软件的展示层—
图形界面程序设计

主讲老师：申雪萍

目前，软件展示层的技术有哪些？你了解多少？

- Java 前端
  - Awt, Swing
- C# 前端
  - **WinForm**
- 嵌入式前端
  - Qt
- Web前端
  - HTML, CSS, Javascript
  - ASP，JSP，PHP等
  - AJAX
  - XML等

目前流行的前端框架

- 除了Javascript，css，html外，还要掌握vue，react，jQuery，angular，Bootstrap，ElementUI等等

讨论：一个图形界面需要具备的元素有哪些？如何设计？

- 窗口；
- 菜单栏，菜单项，弹出式菜单，一级菜单，二级菜单，快捷键等；
- 对话框；
- 文本区，文本域，单选按钮，多选按钮，列表，标签，滚动条，进度条等；
- 字体，颜色，颜色选择器,文件选择器；
- 画布，面板
- 布局管理；
- 事件管理等

主要内容

- Java AWT
- Java Swing
- Java 事件处理
- Java 布局管理器
- Java 对话框（自学）
- Java 菜单（自学）
- Java 二维图形绘制（自学）
- Java 字体和颜色设置（自学）

重点掌握：

1. 图形界面的类和接口的设计思路
2. 容器和组件的关系
3. 以及事件处理的委托机制

编写JAVA图形界面程序，你需要了解的常规问题

- 什么是容器？
  - （窗口，对话框等）
- 什么是组件？
  - （按钮，文本框，文本域，单选按钮，复选按钮，菜单，菜单项等）
- 如何创建与使用容器？
- 如何创建与使用组件？
- 如何使用布局管理器对组件进行管理？
- 如何使用Java的事件处理机制控制组件？

主要内容

- **Java AWT**
- Java Swing
- Java 事件处理
- Java 布局管理器
- Java 对话框
- Java 菜单
- Java 二维图形绘制（自学）
- Java 字体和颜色设置（自学）

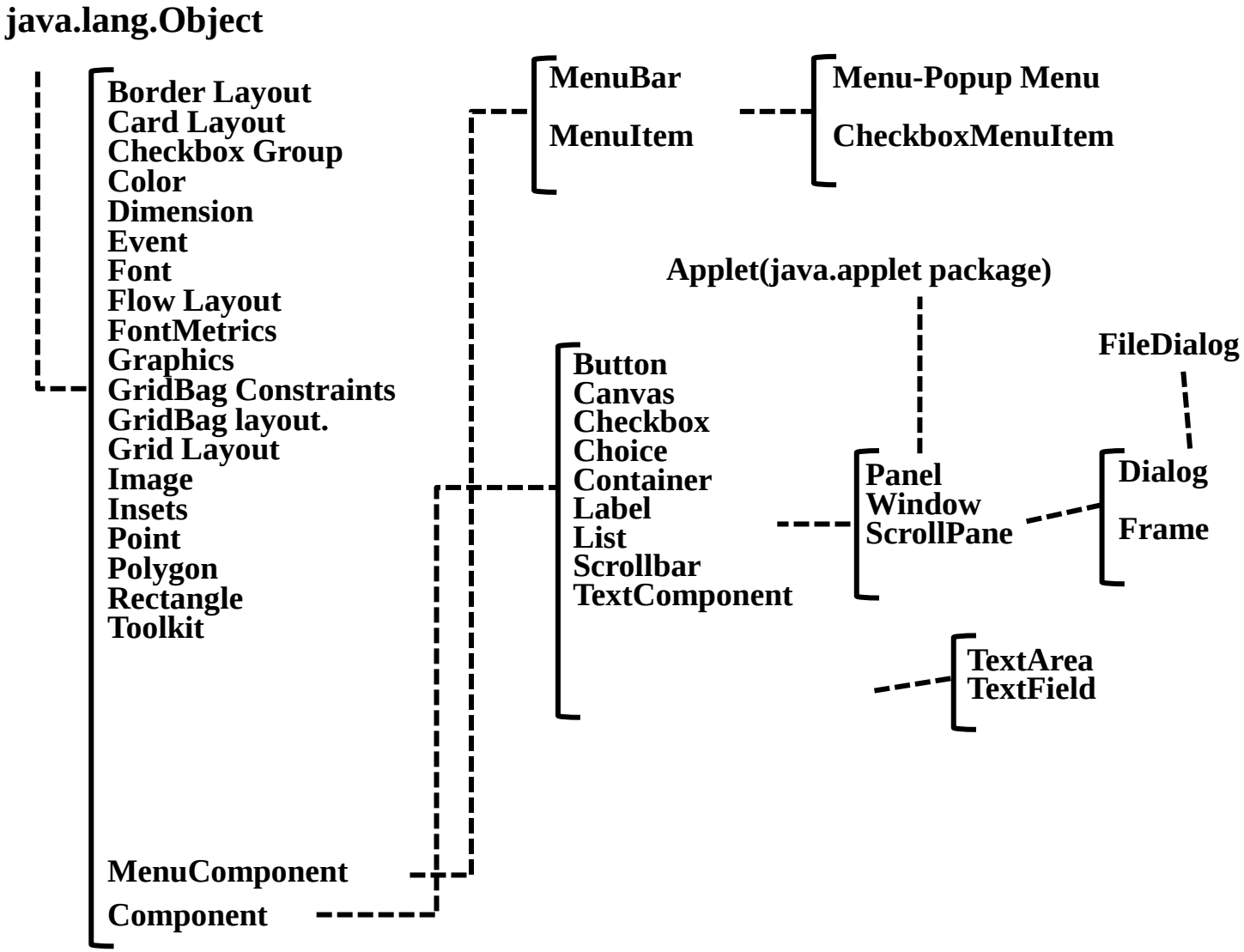
创建Java图形界面的程序类库

- 创建Java图形界面的程序类库主要有两个:
  - AWT—Abstract Window Toolkit（java.awt.*）;
  - Swing（javax.swing.*）
- 无论AWT还是Swing，都是按照面向对象的思想来创建GUI，**它提供了容器类、众多的组件类和布局管理器类。**

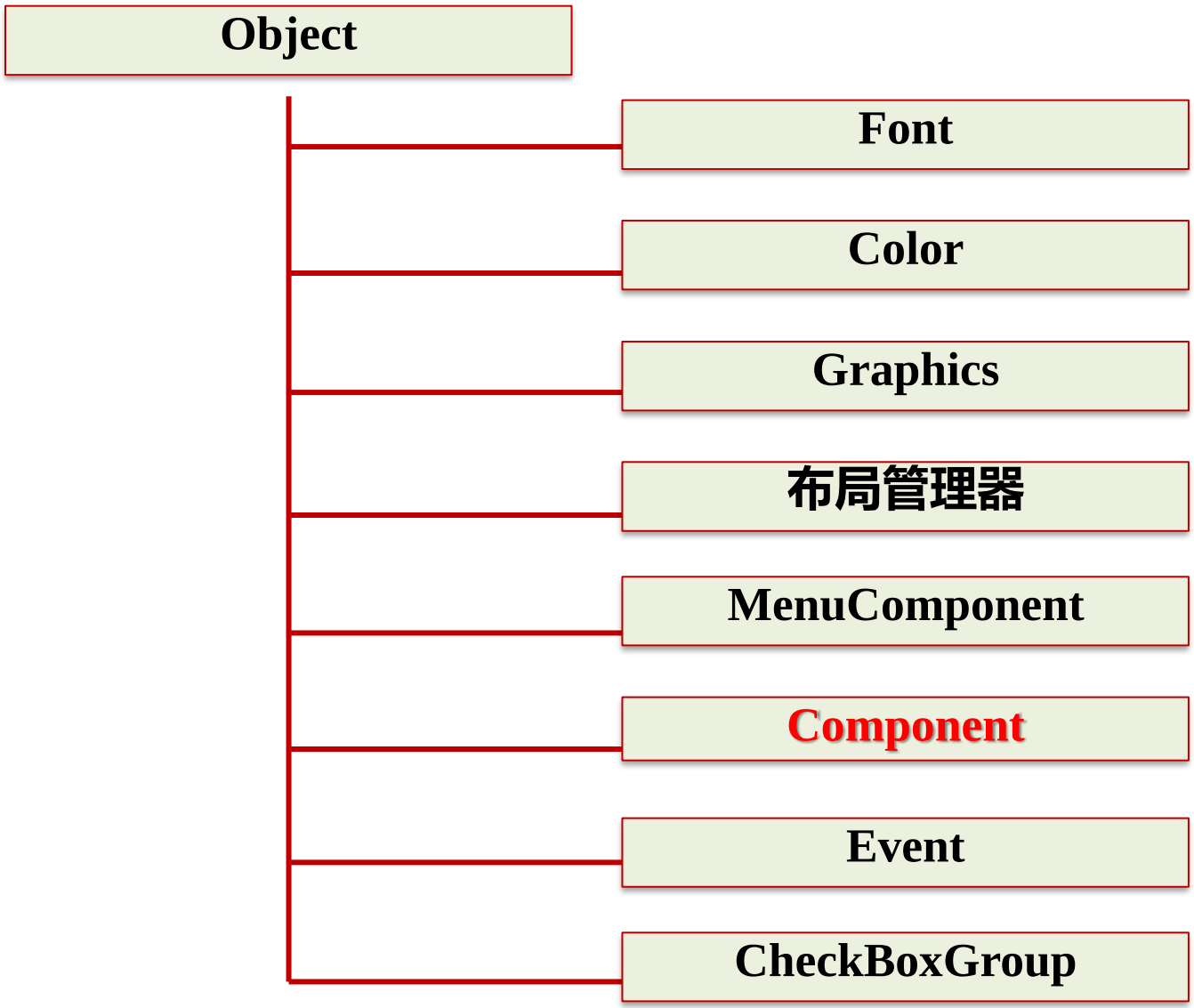
java.awt 包

- 抽象窗口工具集AWT（Abstract Window Toolkit），是使用Java进行GUI设计的基础
  - 目前Java图形界面中主要使用的是AWT中的布局管理器和事件处理

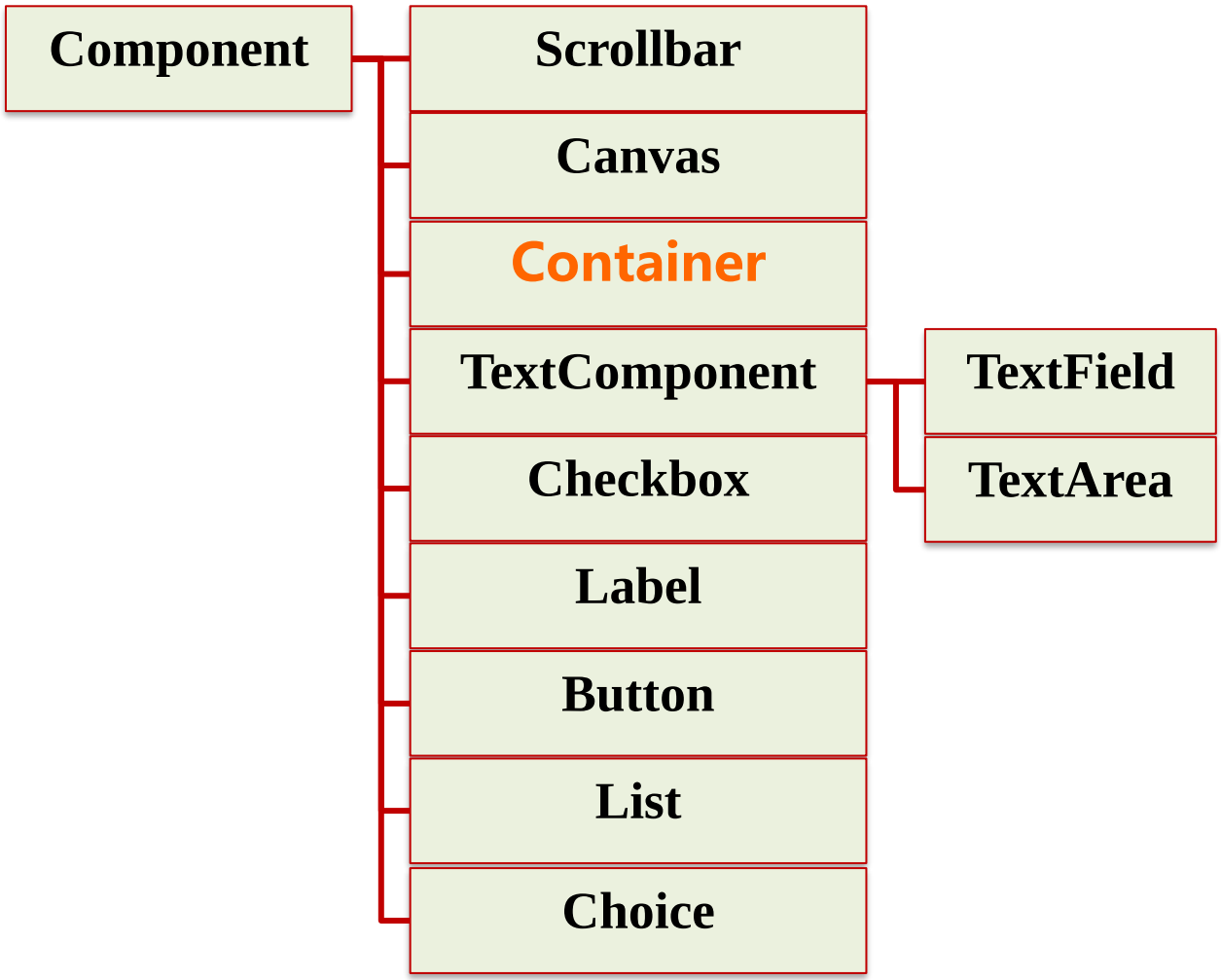
package java.awt



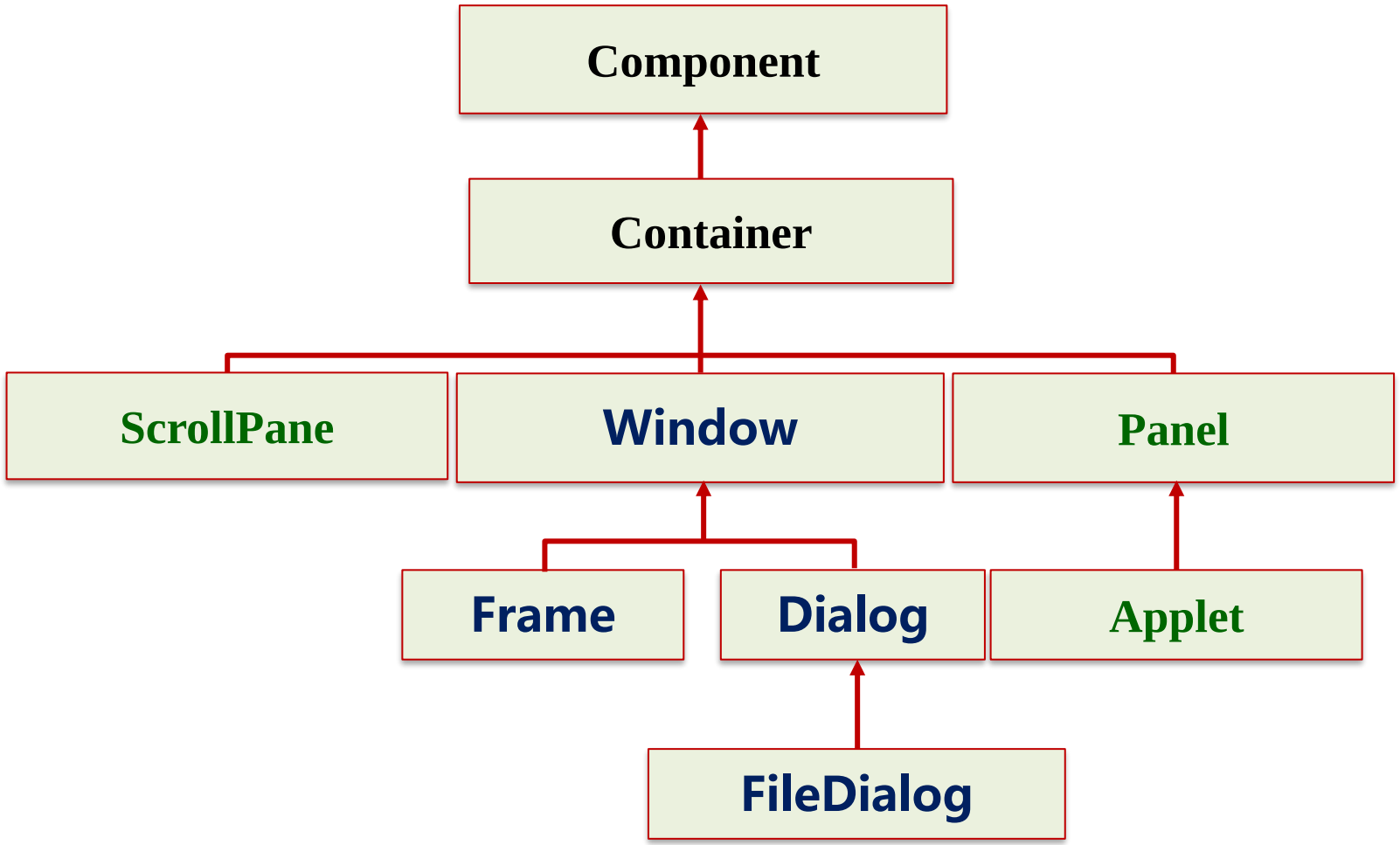
package java.awt



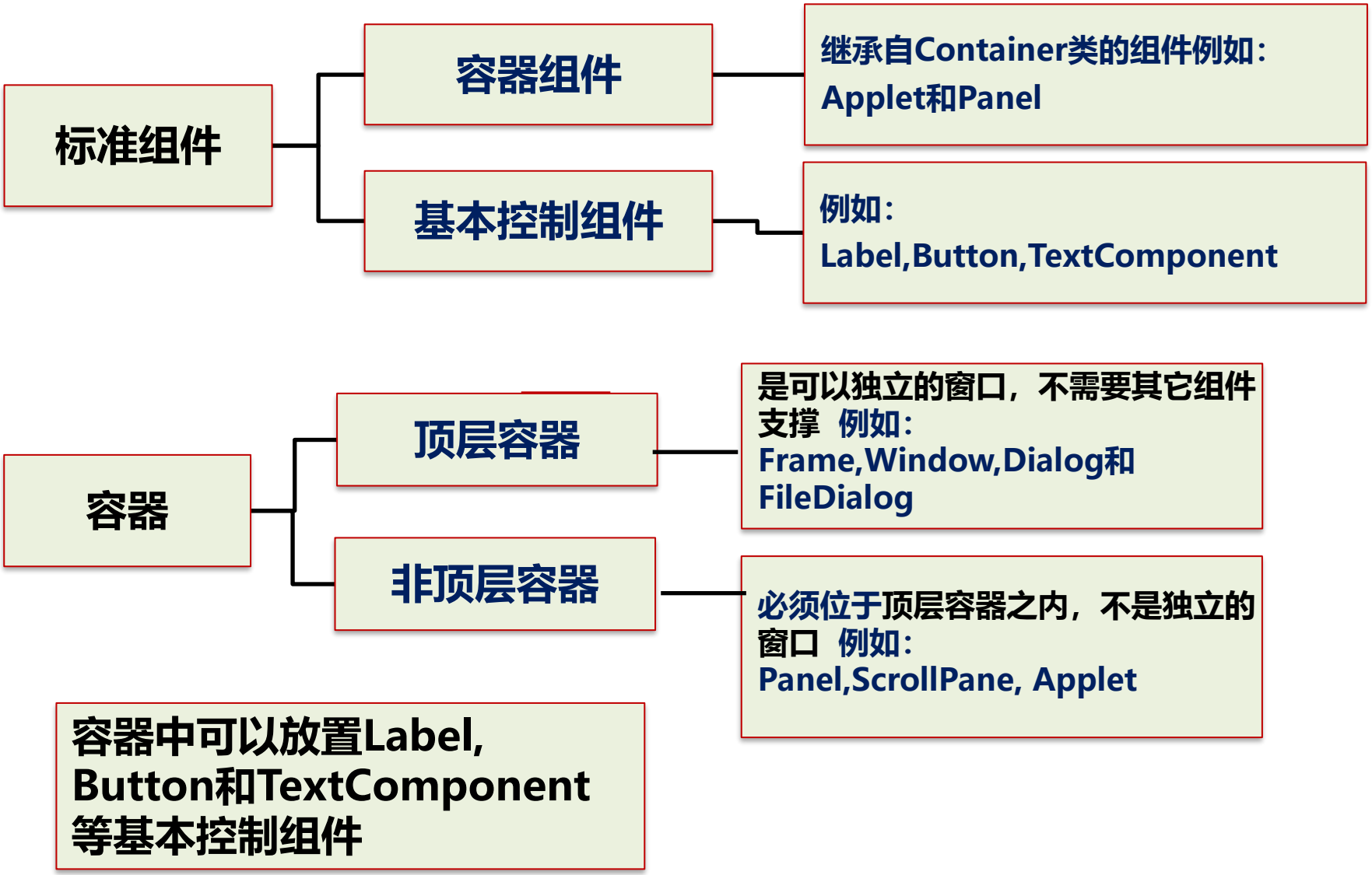
package java.awt



package java.awt



package java.awt



容器类（Container）组件

Container类的常用方法

- add(Component comp);
- remove(Component comp);
- setLayout(LayoutManager mgr);
- setLocation(int x, int y);
- setSize(int x, int y);
- setBackground(Color color);
- show();
- setVisible(boolean b);
- validate();

//Ensures that this component has a valid layout.

控制组件

- **控制组件**主要用来提供人机交互的基本控制界面
- **主要包括：**
  - 按钮（Button）
  - 单选按钮(RadioBox)
  - 复选框(Checkbox)
  - 下拉列表框(Choice)
  - 画板(Canvas)
  - 列表框(List)
  - 标签(Label)
  - 文本组件(TextField和TextArea)
  - 滚动条(ScrollBar)
  - 菜单(Menu)等

Component类常用方法（1）

- public void add(PopupMenu popup)；
- public void setBackground(Color c)；
- public Color getBackground()；
- public void setForeground(Color c)；
- public Color getForeground()；
- public void setFont(Font f)；
- public Font getFont()；

Component类常用方法（2）

- public void setSize(int width,int height)；
- public void setVisible(boolean b)；
- public void setEnabled(boolean b)；
- public void repaint(int x，int y，int width，int height)；
- public void requestFocus()；
- public Graphics getGraphics()；

设置GUI应用程序的流程

- 引用需要的包和类
- 设置一个顶层的容器
- 根据需要为容器设置布局管理器或使用默认布局管理器
- 将组件添加到容器内，位置自行设计
- 为响应事件的组件编写事件处理代码

Java awt 中创建窗口

```
import java.awt.*;
import java.awt.event.*;
public class TestFrame {
    public static void main(String[] s) {
        Frame f = new Frame("First Frame");
        f.setLocation(100, 100);
        f.setSize(200, 200);
        f.setBackground(Color.red);
        f.setVisible(true);
        f.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }
}
```

UseTextArea1.java

```
import java.awt.*;
import java.awt.event.*;
public class UseTextArea1{
   public static void main(String args[]){
          MyFrame f=new MyFrame();
          f.setSize(300,200);
          f.setBackground(Color.lightGray);
          f.setLayout(new FlowLayout());
          f.show();
   }
}
```

```
class MyFrame extends Frame implements ActionListener{
        TextArea ta1;          TextField tf1;
        Button btn1,btn2,btn3;
        public MyFrame(){
          super("my frame");
          ta1=new TextArea(4,30);
          tf1=new TextField(30);
          btn1=new Button("添加文本");
          btn2=new Button("插入文本");
          btn3=new Button("替换选中文本");
          btn1.addActionListener(this);
          btn3.addActionListener(this);
          btn2.addActionListener(this);
          System.out.println(this.toString());
          setLayout(new FlowLayout());
          add(ta1); add(tf1); add(btn1); add(btn2); add(btn3);
          this.addWindowListener(new WindowAdapter(){
      public void windowClosing(WindowEvent e){
                                  dispose();
                                  System.exit(0);
            }});
        }
```

```
    public void actionPerformed(ActionEvent e){
      if(e.getSource()==btn1){
              ta1.append(tf1.getText());
              tf1.setText("");
      }
     if(e.getSource()==btn2){
              ta1.insert(tf1.getText(),ta1.getCaretPosition());
              tf1.setText("");
      }
     if(e.getSource()==btn3){
        int k0=ta1.getSelectionStart();
        int k1=ta1.getSelectionEnd();
              ta1.replaceRange(tf1.getText(),k0,k1);
      }
    }
}
```

主要内容

- Java AWT
- **Java Swing**
- Java 事件处理
- Java 布局管理器
- Java 对话框
- Java 菜单
- Java 二维图形绘制（自学）
- Java 字体和颜色设置（自学）

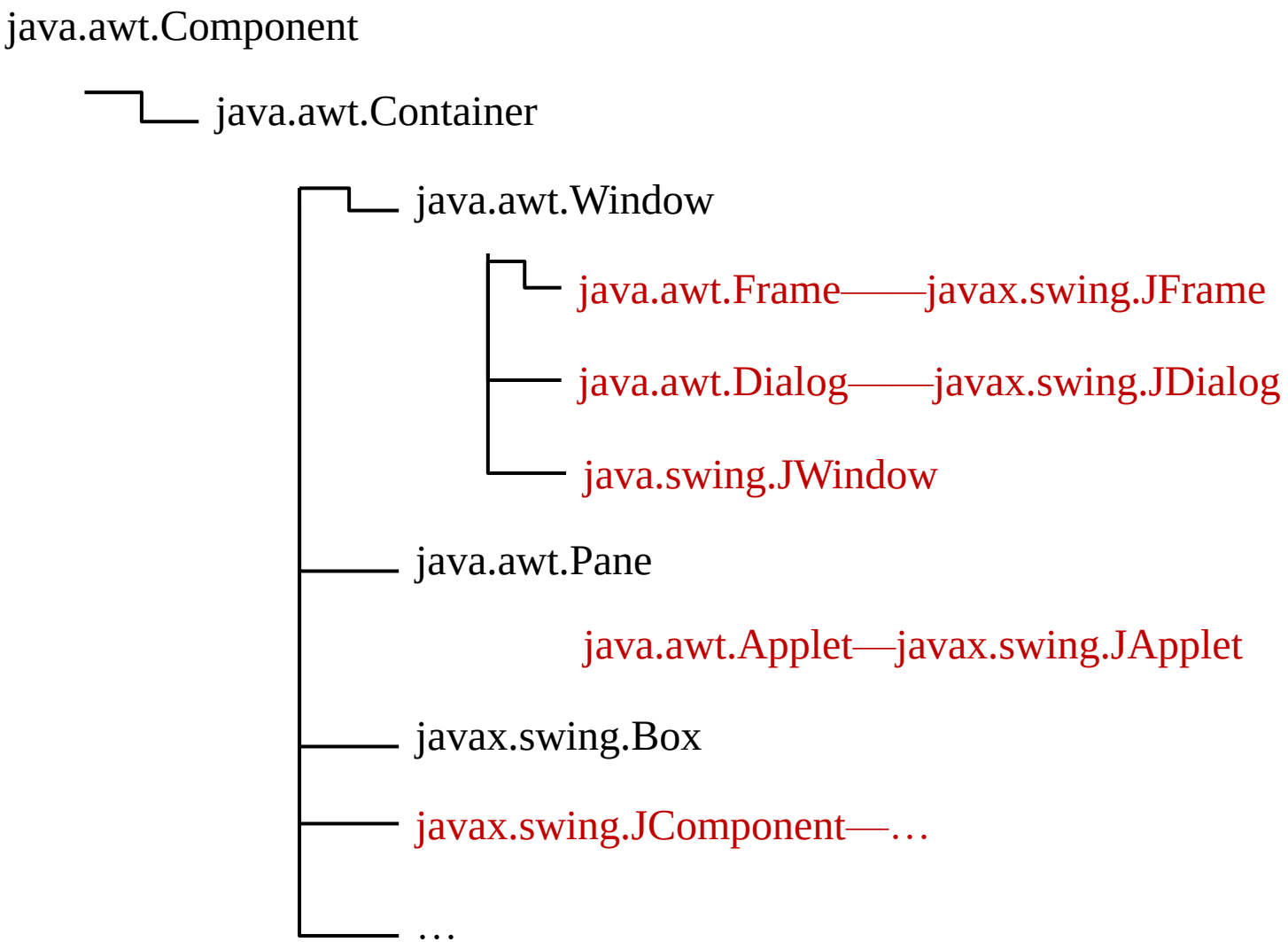
Java Swing

- 为了使用Java语言创建的图形界面也能够跨平台，即在不同操作系统中保持相同的外观，从JDK1.2版本开始引入了Swing组件，这些Swing组件位于javax.swing包中。
  - 几乎所有AWT组件对应有新功能更强的Swing组件；
  - 另外还加入了一些全新的组件；
  - Swing组件在名称前面多了一个字母“J”；
  - 除此之外，使用Swing开发的图形界面更美观，功能更强大。

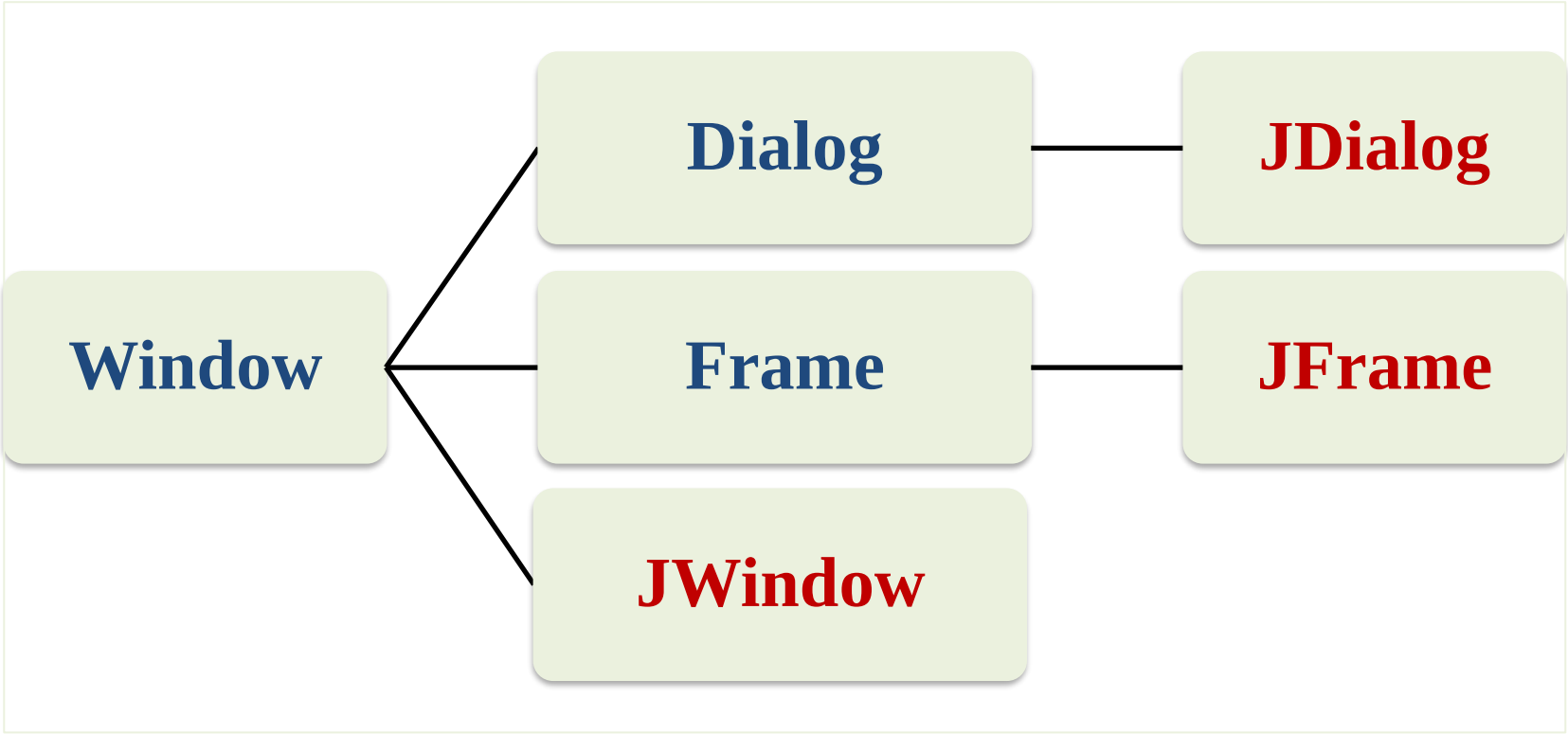
Swing组件的层次

- Swing是一个扩展的AWT，它提供了一个远大于AWT的综合的组件集合，并引入了新的概念和性能
- 在javax.swing包中，定义了两种类型的组件：
  - 顶层容器（JFrame、JApplet、JDialog和JWindow）
  - 和轻量级组件
- Swing组件从AWT的Container类继承而来。

Swing组件的层次


java.awt.Component
java.awt.Container
java.awt.Window
java.awt.Frame——javax.swing.JFrame
java.awt.Dialog——javax.swing.JDialog
java.swing.JWindow
java.awt.Pane
java.awt.Applet—javax.swing.JApplet
javax.swing.Box
javax.swing.JComponent—…
…

JFrame and Windows

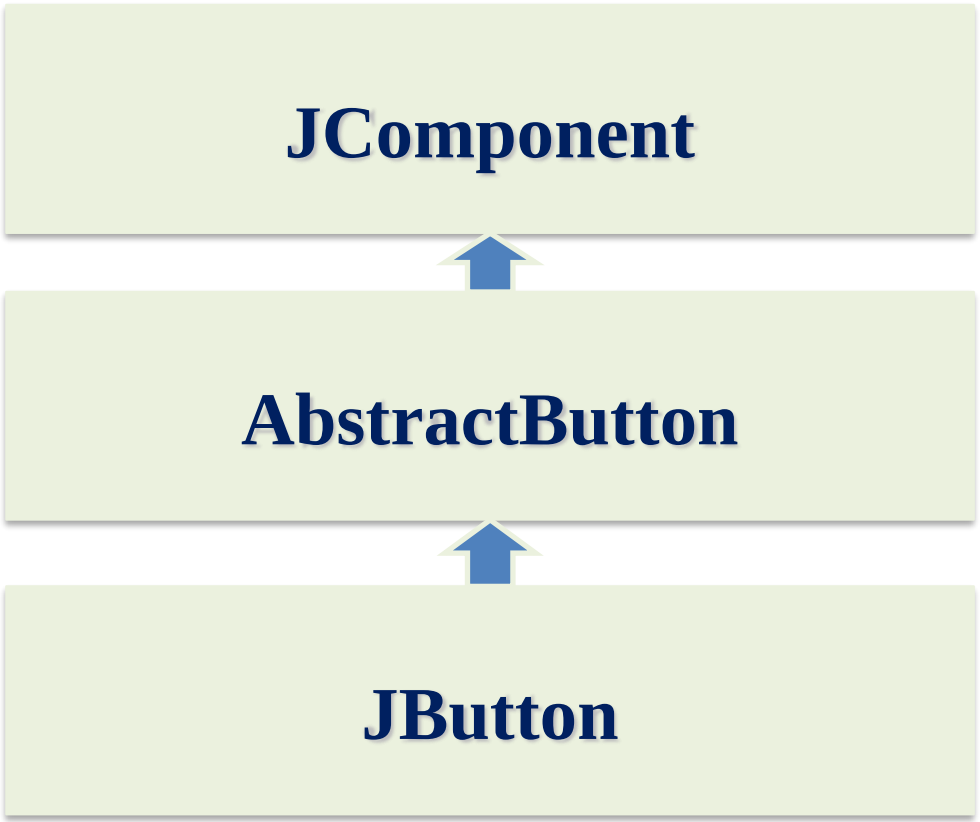


Swing的组件体系

4个Swing类直接派生自其相应的AWT类，
它们是Swing的4个顶级容器

Swing的组件体系

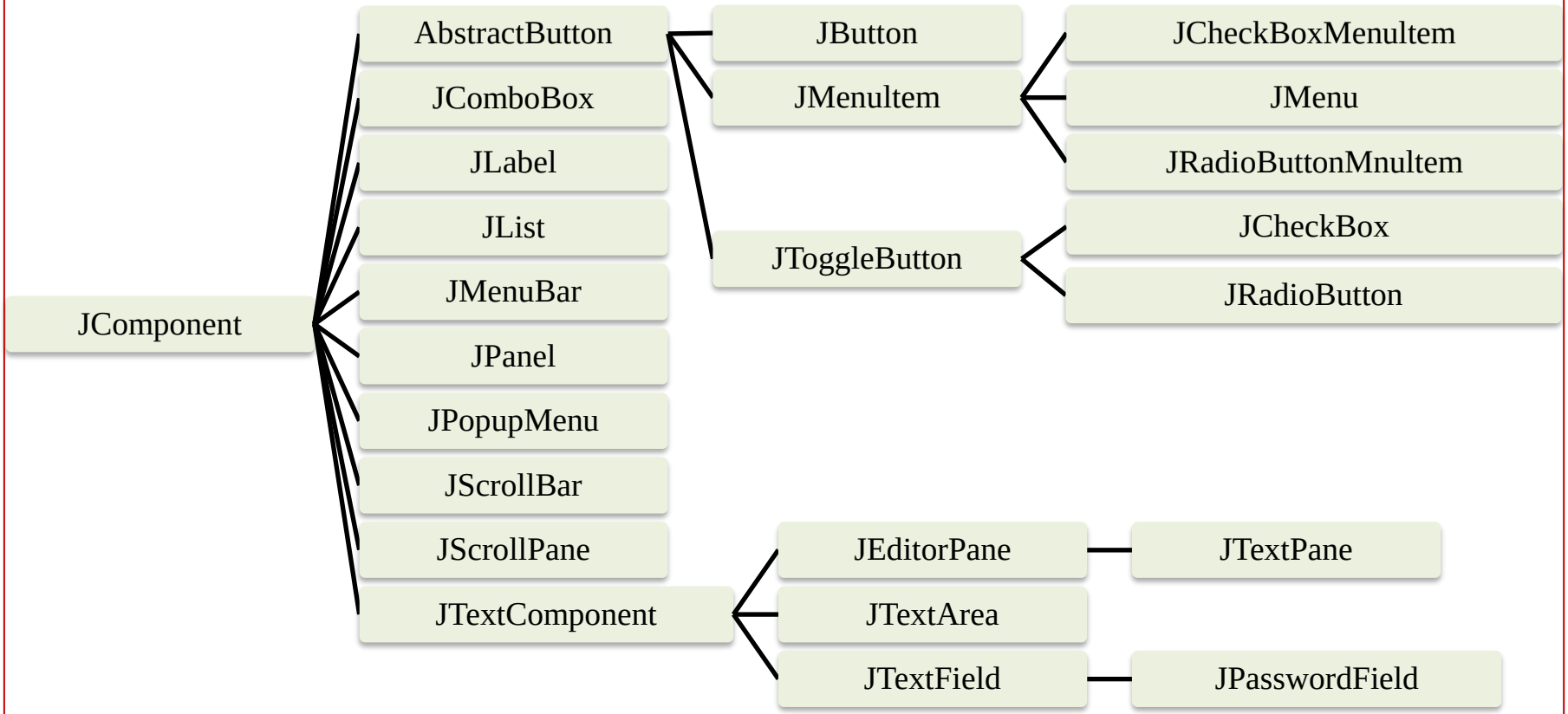
- 除上述4个顶级容器外，其他所有组件都扩充自JComponent类。例如：



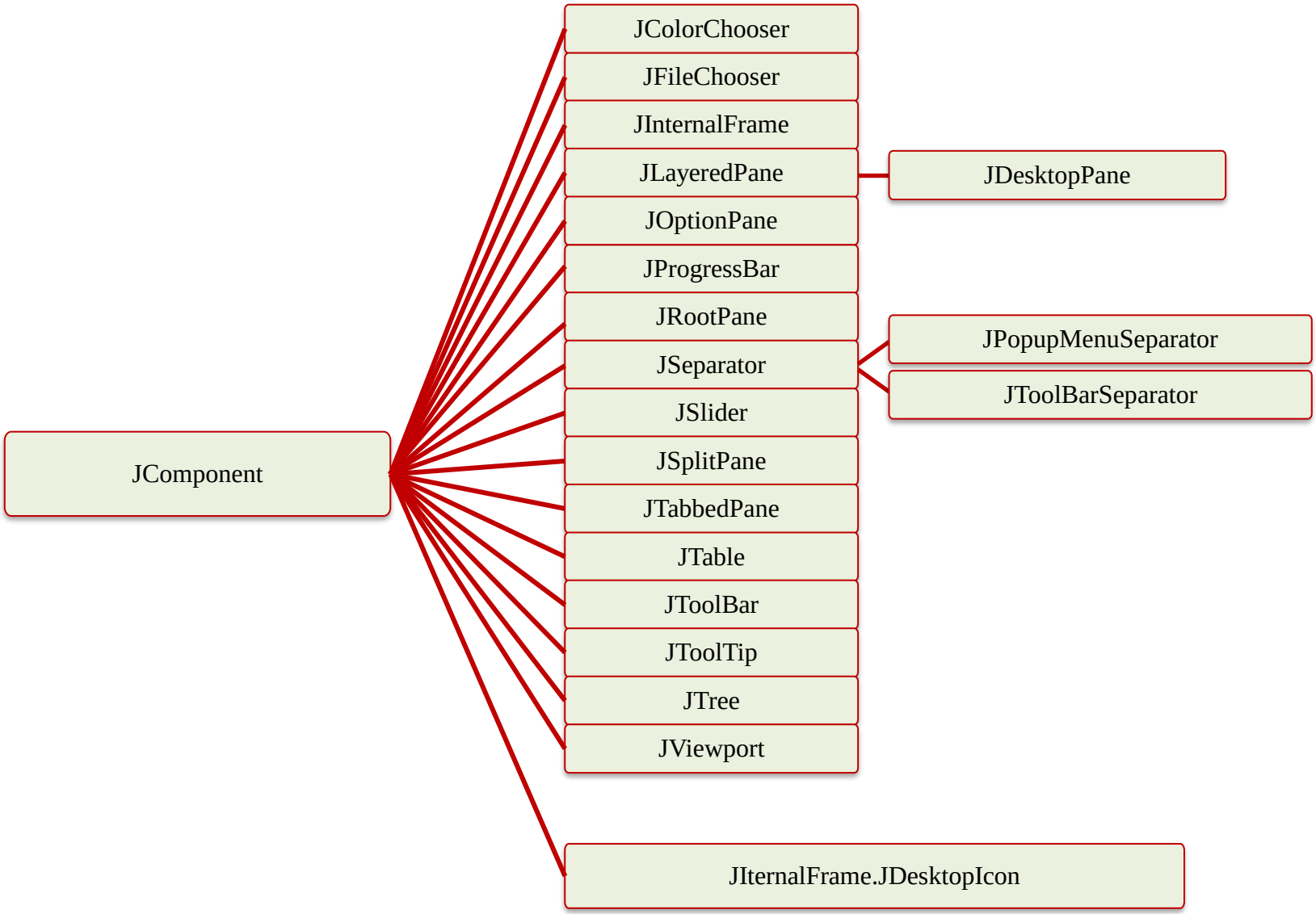
Swing的组件体系

- AWT组件和Swing组件的继承关系

Component Hierarchy—AWT Similar



Component Hierarchy—New and Expanded Components



JFrame容器

- JFrame是放置其他 Swing 组件的顶级容器
- JFrame用于在 Swing 程序中创建窗体
- 它的构造函数：
  - **JFrame()**
  - **JFrame(String title)**
- 组件必须添加至内容面板（content pane）上，而不是直接添加至 JFrame 对象，示例：
  - frame.getContentPane().add(b);

设置GUI应用程序的流程

- 引用需要的包和类
- 设置一个顶层的容器
- 根据需要为容器设置布局管理器或使用默认布局管理器
- 将组件添加到容器内，位置自行设计
- 为响应事件的组件编写事件处理代码

注意事项

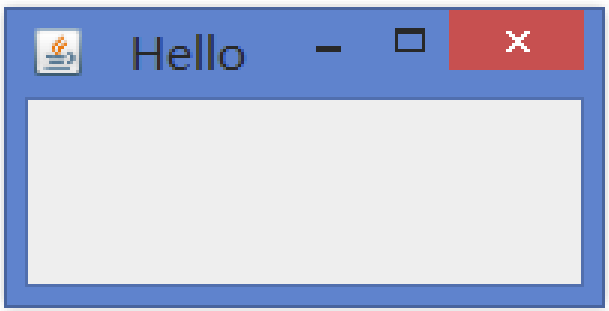
- 在创建GUI时，必须让轻构件完全包含在顶级Swing构件中，当然也可以包含在其他轻构件中，但必须有JDialog、JFrame、JWindow作为根容器

JFrame窗体类

- JFrame窗体类有一个构造方法JFrame(String title)，通过它可以创建一个以参数为标题的JFrame对象。
- 当JFrame被创建后，它是不可见的，必须通过以下方式使JFrame成为可见的：
  - 先调用setSize(int width,int height)显式设置JFrame的大小，或者调用pack()方法自动确定JFrame的大小，pack()方法会确保JFrame容器中的组件都会有与布局相适应的合理大小。
  - 然后调用setVisible(true)方法使JFrame成为可见的

创建一个简单窗体

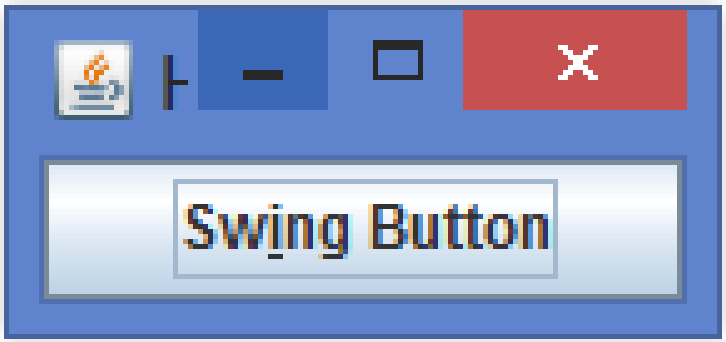
```
import javax.swing.*;

public class SimpleFrame1 {
    public static void main(String args[]) {
        JFrame jFrame = new JFrame("Hello");
        jFrame.setSize(200, 100); // 设置JFrame的宽和高
        jFrame.setVisible(true); // 使JFrame变为可见
    }
}
```



创建一个包含按钮的窗体

```
import javax.swing.*;
public class SimpleFrame2 {
    public static void main(String[] args) {
    JFrame myFrame=new JFrame("Hello");
    JButton myButton = new JButton("Swing Button");

    // 创建一个快捷键：用户按下Alt-i键等价于点击该Button
    myButton.setMnemonic('i');

    //设置鼠标移动到该Button时的提示信息
    myButton.setToolTipText("Press me");

    myFrame.add(myButton);

    //当用户选择myFrame窗体的关闭图标，将结束程序
    myFrame.setDefaultCloseOperation(myFrame.EXIT_ON_CLOSE);
    myFrame.pack();
    myFrame.setVisible(true);
  }
}
```

创建一个包含按钮的窗



设置关闭窗体的操作

- JFrame的setDefaultCloseOperation(int operation)方法用来决定如何响应用户关闭窗体的操作，参数operation有以下可选值：
  - JFrame.DO_NOTHING_ON_CLOSE：什么也不做。
  - JFrame.HIDE_ON_CLOSE：隐藏窗体，这是JFrame的默认选项。
  - JFrame.DISPOSE_ON_CLOSE：销毁窗体。
  - JFrame.EXIT_ON_CLOSE：结束程序。

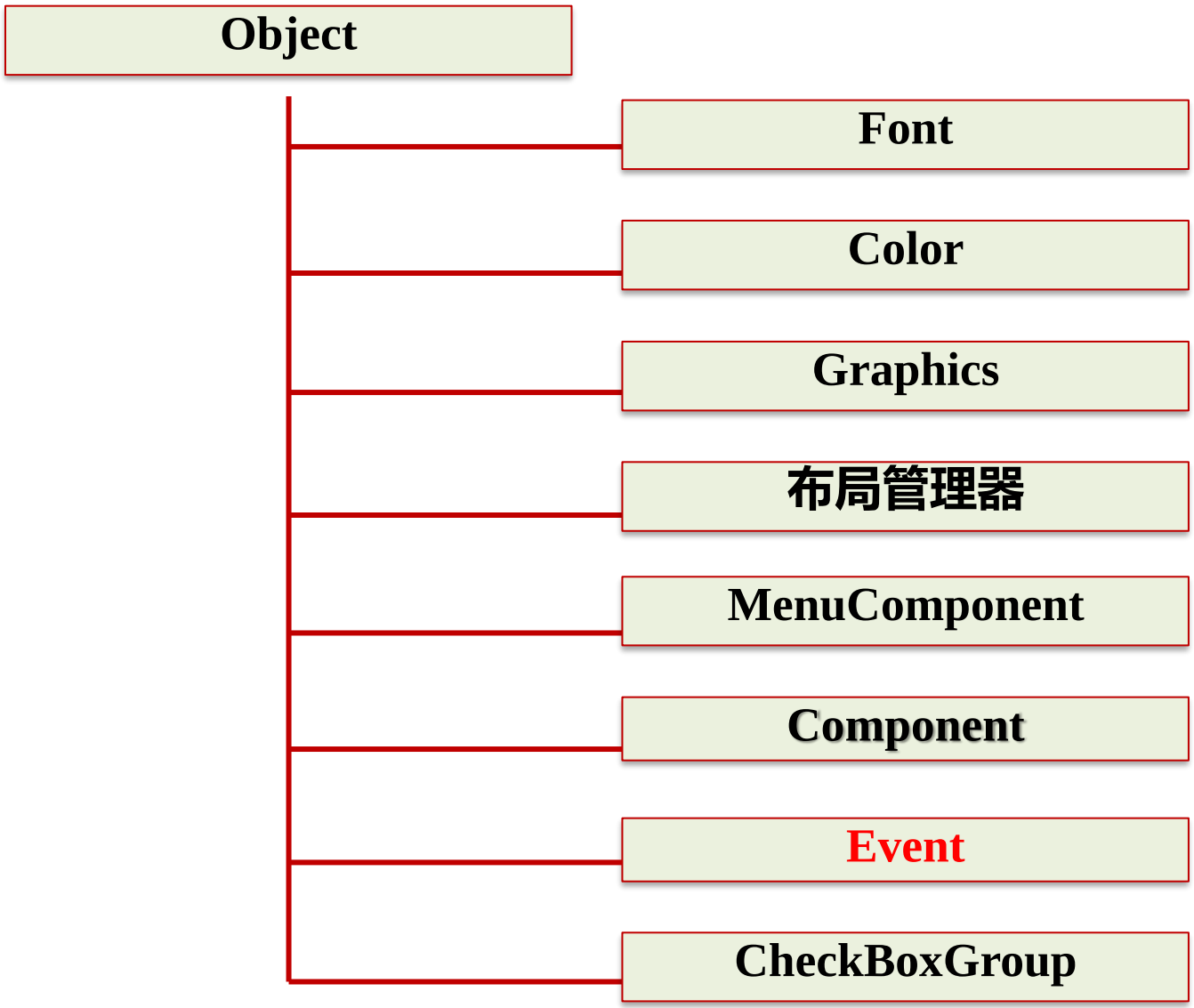
主要内容

- Java AWT
- Java Swing
- **Java 事件处理**
- Java 布局管理器
- Java 对话框
- Java 菜单
- Java 二维图形绘制（自学）
- Java 字体和颜色设置（自学）

Java事件处理

- Java AWT和Swing组件共用了**java.awt.event.***类和接口来处理事件（event）
- 所以涉及到事件处理时，应引入**java.awt.event.***

package java.awt



委托机制

- 自己不擅长的领域可以委托第三方机构帮我们打理
  - 实际工作中我们委托律师帮我们打官司
  - 委托房屋中介帮我们买卖房产，租赁房产
  - 委托第三方机构帮我们理财等

Java事件处理机制

- Java的事件处理使用的是“委托事件模型”
  - 事件：GUI中用户交互行为所产生的一种效果
  - 事件源：GUI中每个可能产生事件的组件
  - 事件处理（监听）者：接收事件并进行处理的方法。
  - *Java中所有的组件都从Component类中继承了将事件处理授权给监听者的方法*

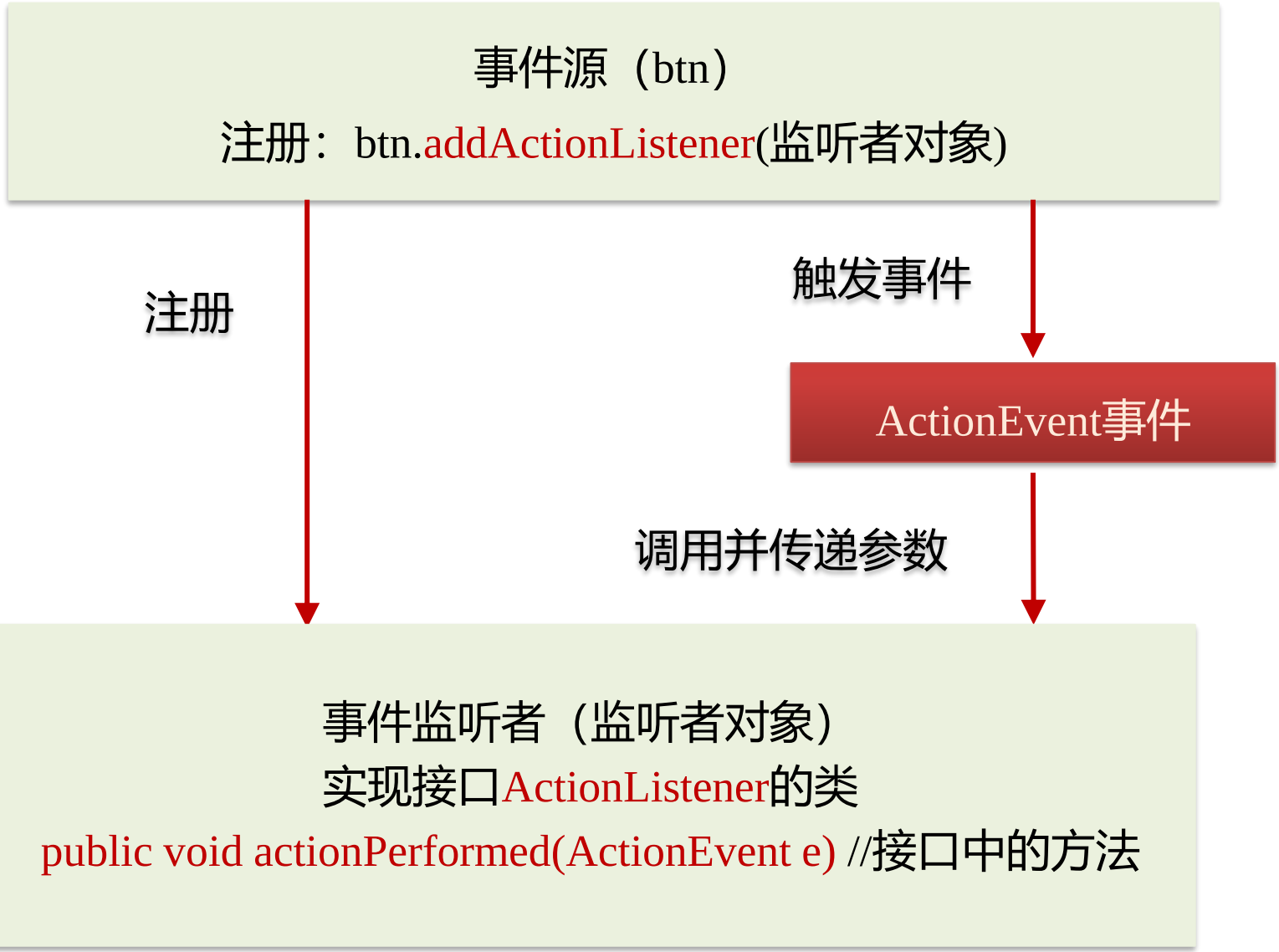
事件处理（监听）者

- 在Java的委托事件模型中，事件处理者可以是任何类的对象，*只要这个类实现了一个监听者接口，那么这个对象就可以处理事件。*也就是事件监听者具有监听和处理某类事件的功能
  - 在此，又一次体现了接口是一种能力

事件处理机制

- 事件处理模型分为3个部分：事件源对象、事件监听器对象与事件对象；
- 能产生事件的组件叫做事件源，如按钮；
- 事件监听器注册在事件源对象（例如按钮或包含按钮的容器）上，用来监听事件是否发生，当事件发生时将调用事件处理方法解决问题；
- 事件对象用来封装已发生的事件的信息，在事件发生后，将信息传递给事件处理者，事件处理者中的相应方法处理事件；

事件处理机制


事件源（btn）
注册：btn.addActionListener(监听者对象)
注册
触发事件
ActionEvent事件
调用并传递参数
事件监听者（监听者对象）
实现接口ActionListener的类
public void actionPerformed(ActionEvent e) //接口中的方法

Java事件处理(1)：注册监听者

- 在每一个事件处理者的程序中都需要编写3段代码
- （1）在生成了组件后（例如按钮），确定该事件源要响应的事件（一个或者多个），注册相应的事件监听者（一个或者多个）。
  - 例如:
  - someComponent.addActionListener (instance of MyClass);

Java事件处理（2）：编写监听者类

- （2）：在事件处理者的类声明中，编写代码说明该类实现了一个监听者接口，或继承了一个实现了监听者接口的类。如：

```
class MyClass implements ActionListener  {
… …
}
```

Java事件处理（3）：实现了监听者接口的方法

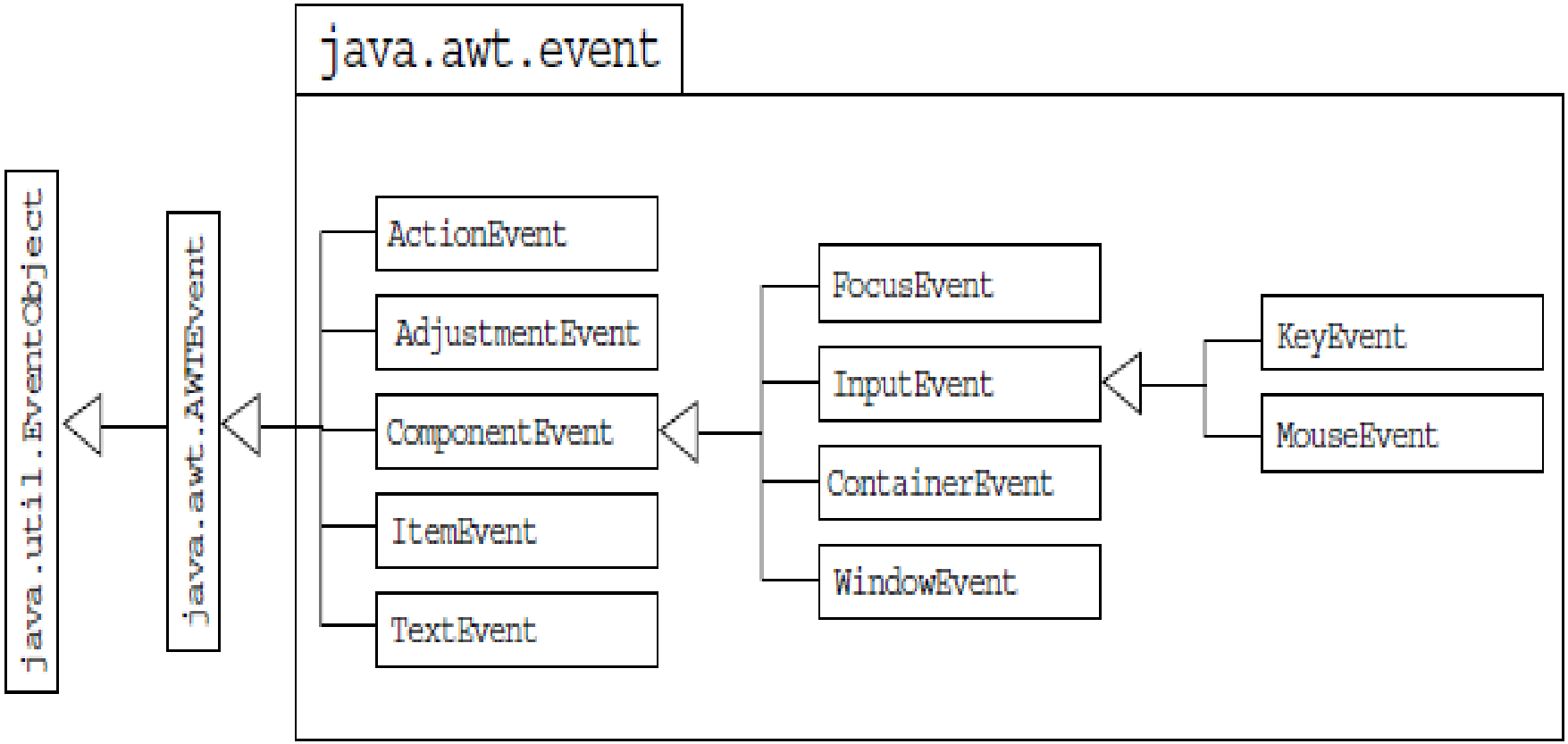
- （3）在事件处理者的类中，都有实现了监听者接口的方法，在该方法中，编写业务逻辑，处理事件。如：

```
public void actionPerformed(ActionEvent   e)
{…
  …
}
```

```
public class J_Test2 extends JFrame{
    public J_Test2() {
        super("Test anonymous inner class");
        Container container = getContentPane();
        container.setLayout(new FlowLayout(FlowLayout.CENTER));
        JButton b = new JButton("Press me");
        container.add(b);
        b.addActionListener(new MyClass());
        setSize(100, 80);
        setVisible(true);
    }
    public static void main(String[] args) {
        J_Test1 application = new J_Test1();
        application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    class MyClass implements ActionListener{
        public void actionPerformed(ActionEvent e) {
            System.out.println("The button is pressed");
        }
    }
}
```

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class J_Test1 extends JFrame {
    public J_Test1() {
        super("Test anonymous inner class");
        Container container = getContentPane();
        container.setLayout(new FlowLayout(FlowLayout.CENTER));
        JButton b = new JButton("Press me");
        container.add(b);
        b.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.out.println("The button is pressed");
            }
        });
        setSize(100, 80);
        setVisible(true);
    }
    public static void main(String[] args) {
        J_Test1 application = new J_Test1();
        application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Event Categories



Method Categories and Interfaces

Category	Interface Name	Methods
Action	ActionListener	actionPerformed(ActionEvent)
Item	ItemListener	itemStateChanged(ItemEvent)
Mouse	MouseListener	mousePressed(MouseEvent) mouseReleased(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mouseClicked(MouseEvent)
Mouse motion	MouseMotionListener	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
Key	KeyListener	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)

Method Categories and Interfaces

Category	Interface Name	Methods
Focus	FocusListener	focusGained(FocusEvent) focusLost(FocusEvent)
Adjustment	AdjustmentListener	adjustmentValueChanged (AdjustmentEvent)
Component	ComponentListener	componentMoved(ComponentEvent) componentHidden(ComponentEvent) componentResized(ComponentEvent) componentShown(ComponentEvent)

Method Categories and Interfaces

Category	Interface Name	Methods
Window	WindowListener	windowClosing(WindowEvent) windowOpened(WindowEvent) windowIconified(WindowEvent) windowDeiconified(WindowEvent) windowClosed(WindowEvent) windowActivated(WindowEvent) windowDeactivated(WindowEvent)
Container	ContainerListener	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)
Text	TextListener	textValueChanged(TextEvent)

主要的接口类型及其作用

ActionListener	按钮、菜单、双击列表
AdjustmentListener	滚动条
ComponentListener	控件状态改变
ContainerListener	控件加入或删除
FocusListener	焦点得失
ItemListener	复选框、列表、可选菜单
KeyListener	键盘输入
MouseListener	鼠标输入
MouseMotionListener	鼠标拖拽
TextListener	文本区文本改变
WindowListener	Window状态改变

Frame类处理窗口关闭事件

- 当一个窗口打开、关闭、最小化时都会引发窗口事件（WindowEvent），实现窗口事件监听的接口是WindowListener接口

Frame类处理窗口关闭事件

WindowListener接口中包含以下7个方法：

```
public void windowActivated(WindowEvent e);
public void windowClosed(WindowEvent e);
public void windowClosing(WindowEvent e);
public void windowDeactivated(WindowEvent e);
public void windowDeiconified(WindowEvent e);
public void windowIconified(WindowEvent e);
public void windowOpened(WindowEvent e);
```

Frame类处理窗口关闭事件

- 如果一个窗口通过实现WindowListener接口来处理窗口事件，则需要实现接口中的7个方法

Frame类处理窗口关闭事件

```
import java.awt.*;  import java.awt.event.*;
public class TestFrame {
public static void main(String[] s){
    Frame f = new Frame("First Frame");
    f.setLocation(100, 100);
    f.setSize(200,200);
    f.setBackground(Color.red);
    f.setVisible(true);
    f.addWindowListener(new WindowAdapter(){
  public void windowClosing(WindowEvent e){
                    System.exit(0);}});
}}
```

事件适配器

- 为方便起见，Java为某些监听者接口提供了适配器类（XXXAdapter），当需要对某种事件进行处理时，只需要让事件处理类继承事件所对应的适配器类，重写需要关注的方法即可，而不必实现无关的方法

事件适配器

```
public abstract class WindowAdapter extends Object
    implements WindowListener,WindowStateListener,
                                        WindowFocusListener{

        … …

        …
}
```

事件适配器

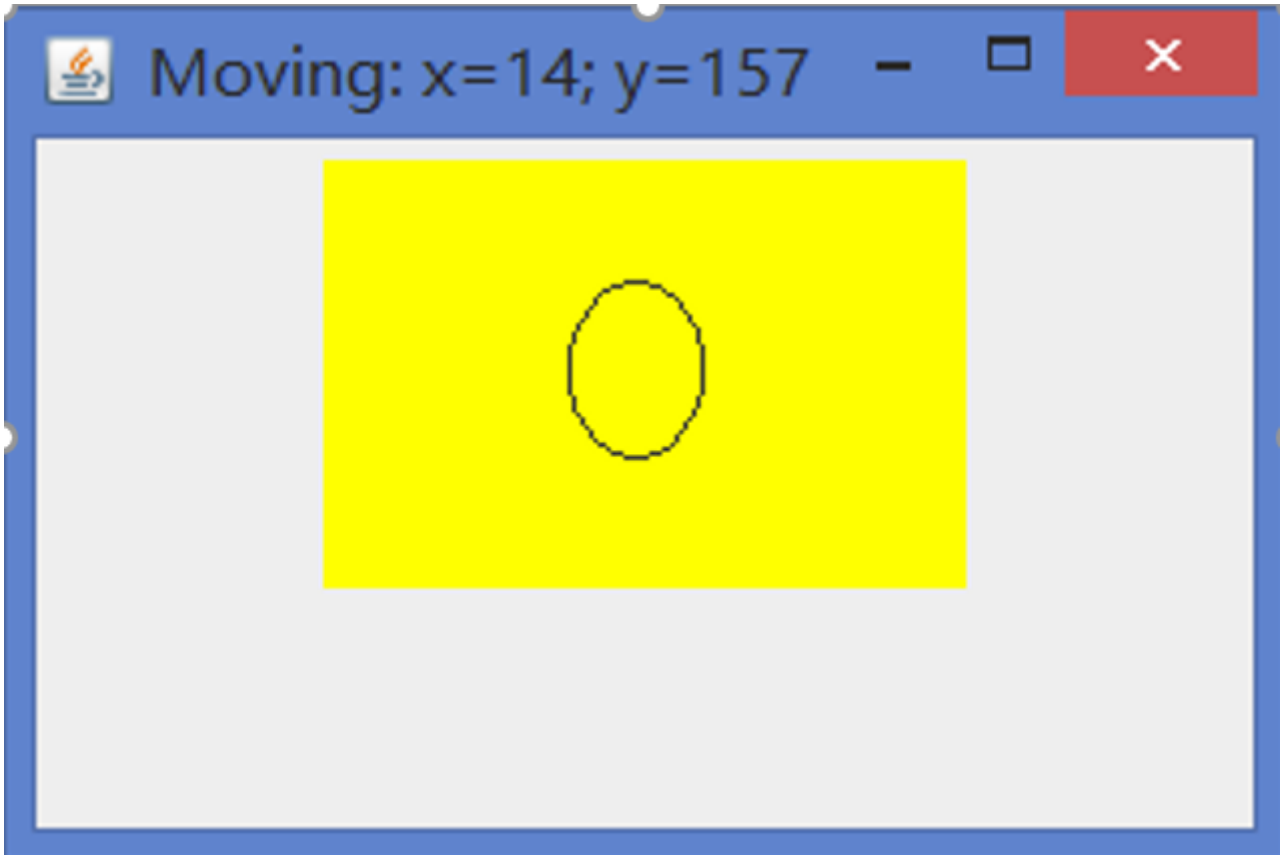
- 事件适配器包括以下几种
  - –ComponentAdapter
  - –ContainerAdapter
  - –FocusAdapter
  - –KeyAdapter
  - –MouseAdapter
  - –MouseMotionAdapter
  - –WindowAdapter

事件适配器

Event-adapter class	Implements interface
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
FocusAdapter	FocusListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
WindowAdapter	WindowListener

示例

- SelfContainedPanel.java
- SelfContainedPanelTest.java



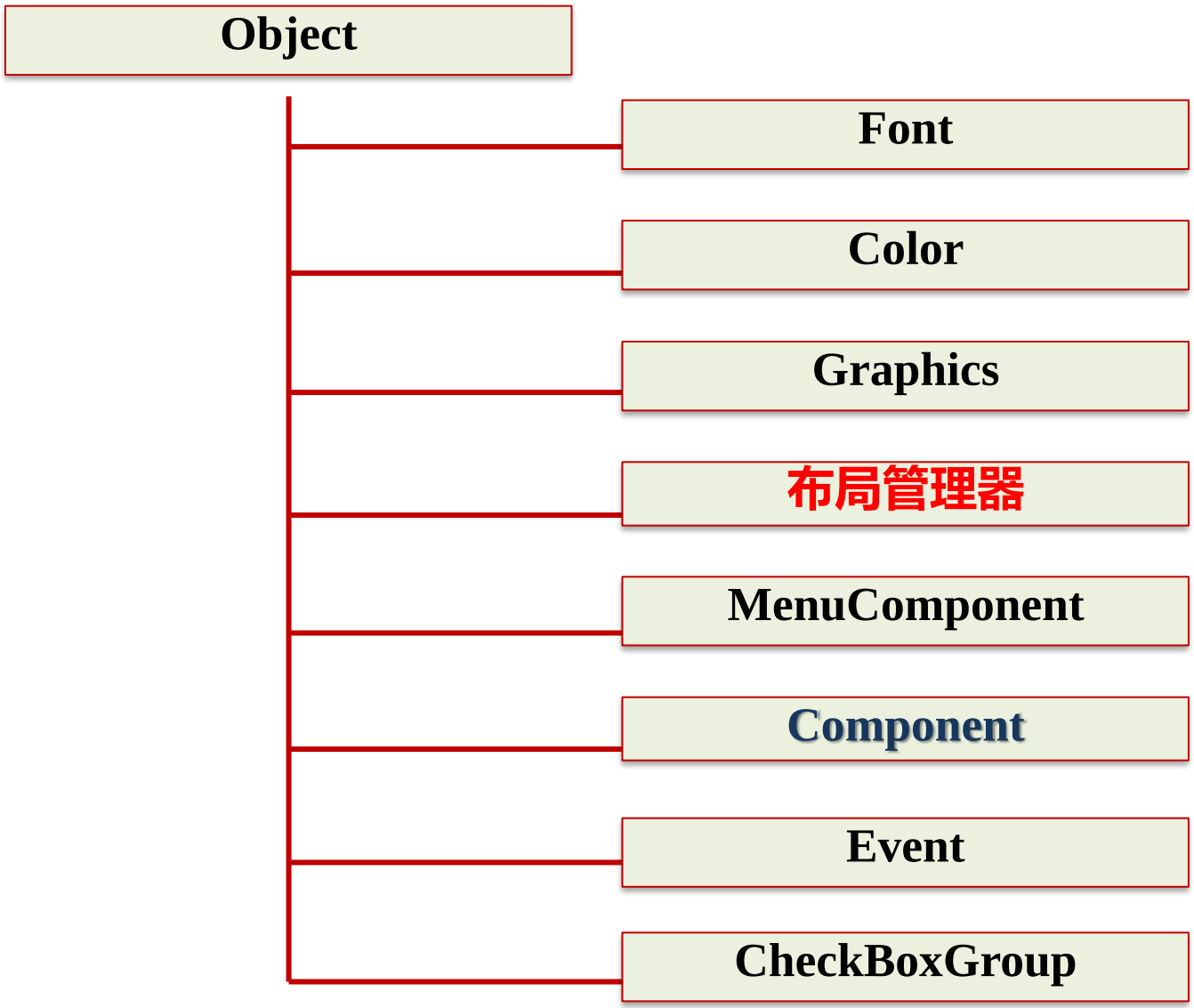
小结：事件处理

- 确认触发的事件，取得事件类（如ActionEvent）的名字，并删掉其中的“Event”，加入“Listener”字样
- 实现上面的接口，针对想要捕获的事件编写方法代码
- 为事件处理器（监听者接口）创建一个对象，让自己的组件和方法完成对它的注册

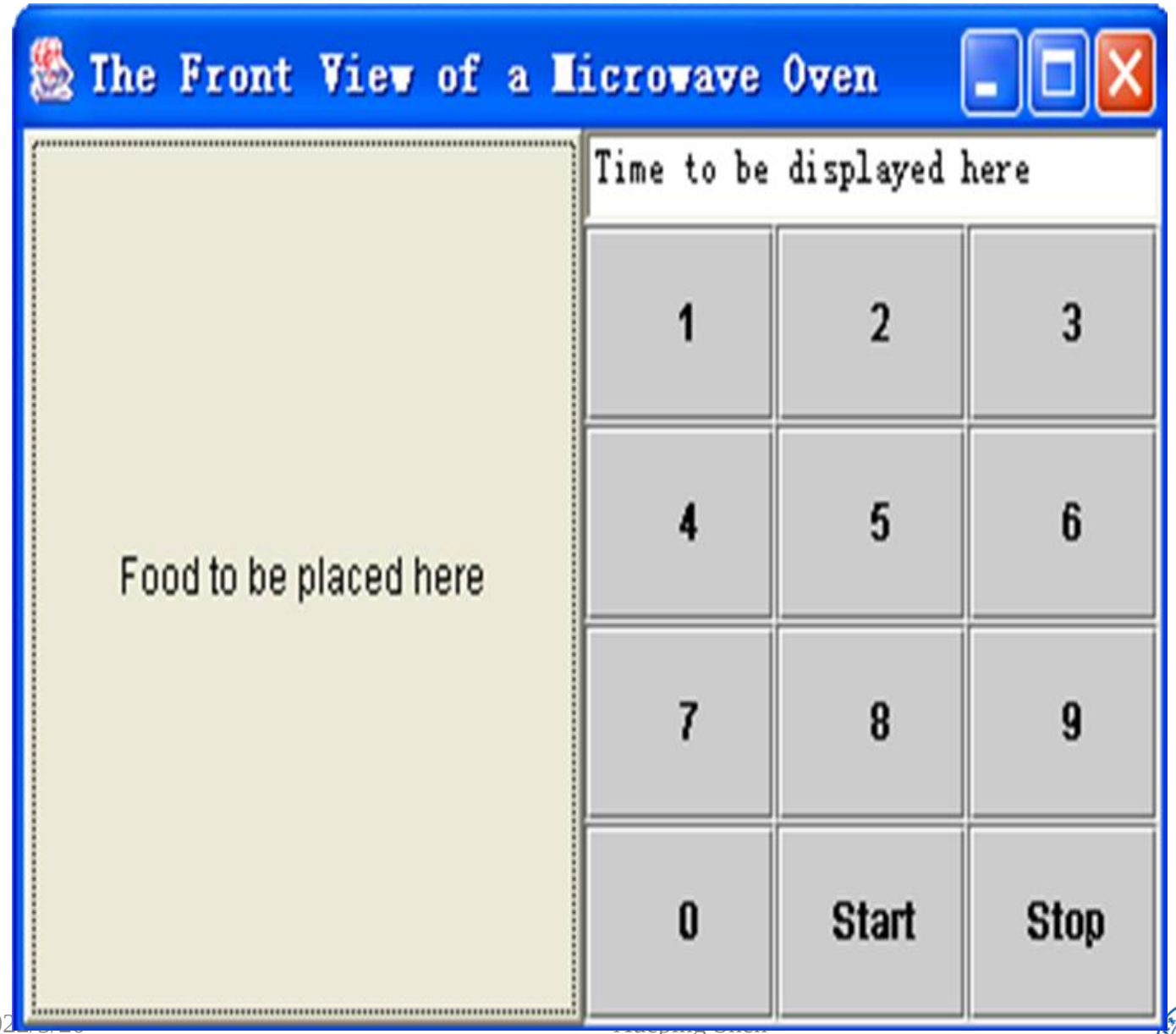
主要内容

- Java AWT
- Java Swing
- Java 事件处理
- **Java 布局管理器**
- Java 对话框
- Java 菜单
- Java 二维图形绘制（自学）
- Java 字体和颜色设置（自学）

Java 布局管理器



Java 布局管理器



Java里有六种布局管理器

- FlowLayout(顺序布局)
- BorderLayout（边界布局）
- GridLayout（网格布局）
- BoxLayout
- CardLayout（卡片布局）
- GridBagLayout(网格包布局)
- null布局

FlowLayout布局

- 每个部件从左到右、从上到下，依据容器的大小逐行在容器中顺序摆放
- FlowLayout是Applet类和Panel类、JPanel类的默认布局方式

FlowLayout布局

- FlowLayout中的主要方法
  - 构造函数

    FlowLayout();

    FlowLayout(int align);

    FlowLayout(int align, int hgap, int vgap);
  - 设置布局

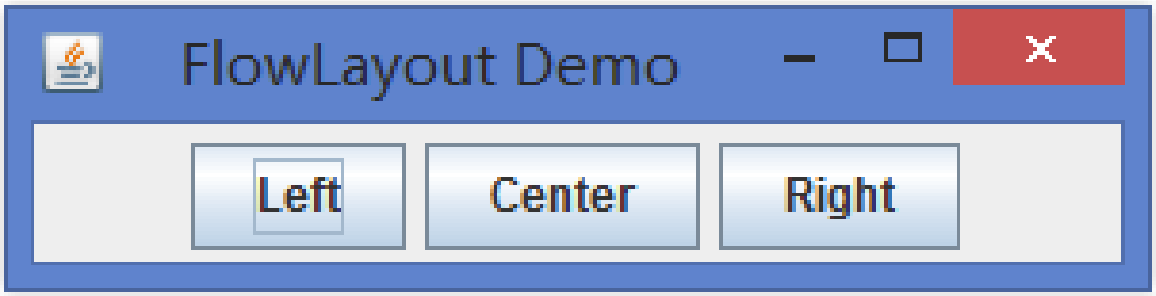
    setLayout(new FlowLayout());

对齐方式
FlowLayout.RIGHT
FlowLayout.LEFT
FlowLayout.CENTER

表示组件之间间隔

FlowLayout布局

- 示例：FlowLayoutDemo.java



BorderLayout布局

- BorderLayout布局方式是将组件按东、南、西、北、中五种方位放置在容器中
- 如果东南西北某个位置上没有放置组件，则该区域会被中间区域和相关的某个位置区域自动充满
- BorderLayout是Frame、JFrame和Dialog、JApplet的默认布局方式

BorderLayout布局

- 构造函数

1. BorderLayout();
2. BorderLayout(int hgap, int vgap)

- 设置布局

  setLayout(new BorderLayout());

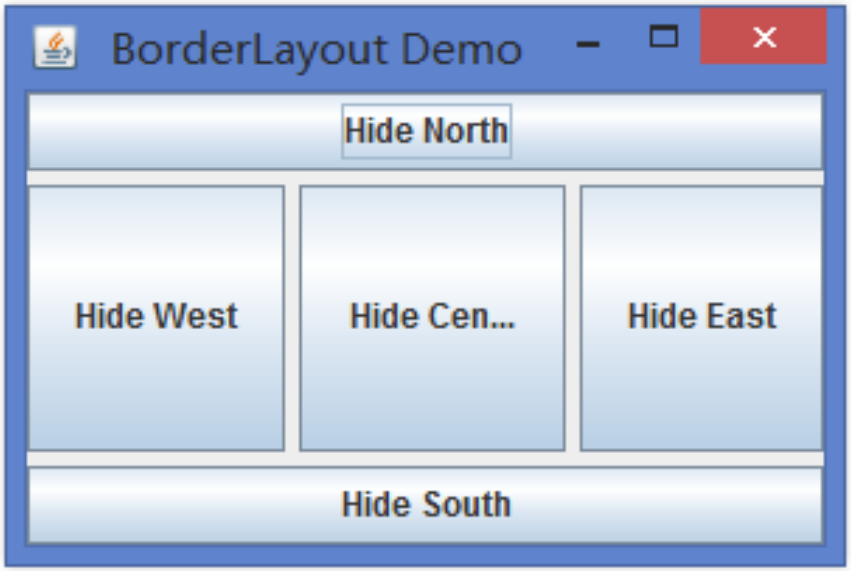
- 添加组件到指定布局

  addLayoutComponent(Component comp,Object constraints);

BorederLayout.EAST
BorederLayout.SOUTH
BorederLayout.WEST
BorederLayout.NORTH
BorederLayout.CENTER

示例：

- BorderLayoutDemo.java



GridLayout布局

- GridLayout布局
  - 将每个组件放置在rows行及columns列中，即将容器分成大小相等的矩形域，当一排满了，就从下一行开始
- GridLayout的构造函数

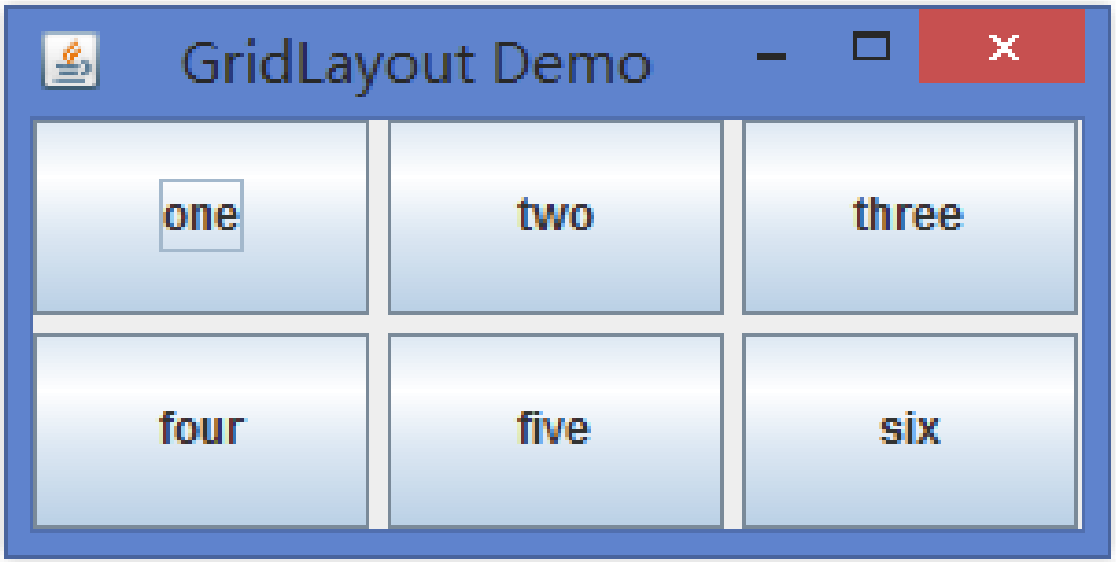
```
GridLayout(int rows, int cols);
GridLayout(int rows, int cols, int hgap, int vgap);
```

- 设置布局

```
setLayout(new GridLayout(3,3,5,5));
```

示例

- GridLayoutDemo.java



CardLayout布局

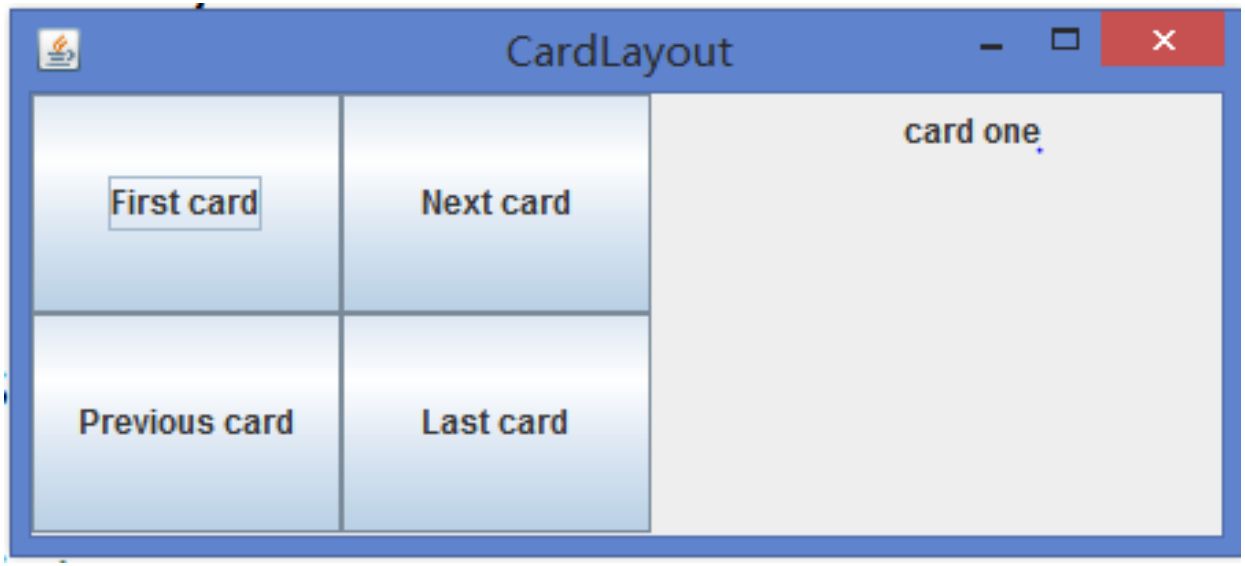
- CardLayout布局管理是把容器的所有组件当成一叠卡片，卡片布局方式中只有其中的一个组件，即一张卡片被显示出来，其余组件是不可见的
  - 构造函数
    - CardLayout(int hgap, int vgap);
    - CardLayout();

CardLayout布局

- 常用方法
  - addLayoutComponent(Component comp,Object constraints);
  - first(Container parent);
  - last(Container parent);
  - next(Container parent);
  - show(Container parent, String name);

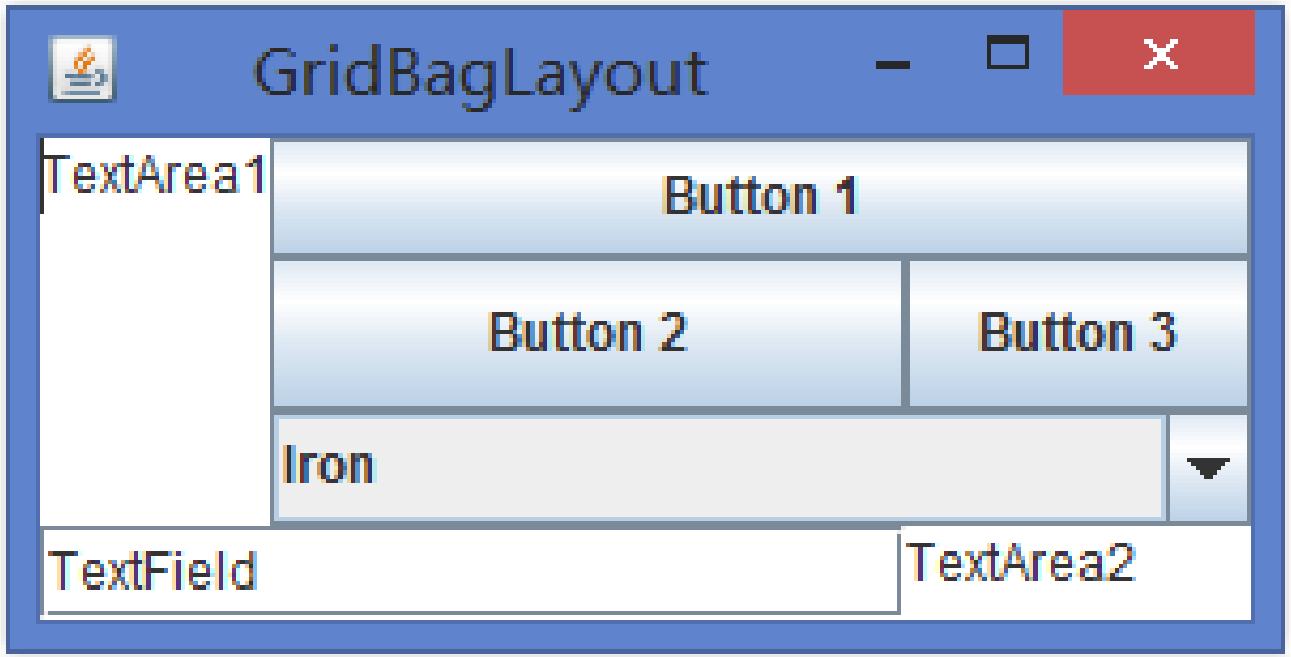
示例

- CardDeck.java



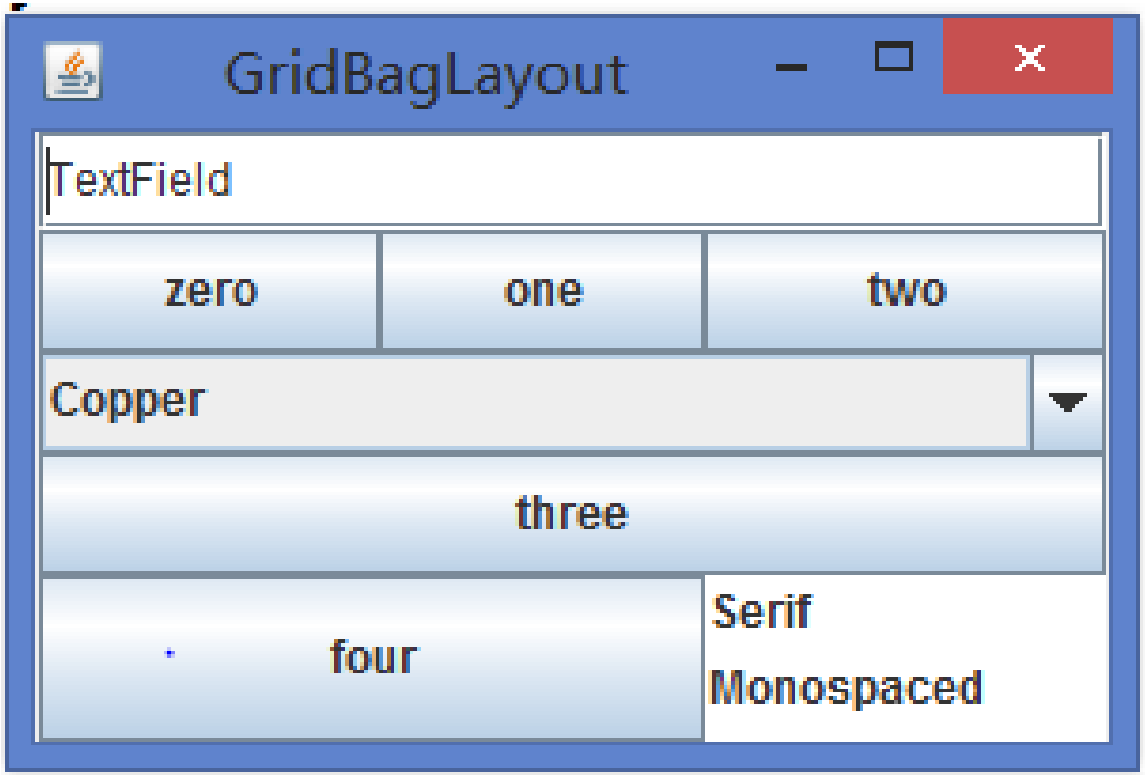
GridBagLayout布局

- GridBagDemo.java



GridBagLayout布局

- GridBagDemo2



null布局

- null布局又称为空布局
- 对一个容器（Container）而言，可以用下面方式设置其布局管理器为null
  - **public void setLayout(null);**
- 这样，容器内的组件（Component）可以利用下面的方法来设置其大小和位置
  - **public void setBounds(int x, int y,int width,int height);**

x,y表示组件左上角在容器中的坐标，width和height表示组件的宽和高

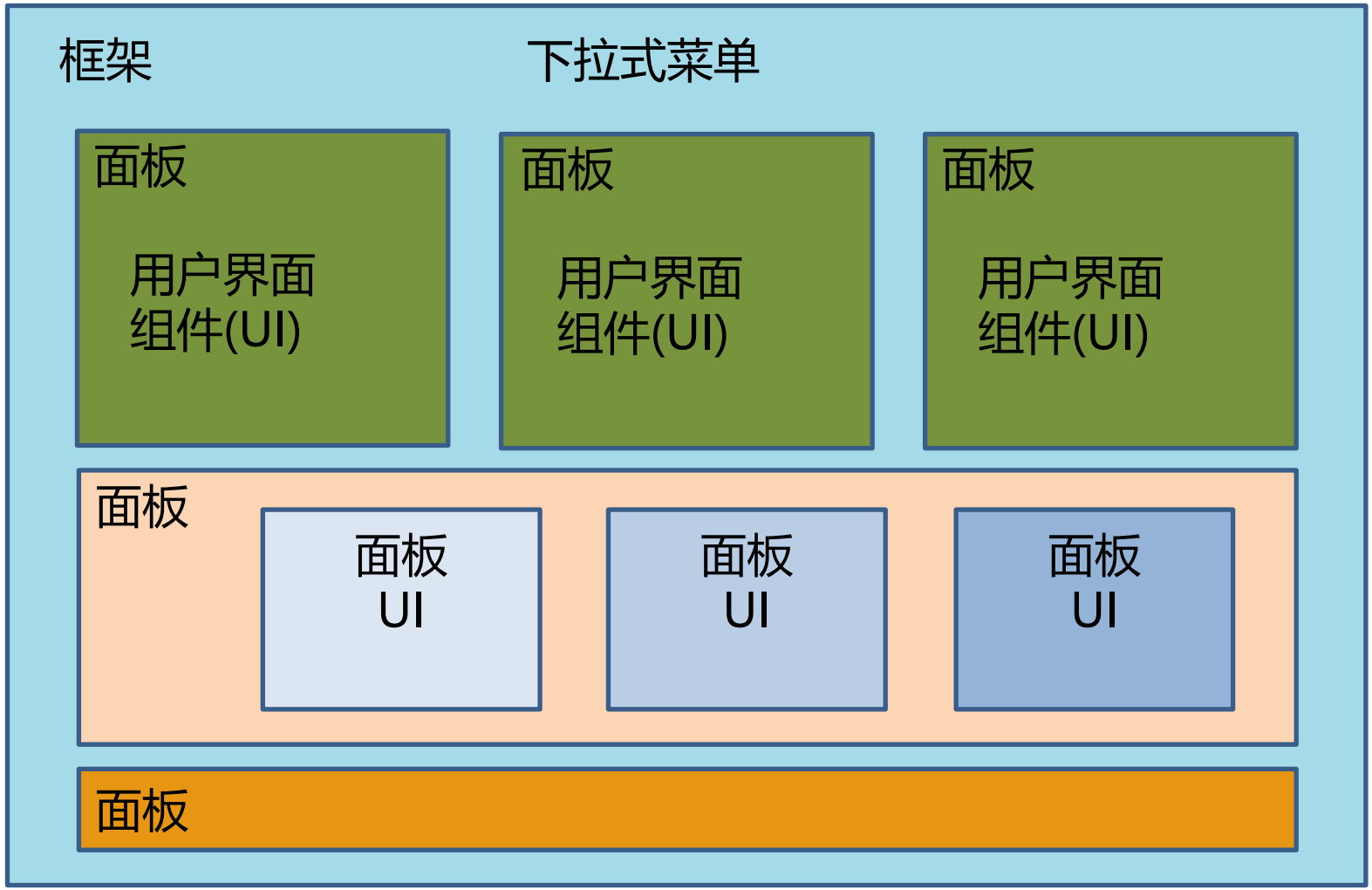
容器的嵌套

- 容器的嵌套：在实际的编程中，我们经常遇到向容器中添加容器，这就是容器的嵌套
- **JPanel类型的容器常常扮演这种角色，在多种布局方式之间起到了一种桥梁的作用**

容器的嵌套

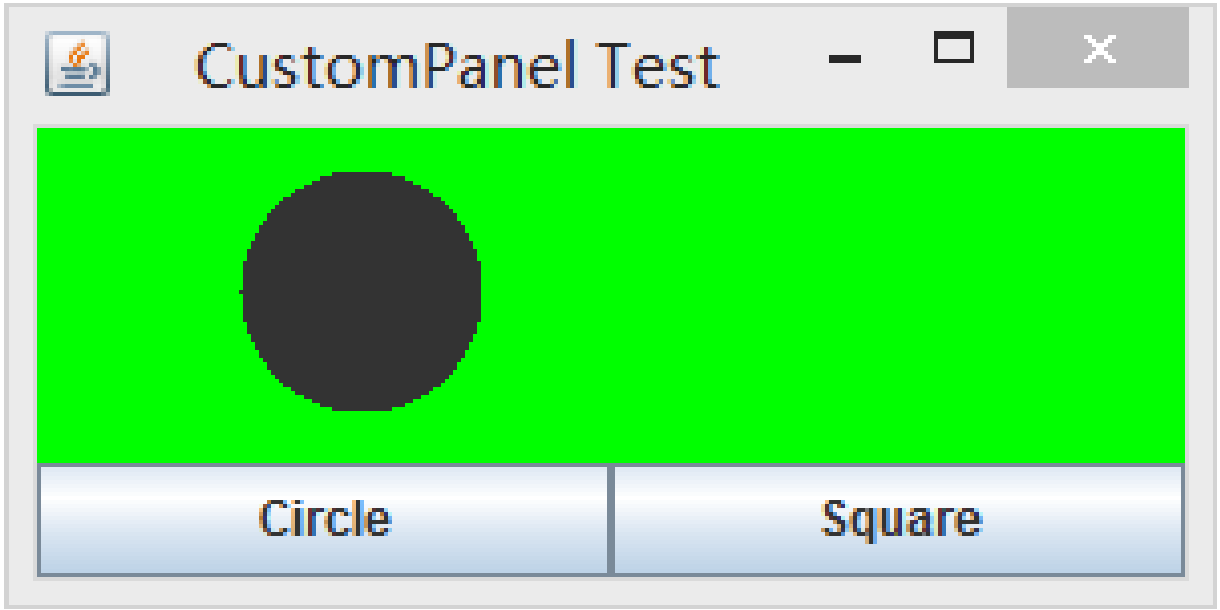
- 面板容器JPanel
  - JPanel 组件是一个中间容器
  - 用于将小型的轻量级组件组合在一起
  - JPanel 的缺省布局为 FlowLayout

容器的嵌套

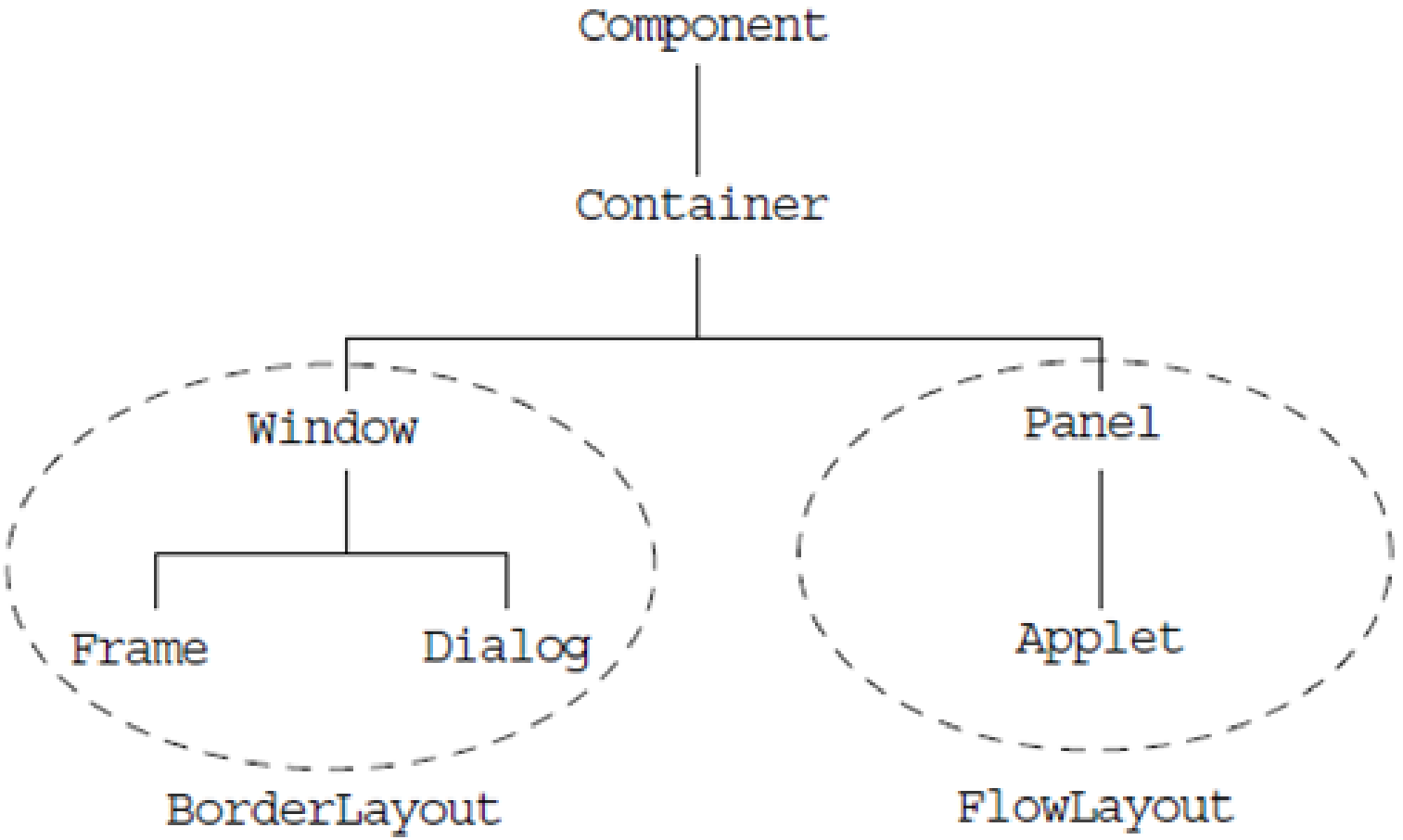


示例

- CustomPanelTest



默认布局管理器


Component
Container
Window
Frame
Dialog
BorderLayout
Panel
Applet
FlowLayout

主要内容

- Java AWT
- Java Swing
- Java 事件处理
- Java 布局管理器
- Java 对话框
- Java 菜单
- Java 二维图形绘制（自学）
- Java 字体和颜色设置（自学）

对话框（JDialog）

- JDialog表示对话框。对话框是在现有窗口的基础上弹出的另一个窗口。
- 对话框可用来处理个别细节问题，使得这些细节不与原先窗口的内容混在一起。

对话框（JDialog）

- 对话框(JDialog)是与框架类似的有边框、有标题、可移动的独立存在的容器，默认的布局方式为BorderLayout布局
- 对话框不能作为程序的最外层容器，也不能包含菜单栏，它被框架所拥有并由框架负责弹出
- 默认的对话框是不显示的，需用setVisible();
- 对话框分为模态对话框和非模态对话框两种

模态对话框、非模态对话框、文件对话框

- 模态对话框：即对话框显示时程序被阻塞，在对话框被关闭之前，其他窗口无法接受任何形式的输入。
- 非模态对话框：没有上述限制
- **文件对话框：**是用于文件选择的对话框，允许用户对目录或文件进行浏览和选择。

对话框（JDialog）

- 对话框（JDialog）具有以下形式的构造方法：
  - public JDialog(Frame owner,String title,boolean modal)
  - owner表示对话框所属的Frame，参数title表示对话框的标题，参数modal有两个可选值：
  - 参数modal为true：表示模式对话框，这是JDialog的默认值。
  - 参数modal为false：表示非模式对话框。
- 当对话框被关闭时，通常不希望结束整个应用程序，因此只需调用JDialog的dispose()方法销毁对话框，从而释放对话框所占用的资源。

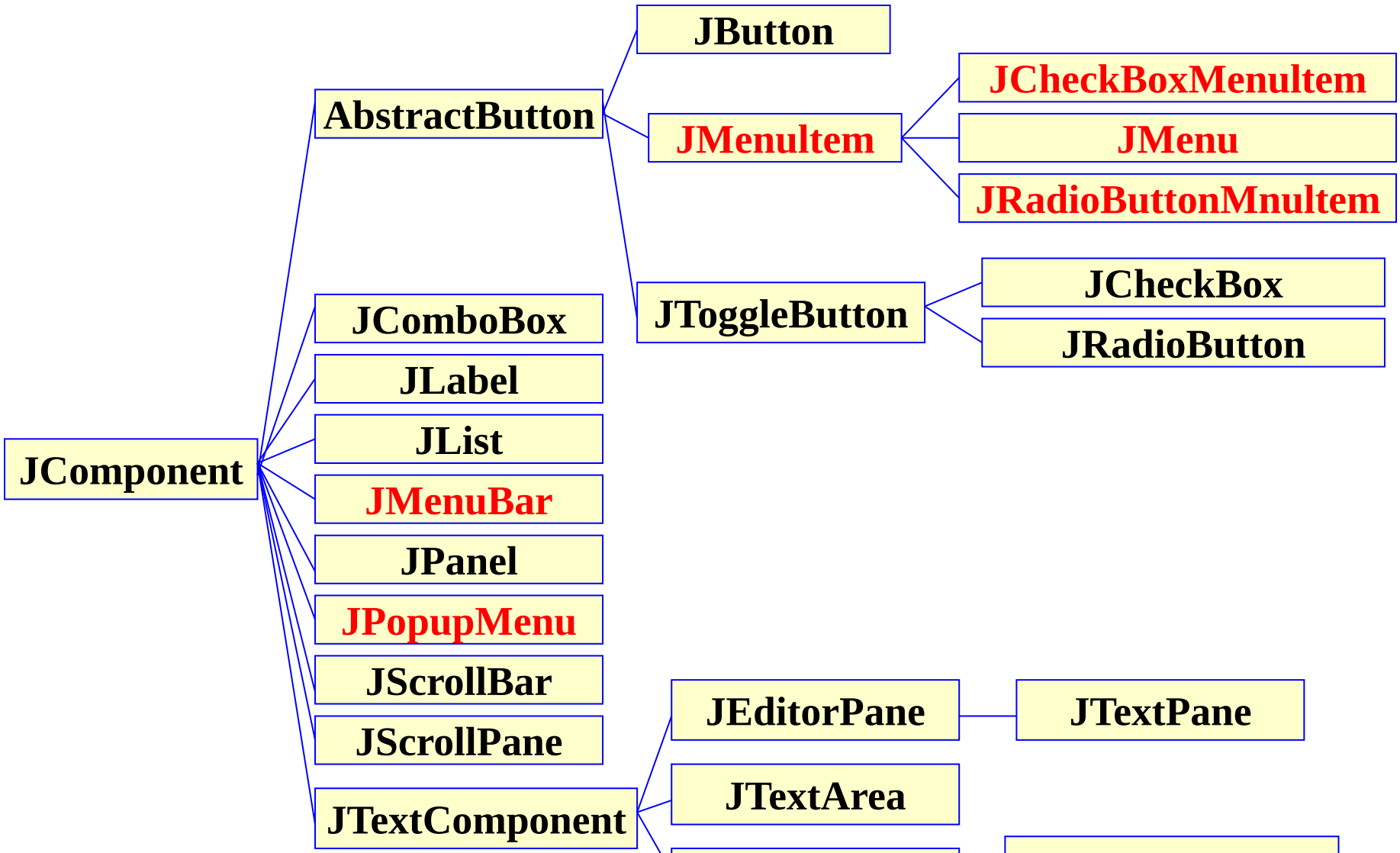
示例代码：

- DialogDemo.java

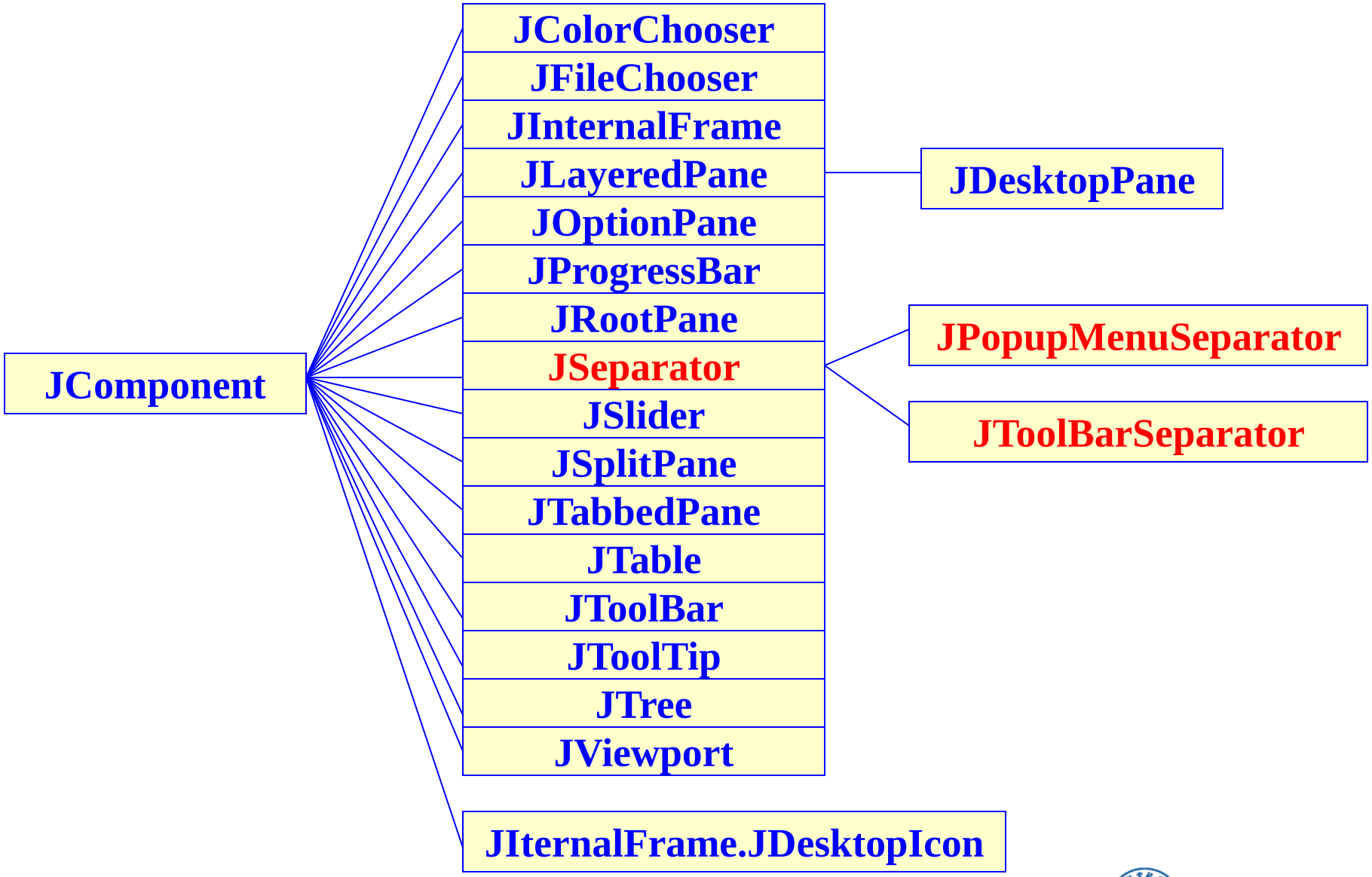
主要内容

- Java AWT
- Java Swing
- Java 事件处理
- Java 布局管理器
- Java 对话框
- Java 菜单
- Java 二维图形绘制（自学）
- Java 字体和颜色设置（自学）

Java 菜单



Java 菜单


JComponent
JColorChooser
JFileChooser
JInternalFrame
JLayeredPane
JOptionPane
JProgressBar
JRootPane
JSeparator
JSlider
JSplitPane
JTabbedPane
JTable
JToolBar
JToolTip
JTree
JViewport
JDesktopPane
JPopupMenuSeparator
JToolBarSeparator
JIternalFrame.JDesktopIcon

Java 菜单

At least three different menu types: MenuBar, Menu, MenuItem

MenuBar: The strip along the top of the window

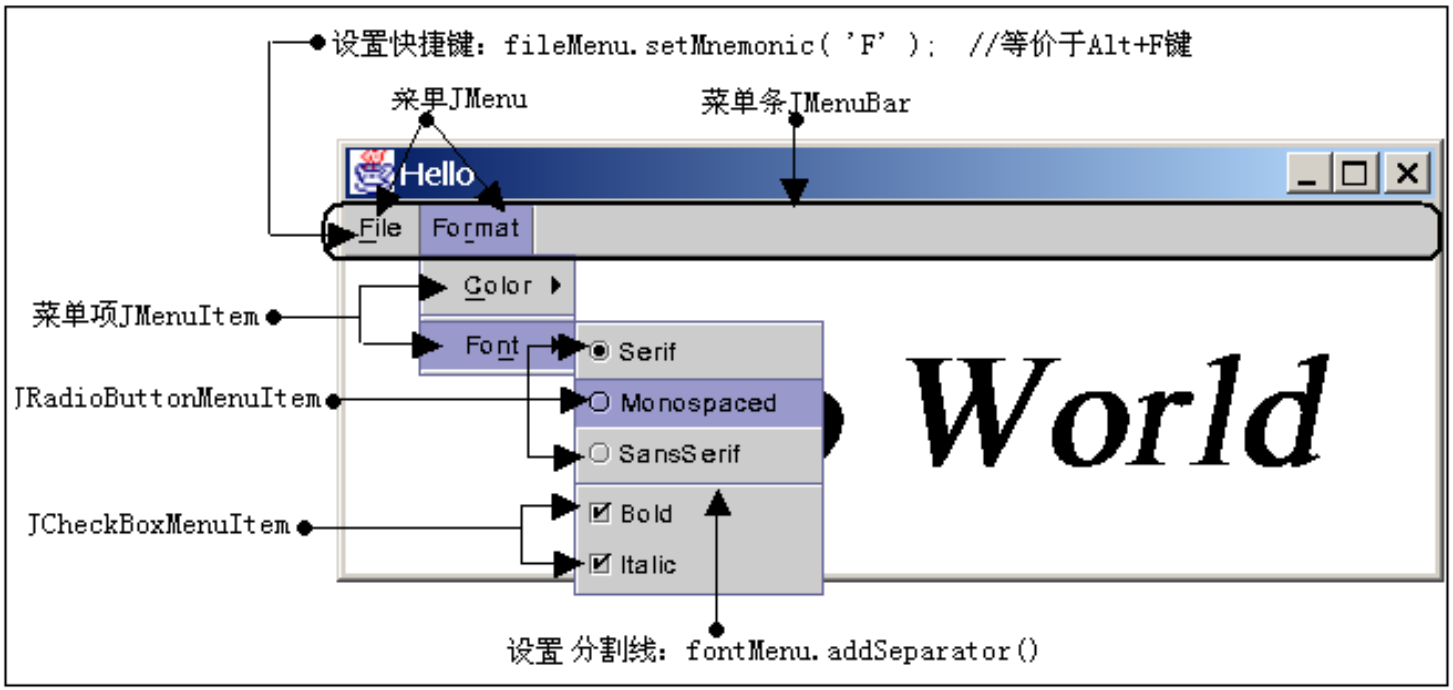
Menus: contained in MenuBar

MenuItems: contained in Menus

Java 菜单的组织方式

- 支持菜单的组件有JFrame、JDialog和Japplet。这些组件中有一个setMenuBar(JMenuBar bar)方法，可以用这个方法来设置菜单条。
- 一个菜单条JMenuBar中可以包含多个菜单Jmenu
- 一个菜单JMenu中可以包含多个菜单项JMenuItem。
- 当用户选择了某个菜单项，就会触发一个ActionEvent事件，该事件由ActionListener负责处理。

- JMenuItem有两个子类：JRadioButtonMenuItem和JCheckBoxMenuItem，它们分别表示单选菜单项和复选菜单项。



示例代码：

- MenuDemo.java

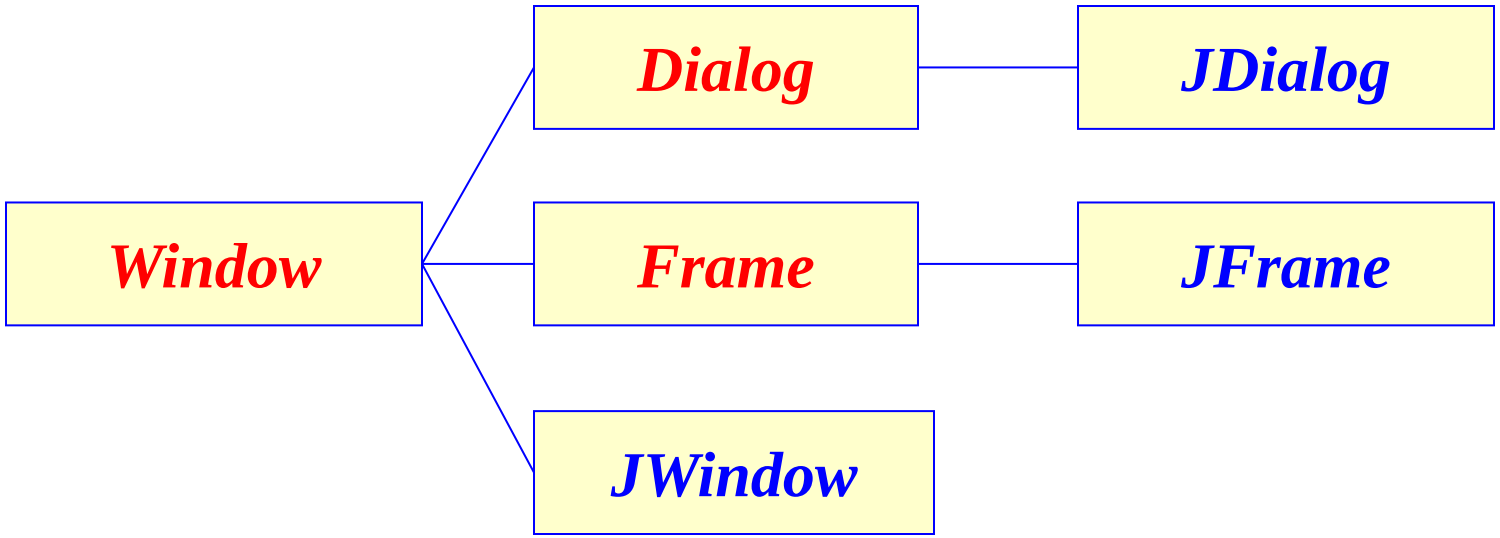
小结(事件处理)

- 确认触发的事件，取得事件类（如ActionEvent）的名字，并删掉其中的“Event”，加入“Listener”；
- 实现上面的接口，针对想要捕获的事件编写方法代码；
- 为事件处理器（监听者接口）创建一个对象，让自己的组件和方法完成对它的注册。

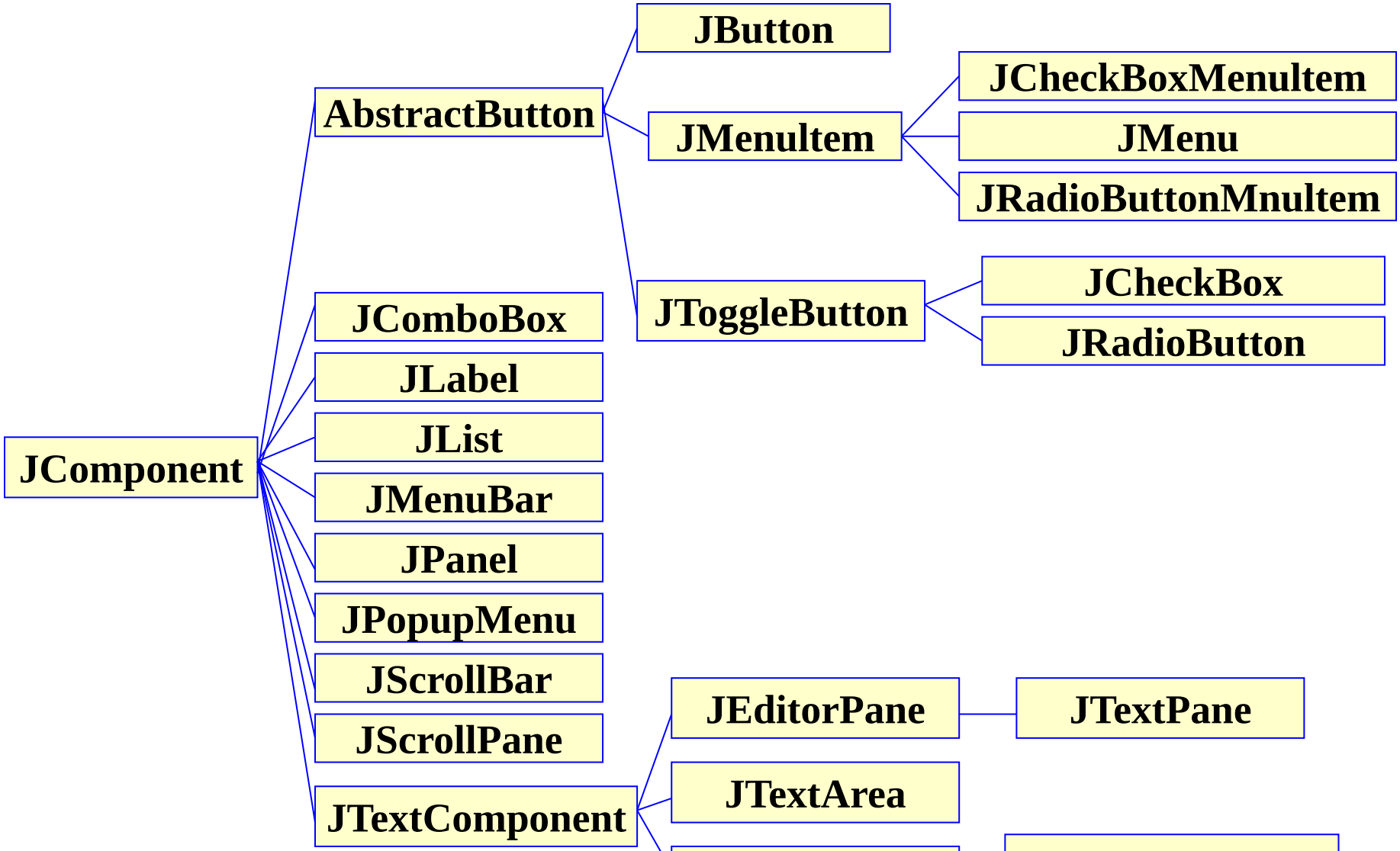
小结：创建容器与组件基本步骤

- 创建顶层容器（常用的为窗口对象）
- 创建内容面板，设置其背景颜色，设置其布局管理器
- 创建普通面板，设置其背景颜色，设置其位置、大小，设置其布局管理器
- 创建组件，设置其背景颜色，设置其位置、大小、字体等
- 将面板添加到窗口，将组件添加到指定面板
- 创建事件监听器，实现事件接口方法
- 给事件源注册监听器

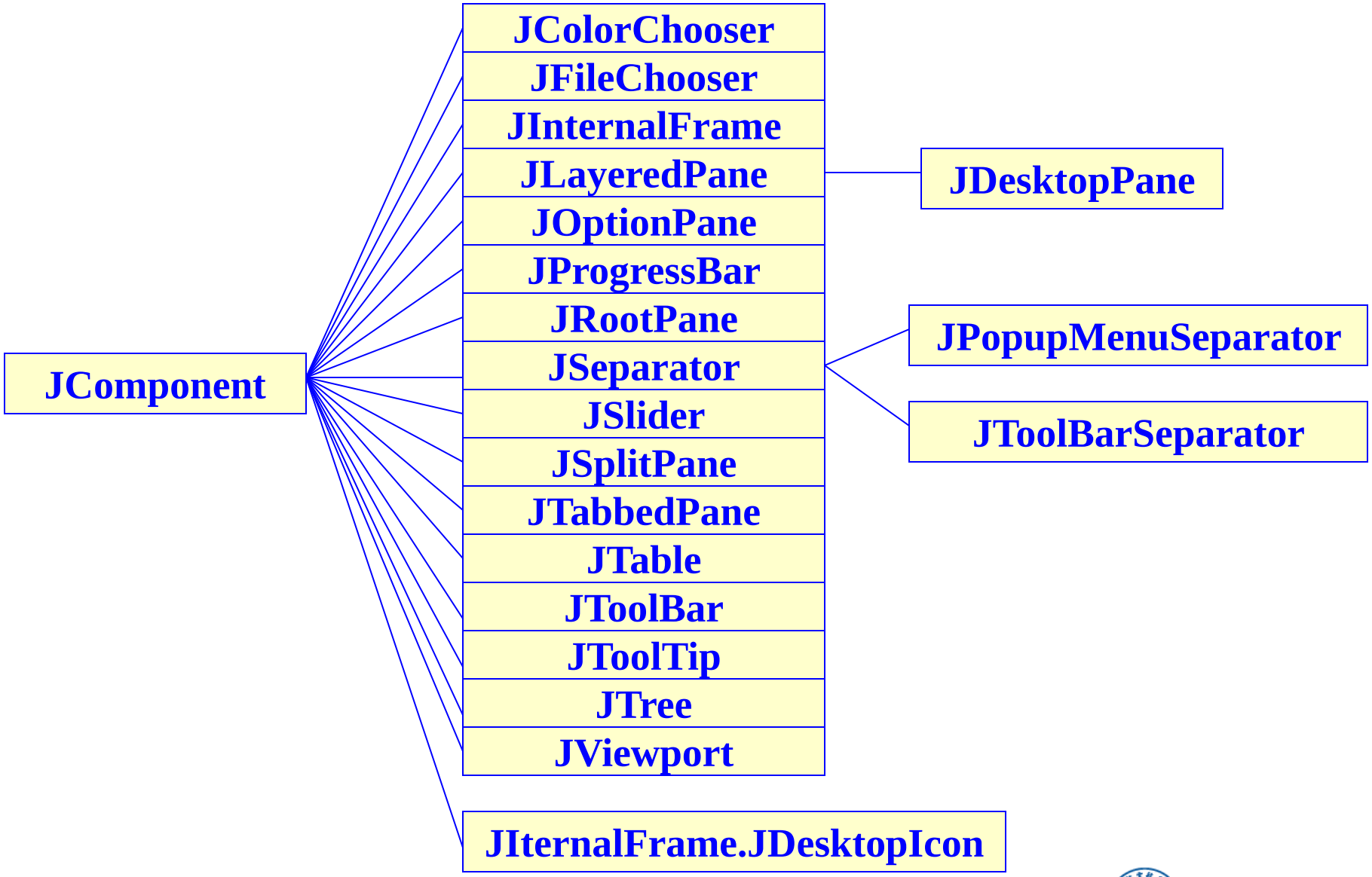
小结：JFrame and Windows



小结：Component Hierarchy—AWT Similar



小结：ComponentHierarchy—New and Expanded Components



小型计算器：

思考题

- 分析计算器的界面元素（控件）
- 分析计算器的整体界面布局
- 分析计算器的菜单层次结构

- *当本次课结束的时候，希望你能**完成计算器整个界面的实现***

谢谢！