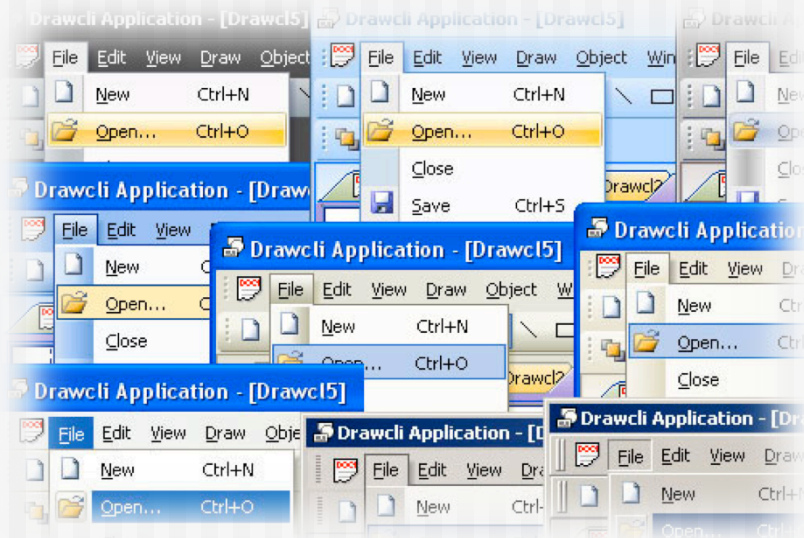


# Building GUIs

---

## Constructing Graphical User Interfaces in Java



# Overview

---

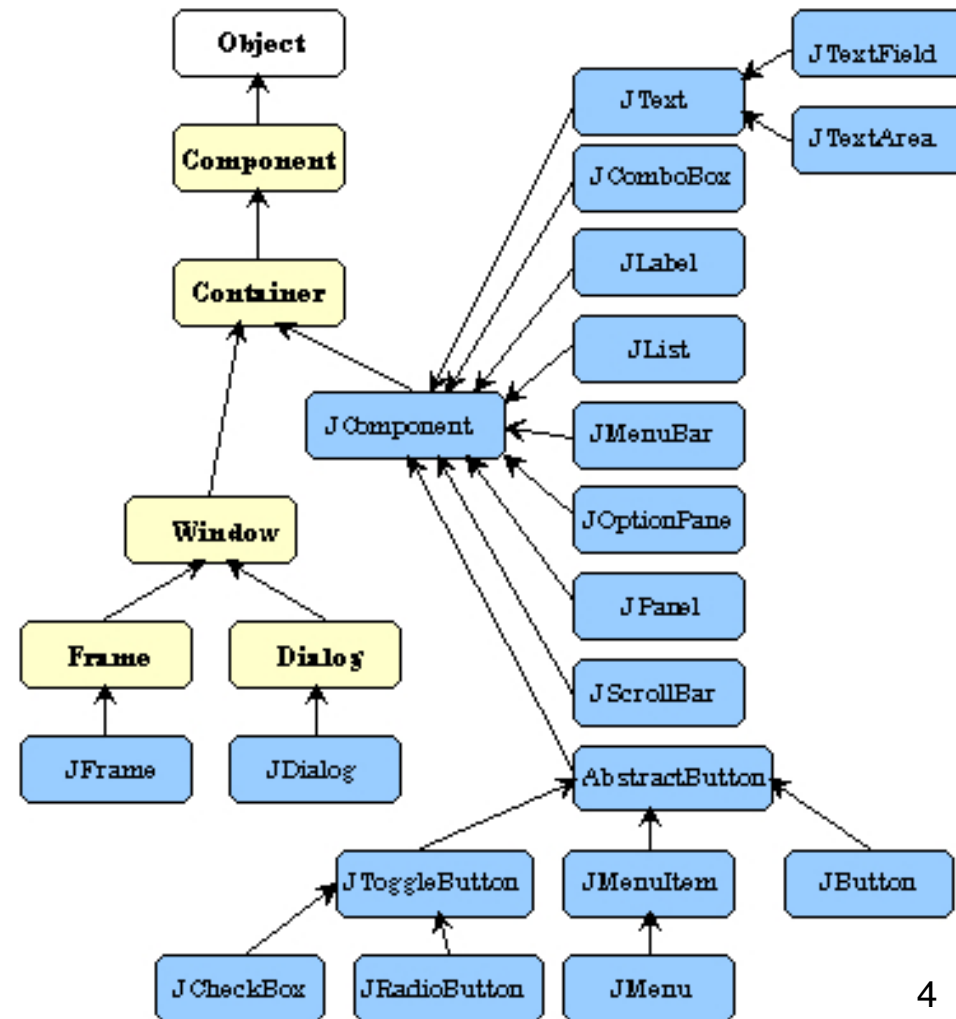
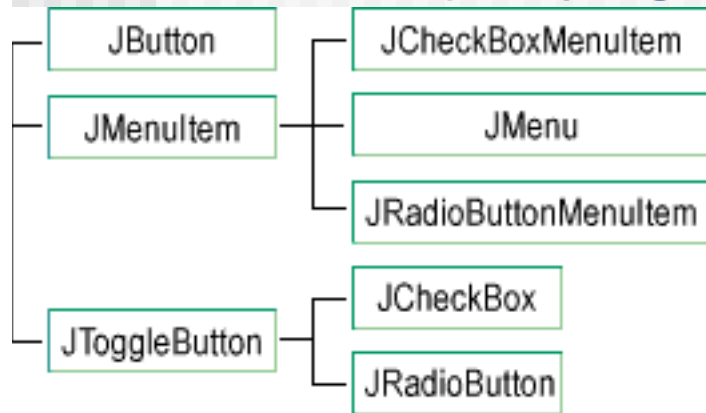
- Constructing GUIs
- Interface components
- GUI layout
- Event handling

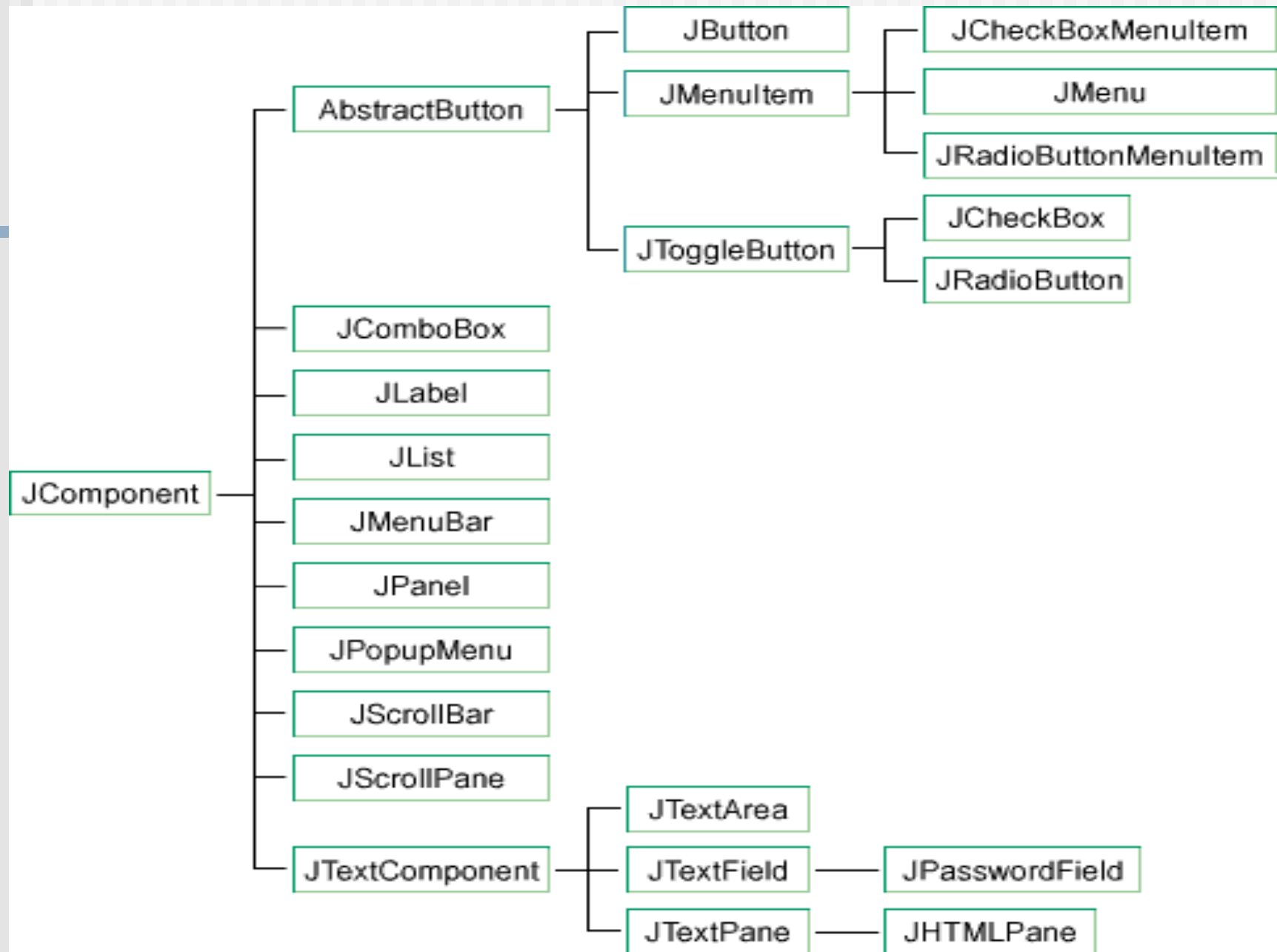
# GUI Principles

---

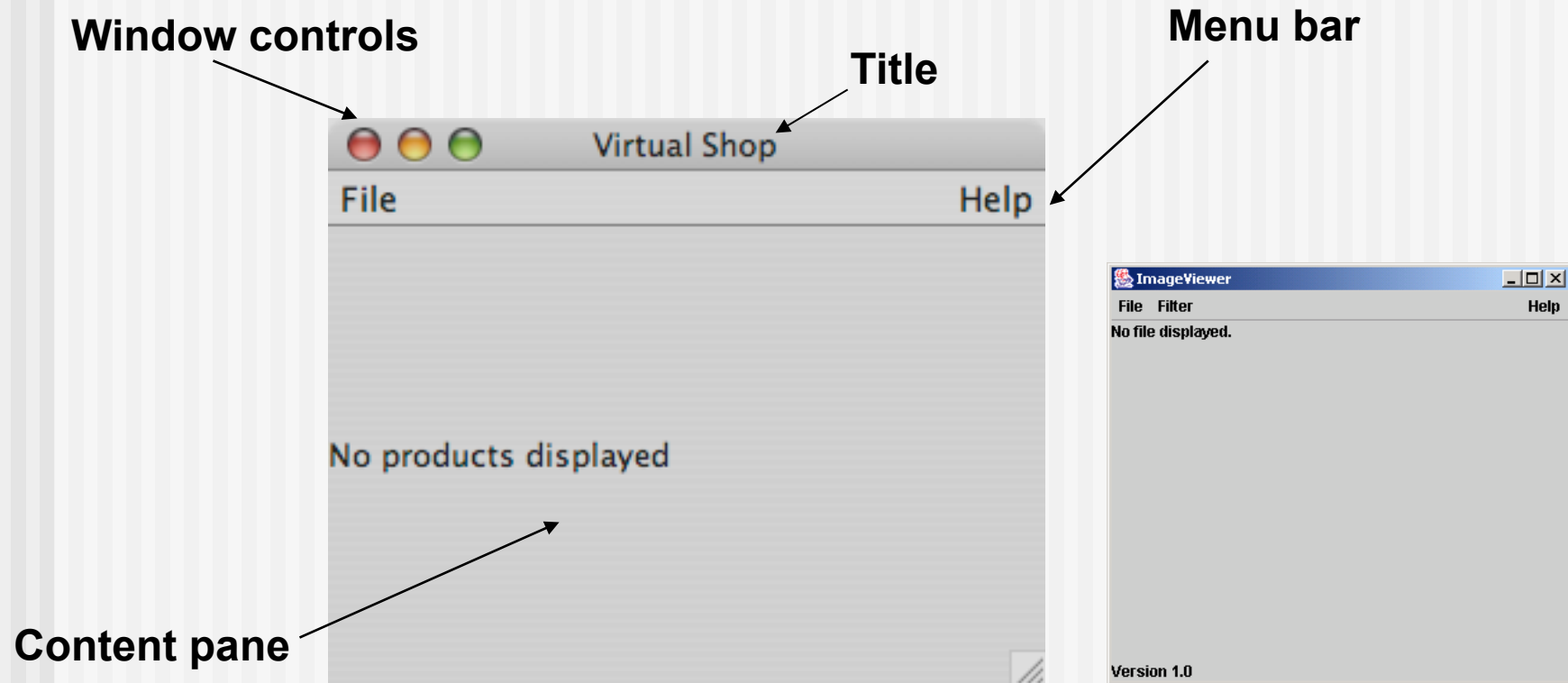
- **Components:** GUI building blocks
  - Buttons, menus, sliders, etc.
- **Layout:** arranging components to form a usable GUI
  - Using layout *managers*
- **Events:** reacting to user input
  - Button presses, menu selections, etc.

# AWT and Swing





# Elements of a frame



# Creating a frame

---

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

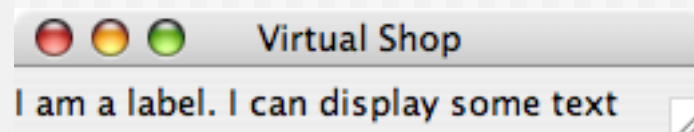
// comment omitted

public class StoreGUI extends JFrame
{
    public StoreGUI()
    {
        super("Virtual Shop");
    }
    // rest of class omitted
}
```

# The content pane

```
/**
 * Create the GUI for the virtual shop
 */
public void interact()
{
    frame = new JFrame("Virtual Shop");
    Container contentPane = this.getContentPane();
    JLabel label = new JLabel("I am a label. I can display
                               some text");

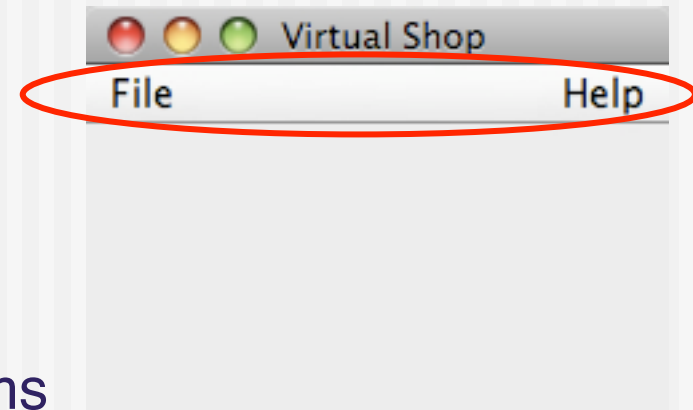
    contentPane.add(label);
    frame.pack();
    frame.setVisible(true);
}
```



# Adding menus

---

- `JMenuBar`
  - Displayed below the title
  - Contains the menus
- `JMenu`
  - e.g. *File*. Contains the menu items
- `JMenuItem`
  - e.g. *Open*. Individual items



```
private void makeMenuBar()  
{  
    JMenuBar menubar = new JMenuBar();  
    setJMenuBar(menubar);  
    // create the File menu  
    JMenu fileMenu = new JMenu("File");  
    menubar.add(fileMenu);  
  
    ...  
    JMenuItem quitItem = new JMenuItem("Quit");  
    fileMenu.add(quitItem);  
  
    ...  
    JMenuItem aboutItem = new JMenuItem("About Virtual Shop ...");  
    fileMenu.add(aboutItem);  
  
    ...  
}
```

# Event handling

---

- Events correspond to user interactions with components
- Components are associated with different event types
  - Frames are associated with **WindowEvent**
  - Menus are associated with **ActionEvent**
- Objects can be notified when an event occurs
  - Such objects are called *Listeners*
    - **Window Listeners**
    - **Action Listeners**
    - **Mouse Listeners**

# Centralized event receipt

---

- A single object handles all events
  - Implements the `ActionListener` interface
  - Defines an `actionPerformed` method
- It registers as a listener with each component
  - `item.addActionListener(this)`
- It has to work out which component has dispatched the event

```

public class StoreGUI extends JFrame implements ActionListener
{
    ...
    public void actionPerformed(ActionEvent e)
    {
        String command = e.getActionCommand();
        if(command.equals("Quit")) {
            ...
        }
        else if (command.equals("About Virtual Shop ...")) {
            ...
        }
        ...
    }
    ...
    private void makeMenuBar(JFrame frame)
    {
        ...
        quitItem.addActionListener(this);
        aboutItem.addActionListener(this);
        ...
    }
}

```

# Centralized event handling

---

- The approach works
- It is used, so you should be aware of it
- However ...
  - It does not scale well
  - Identifying components by their text is fragile
- An alternative approach is preferred

# Nested class syntax

- Class definitions may be nested

```
public class Enclosing
{
    ...
    private class Inner
    {
        ...
    }
}
```

Inner class



# Anonymous inner classes

---

- Obey the rules of inner classes
- Used to create one-off objects for which a class name is not required
- Use a special syntax
- The instance is always referenced via its supertype, as it has no subtype name

# Anonymous action listener

---

```
JMenuItem quitItem = new JMenuItem("Quit");

quitItem.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        quit();
    }
});
```

# Anonymous class elements

```
quitItem.addActionListener (
```

**Anonymous object creation  
(referenced via its  
supertype)**

```
new ActionListener ()
```

```
{
```

```
public void actionPerformed(ActionEvent e)
```

```
{
```

```
quit();
```

```
}
```

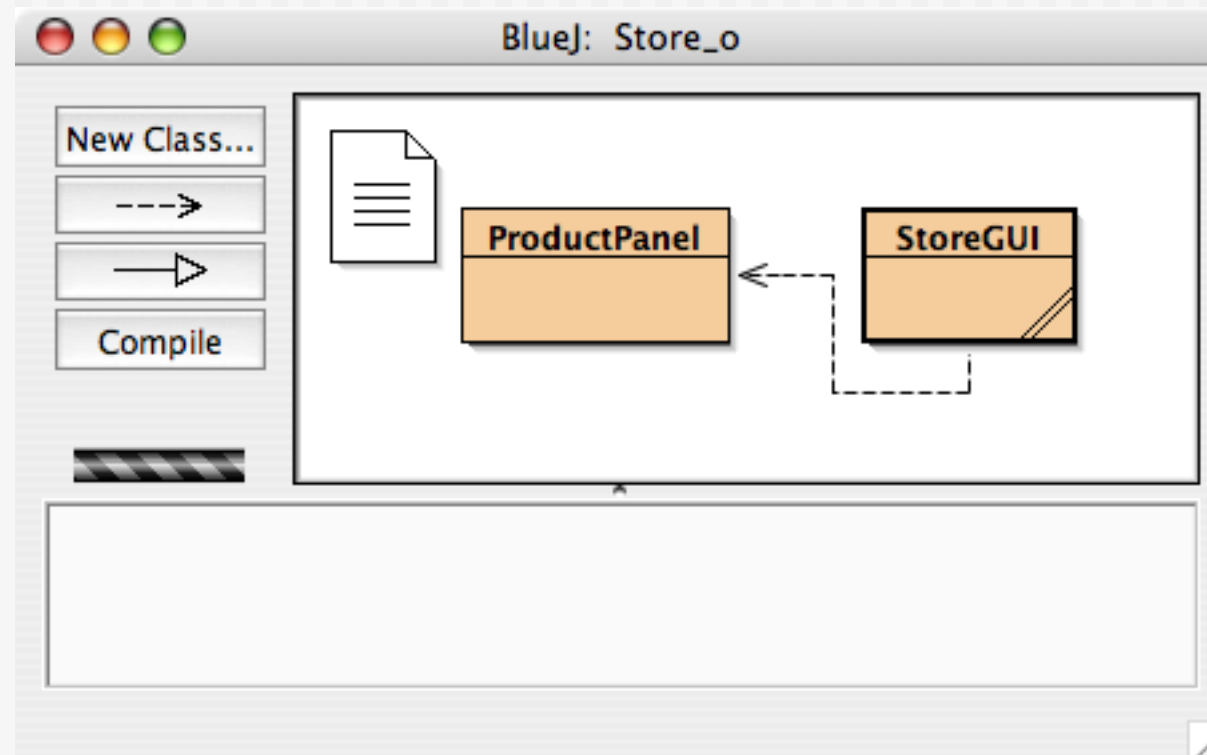
```
}
```

```
);
```

**Actual parameter**

**Class definition**

# The store GUI project



# The center pane

The screenshot shows a web application window titled "Virtual Corner Shop". The interface includes a menu bar with "File" and "Help", a search bar, and navigation buttons for "All products", "Food and Drink", "House and Pet", and "Toiletry and Baby". A left sidebar contains buttons for "Basket", "Delivery", "My account", and "Checkout". The main area displays a list of products with their prices and quantity controls. A summary table on the right lists the items in the basket with their quantities, unit prices, and total prices.

Product	Quantity	Unit Price	Total Price
baked beans	2	1.20	2.40
chedar cheese	3	1.20	3.60
edam cheese	2	1.80	3.60
goats cheese	1	1.80	1.80
gouda cheese	1	1.80	1.80
grated cheese	2	1.50	3.00
houmous dip	2	0.64	1.28
new potatoes	0	0.88	
vegetable oil	2	3.20	6.40
peanut butter	1	1.37	1.37
<b>Total for groceries</b>			<b>25.25</b>

# Adding the center pane

---

```
public class StoreGUI extends JFrame
{
    private JPanel centerPane;
    ...

    private void interact()
    {
        Container contentPane = getContentPane();
        centerPane = new JPanel();
        contentPane.add(centerPane);
    }
    ...
}
```

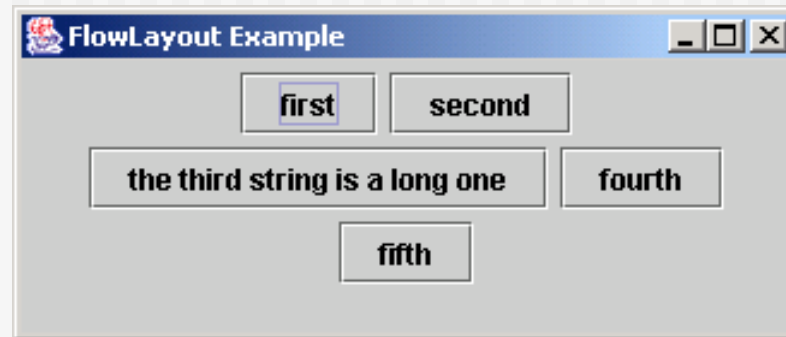
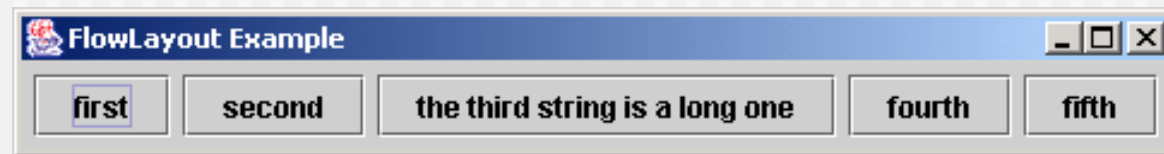
# Layout managers

---

- Manage limited space for competing components
  - `FlowLayout`, `BorderLayout`, `GridLayout`, `BoxLayout`, `GridBagLayout`.
- Manage `Container` objects, e.g. a content pane
- Each imposes its own style

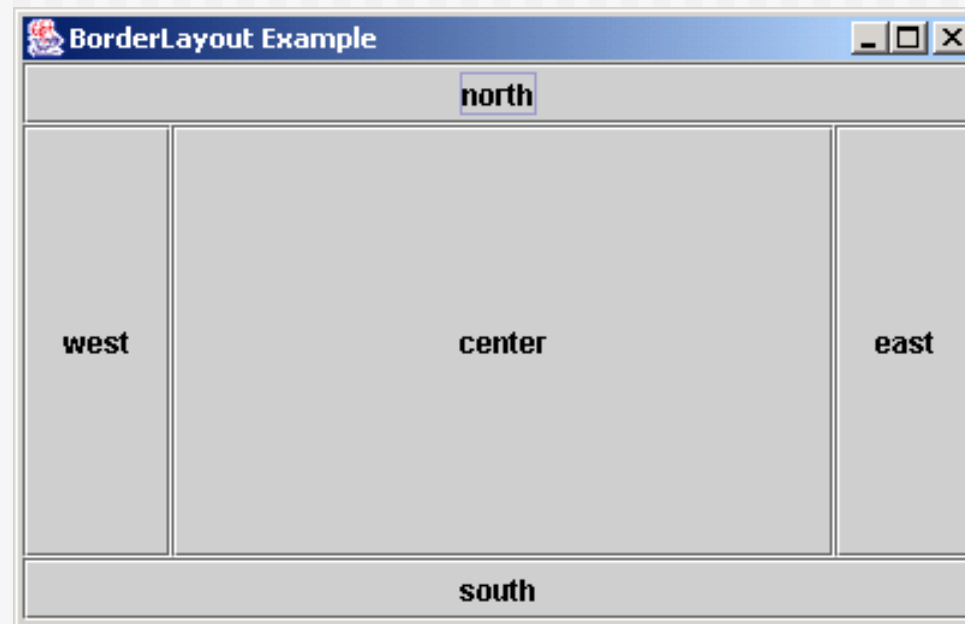
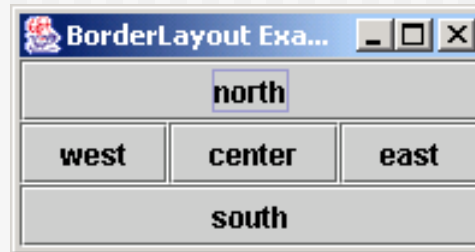
# FlowLayout

---

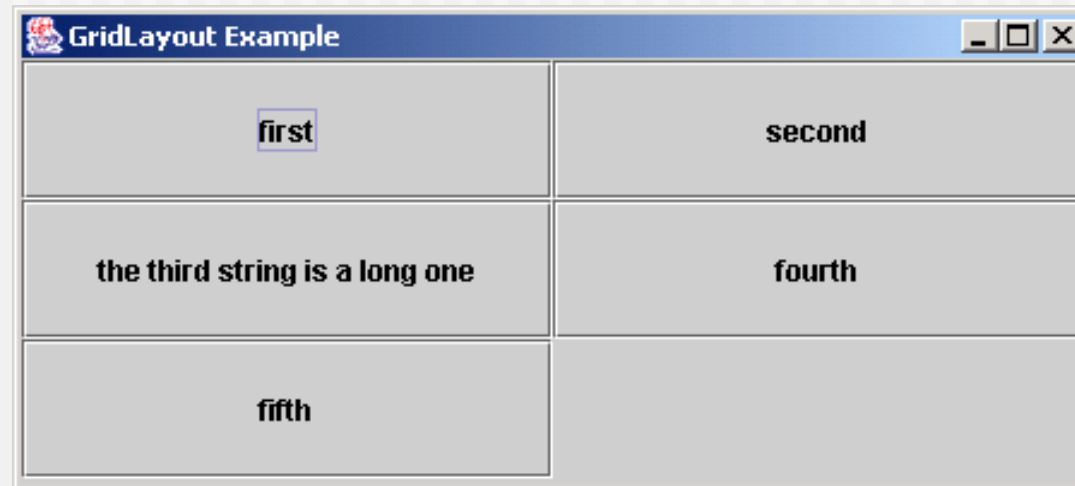
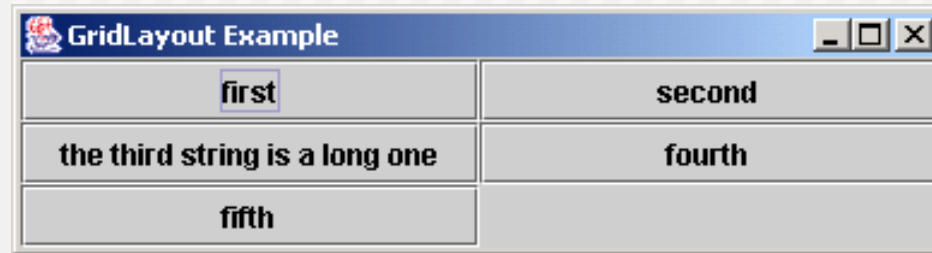


# BorderLayout

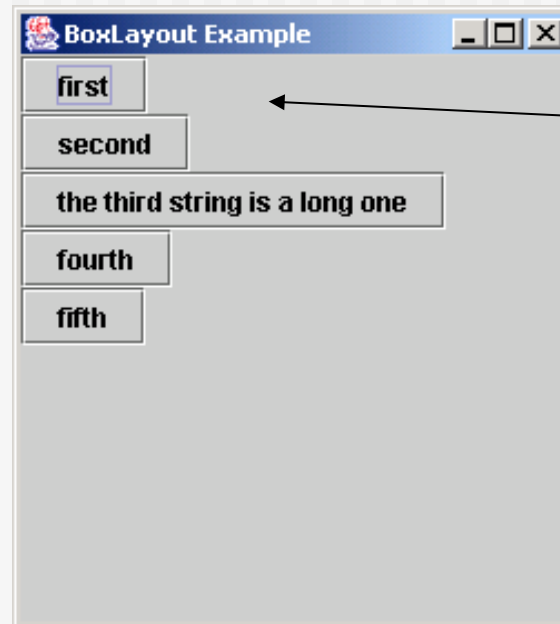
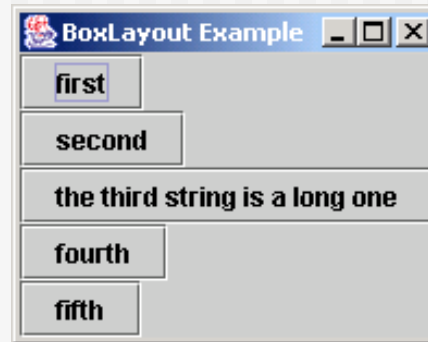
---



# GridLayout



# BoxLayout



Note: no component resizing

# Nested containers

---

- Sophisticated layouts can be obtained by nesting containers
  - Use `JPanel` as a basic container
- Each container will have its own layout manager
- Often preferable to using a `GridBagLayout`

# Buttons and nested layouts

The screenshot shows a shopping cart interface titled "Virtual Corner Shop". The interface is divided into several sections: "Basket", "Delivery", "My account", and "Checkout". Each section contains a list of products with their prices, quantities, and "Add" buttons. A table on the right side of the interface displays the items in the cart, including their quantities, unit prices, and total prices. The interface is annotated with four layout types: "Border layout" (red arrow), "Box layout" (green arrow), "Border layout" (brown arrow), and "Flow layout" (blue arrow).

Product	Quantity	Unit Price	Total Price
baked beans	2	1.20	2.40
chedar cheese	3	1.20	3.60
edam cheese	2	1.80	3.60
goats cheese	1	1.80	1.80
gouda cheese	1	1.80	1.80
grated cheese	2	1.50	3.00
houmous dip	2	0.64	1.28
peanut butter	1	1.37	1.37
vegetable oil	2	3.20	6.40
Total for groceries			25.25

Border layout

Box layout

Border layout

Flow layout

# Buttons and nested layouts

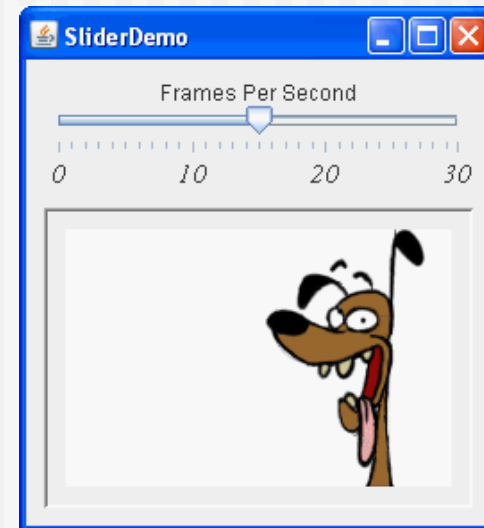
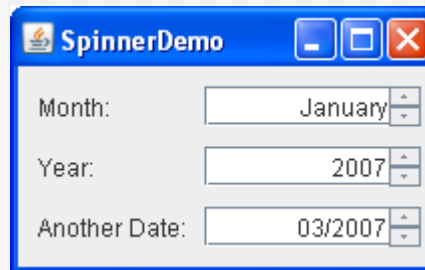
```
public ProductPanel(String n, int p)
{
    setLayout(new BorderLayout());
    ...
    JLabel productName = new JLabel(name);
    add(productName, BorderLayout.CENTER);

    // Create the right side
    JPanel rightPane = new JPanel();
    ...
    JButton addButton = new JButton("Add");
    addButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) { add(); }
    });
    rightPane.add(addButton);
    add(rightPane, BorderLayout.EAST);
}
```



# Other components

- Dialogs
- Borders
- Slider
- Spinner
- Tabbed pane
- Scroll pane



# Key Points

---

- Aim for cohesive application structures
  - Endeavour to keep GUI elements separate from application functionality
- Pre-defined components simplify creation of sophisticated GUIs
- Layout managers handle component juxtaposition
  - Nest containers for further control

## Key Points *(cont.)*

---

- Many components recognize user interactions with them
- Reactive components deliver events to listeners
- Anonymous inner classes are commonly used to implement listeners