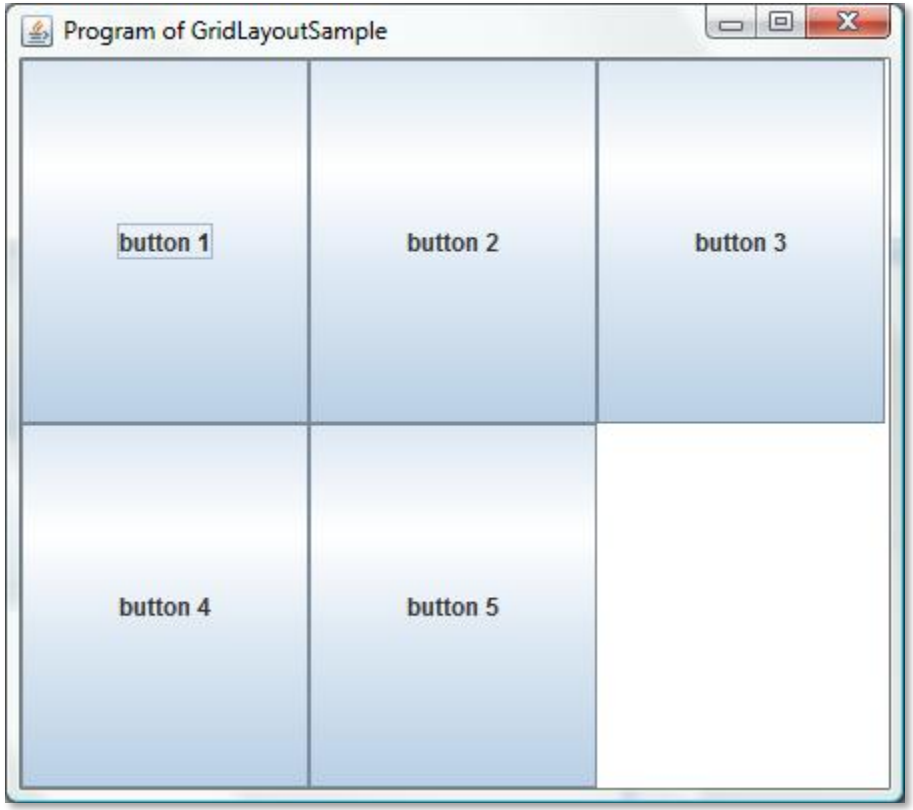
- This layout manager places GUI components on equal-size `N` by `M` grids
- Components are placed in left-to-right, and top-to-bottom order
- The number of rows and columns remains the same after the frame is resized, but the width and height of each region will be changed

GridLayout Sample

When the frame first appears on the screen.



After the frame is resized.

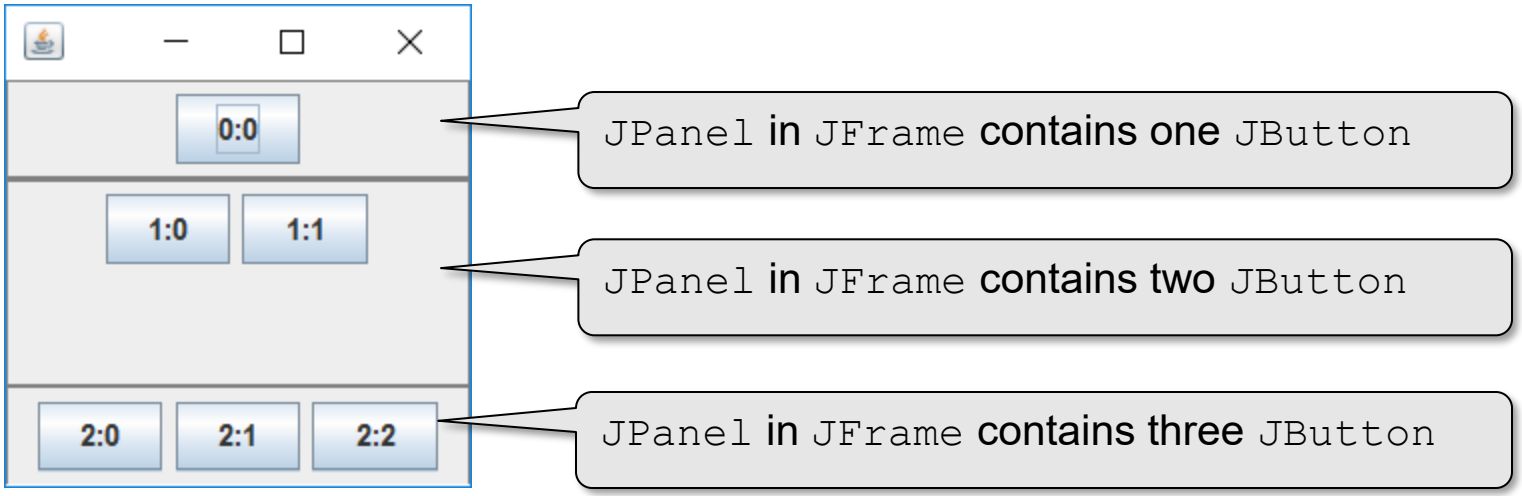


Common GUI Components

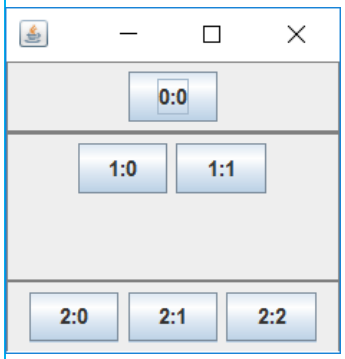
- There are different Swing GUI classes
- Often we need a contain class which can contain other components, e.g. we want to put a number of buttons / icons to form a tool bar. In this case the `JPanel` class supports such a purpose to contain other components.
- Swing GUI classes `JTextField`, `JLabel` and `JTextArea` deal with text
  - A `JTextField` object allows the user to enter a single line of text
  - A `JLabel` object displays uneditable text (or image)
  - A `JTextArea` object allows the user to enter multiple lines of text. It can also be used for displaying multiple lines of uneditable text

A Useful Container: `JPanel`

- `JPanel` class provides general-purpose containers for lightweight components
  - May use `JPanel` in the similar way as the content pane of `JFrame`
  - A panel uses a layout manager to position and size its components
    * *Remark: By default, `JPanel` has a `FlowLayout` manager*
- Use `JPanel` for more complicated GUI, e.g.
  - The following sample codes add 3 `JPanel` in `JFrame` (content pane), where each `JPanel` would then contain its own component(s) (button(s)).



Sample Codes of Using `JPanel`




```
// DemoJPanelSF.java; OOP
import javax.swing.*;
import java.awt.*;
public class DemoJPanelSF extends JFrame {
        JPanel [] someJP = new JPanel[3];
        DemoJPanelSF() {
                for (int i=0; i<someJP.length; i++){
                        someJP[i] = new JPanel();
                        someJP[i].setBorder(
                           BorderFactory.createLineBorder(Color.GRAY));
                        for (int j=0; j<=i; j++){
                                someJP[i].add(new JButton(i+":"+j));
                        }
                }
            getContentPane().add(someJP[0], BorderLayout.NORTH);
            getContentPane().add(someJP[1], BorderLayout.CENTER);
            getContentPane().add(someJP[2], BorderLayout.SOUTH);
                setSize(200,200);
                setDefaultCloseOperation(EXIT_ON_CLOSE);
        }
        public static void main(String [] arg) {
                DemoJPanelSF myDPF = new DemoJPanelSF();
                myDPF.setVisible(true);
        }
}
```

JTextField

- Use a `JTextField` object to accept a single line to text from a user

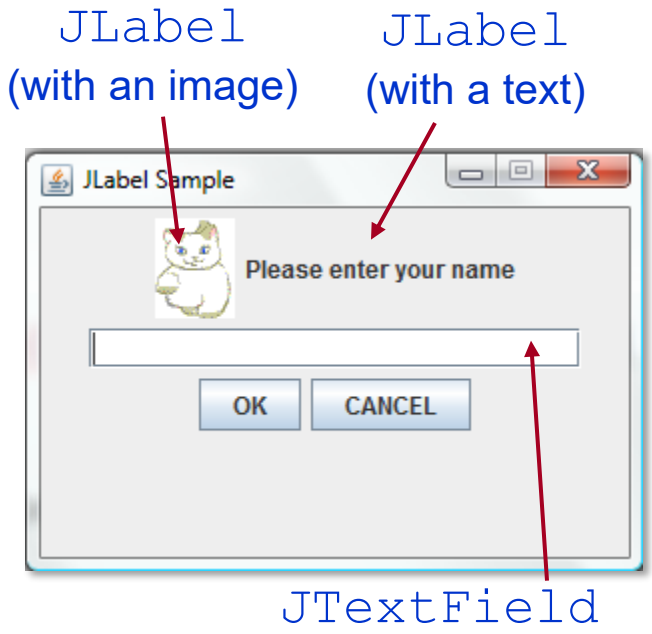
```
// Create a TextField with column width 22
JTextField inputTF = new JTextField(22);
getContentPane().add(inputTF);
```

- The `getText()` method of `JTextField` is used to retrieve the text that the user entered
- An action event is generated when the user presses the ENTER key. We may handle this ENTER key action similar to handling Button click.

```
inputTF.addActionListener(e -> { // ENTER key jumps here
    ////// code to let User-Press-ENTER respond HERE  //////
    JOptionPane.showMessageDialog(null, "String Typed in TF:\n"
            + ((JTextField)e.getSource()).getText() );
});
```

JLabel

- Use a `JLabel` to display a label
- A label can be a text line or an image
- When creating an image label, we pass `ImageIcon` object



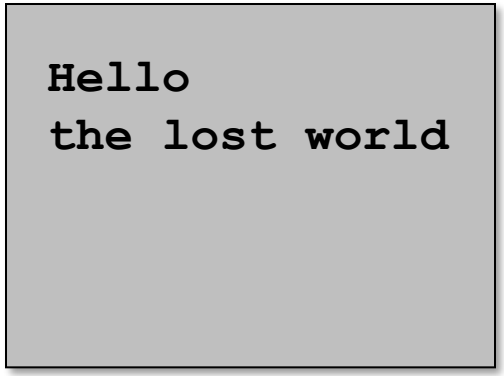

```
JLabel textLabel = new JLabel("Please enter your name");
getContentPane().add(textLabel);

JLabel imgLabel = new JLabel(new ImageIcon("cat.gif"));
getContentPane().add(imgLabel);
```

JTextArea

- We use a `JTextArea` object to display or allow the user to enter multiple lines of text
- The `setText()` method assigns the text to a `JTextArea`, replacing the current content
- The `append()` method appends the text to the current text

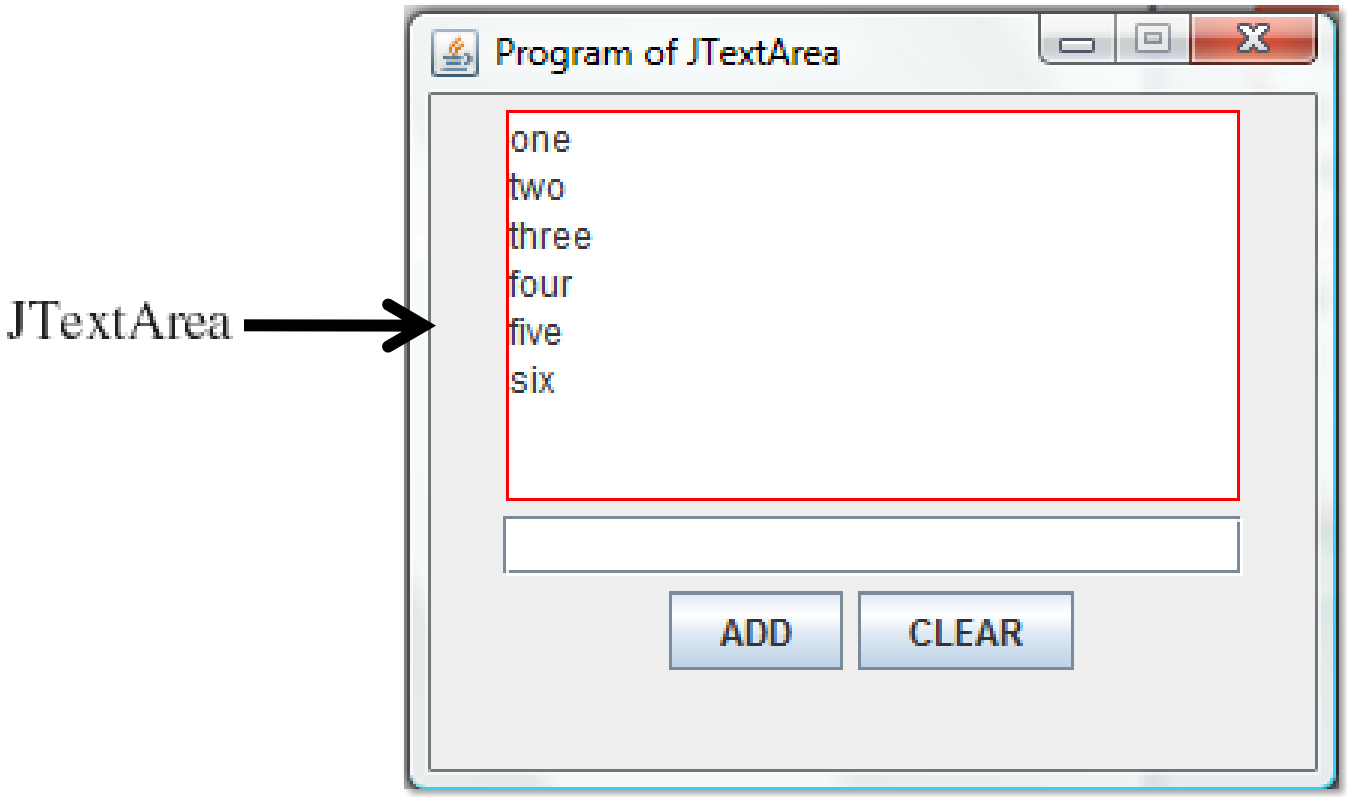
```
JTextArea textArea
        = new JTextArea( );
// . . .
textArea.setText("Hello\n");
textArea.append("the lost ");
textArea.append("world");
```



JTextArea

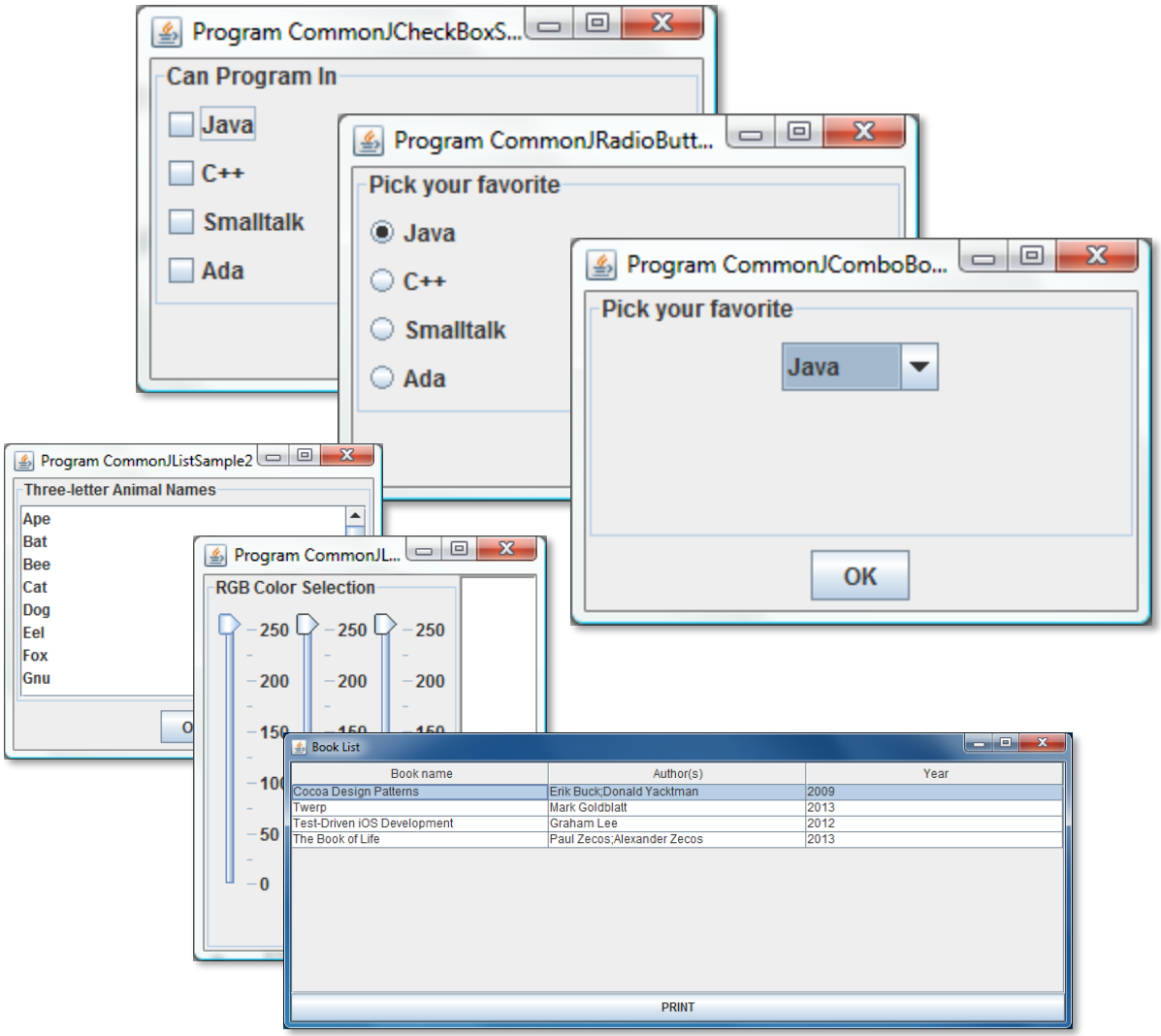
`JTextArea` Sample

- The state of a `JTextAreaSample` window after six words are entered



Other Common GUI Components

- `JCheckbox`
- `JRadioButton`
- `JComboBox`
- `JList`
- `JSlider`
- `JTable`



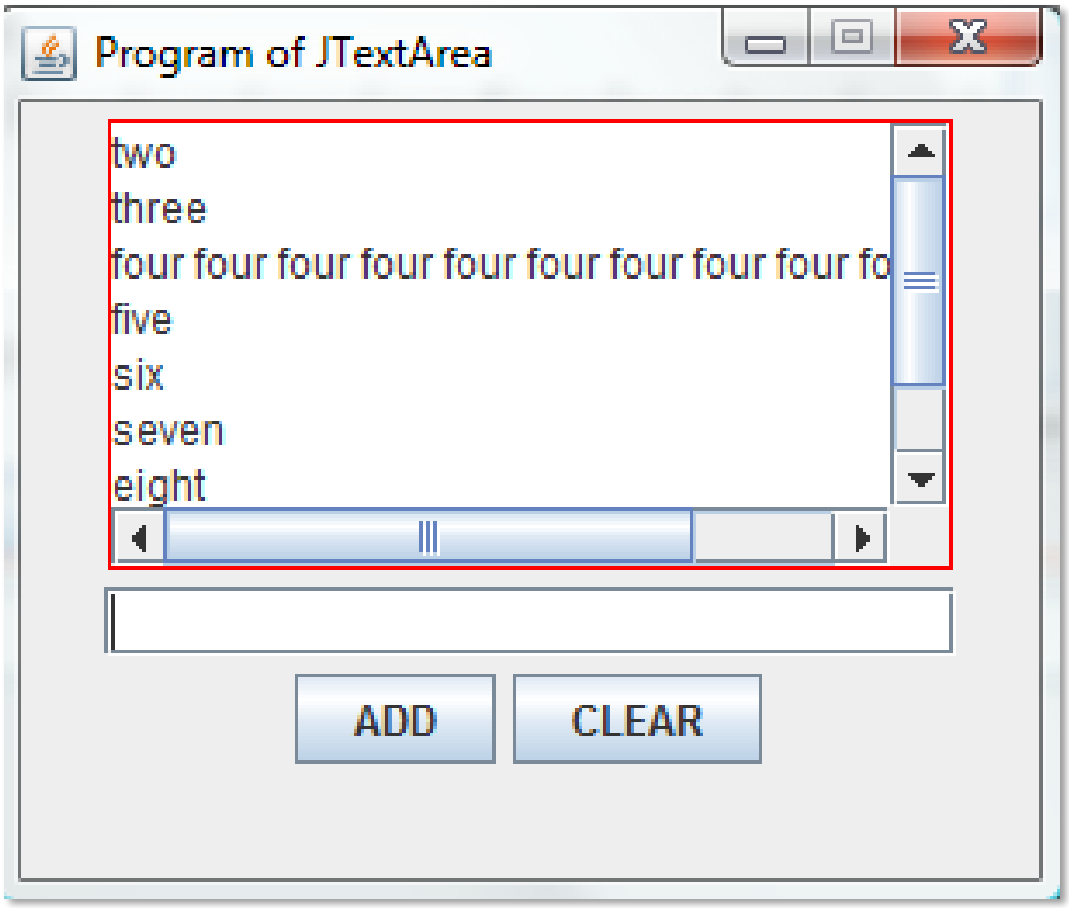
Adding Scroll Bars to `JTextArea`

- By default a `JTextArea` does not have any scroll bars. To add scroll bars, we place a `JTextArea` in a `JScrollPane` object

```
JTextArea   textArea   = new JTextArea();
// . . .
JScrollPane scrollText = new JScrollPane(textArea);
// . . .
getContentPane().add(scrollText);
```

`JTextAreaSample` with Scroll Bars

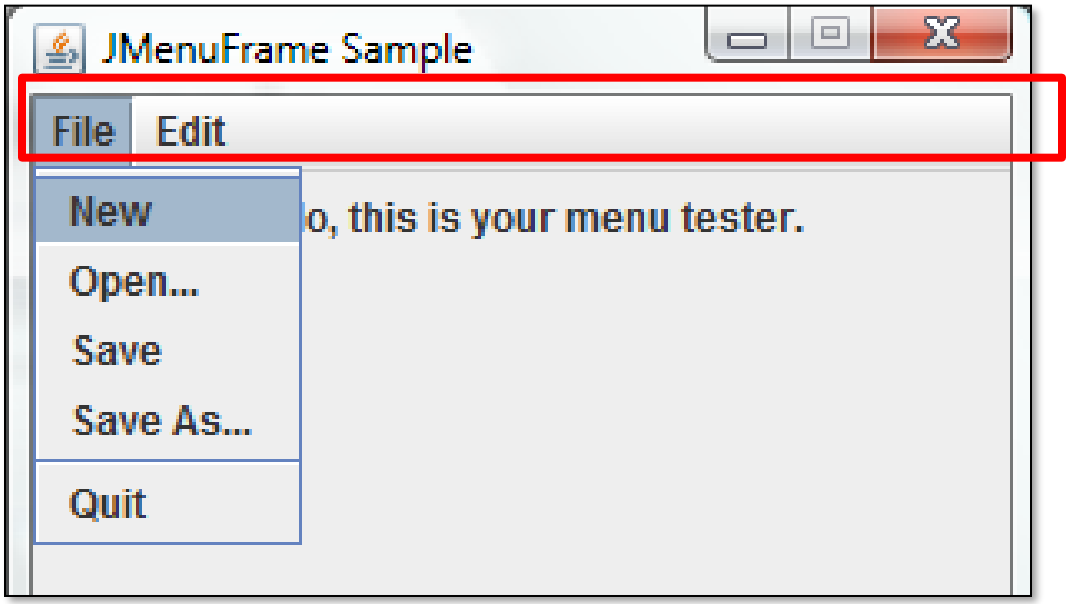
- A sample `JTextAreaSample` window when a `JScrollPane` is used



Menus

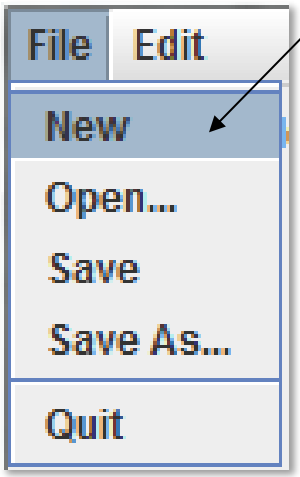
JMenuBar

`JMenuBar` is a bar where the menus are placed. There is one menu bar per frame.



JMenu

`JMenu` (such as File or Edit) is a group of menu choices.



JMenuItem

`JMenuItem` (such as New, Open, or Save) is an individual menu choice in a `JMenu` object.

separator

** Only the `JMenuItem` objects generate events*

A Typical Sequence for Creating Menus

1. Create a `JMenuBar` object and attach it to a frame
2. Create a `JMenu` object
3. Create `JMenuItem` objects and add them to the `JMenu` object
4. Attach the `JMenu` object to the `JMenuBar` object

References

- This set of slides is only for educational purpose.

- Part of this slide set is referenced, extracted, and/or modified from the followings:
  - Deitel, P. and Deitel H. (2017) “Java How To Program, Early Objects”, 11ed, Pearson.
  - Liang, Y.D. (2017) “Introduction to Java Programming and Data Structures”, Comprehensive Version, 11ed, Prentice Hall.
  - Wu, C.T. (2010) “An Introduction to Object-Oriented Programming with Java”, 5ed, McGraw Hill.
  - Oracle Corporation, “Java Language and Virtual Machine Specifications” https://docs.oracle.com/javase/specs/
  - Oracle Corporation, “The Java Tutorials” https://docs.oracle.com/javase/tutorial/
  - Wikipedia, Website: https://en.wikipedia.org/