第6章 GUI

涉及到课本章节:

- 第6章 图形用户界面

- 6.1 JFC
- 6.2 AWT与Swing
- 6.3 Java 创建GUI步骤
- 6.4 AWT创建GUI
- 6.5 Swing创建GUI

- 6.1 JFC
- 6.2 Swing与AWT
- 6.3 Java 创建GUI步骤
- 6.4 AWT创建GUI
- 6.5 Swing创建GUI

6.1 JFC

- 1. JDK、JFC、AWT、Swing
- 2. 介绍四者在JavaSE Platform的位置
- 3. 四者之间的关系

1. JDK、JFC、AWT、Swing

- JDK=JRE+Java Tools+Java Language

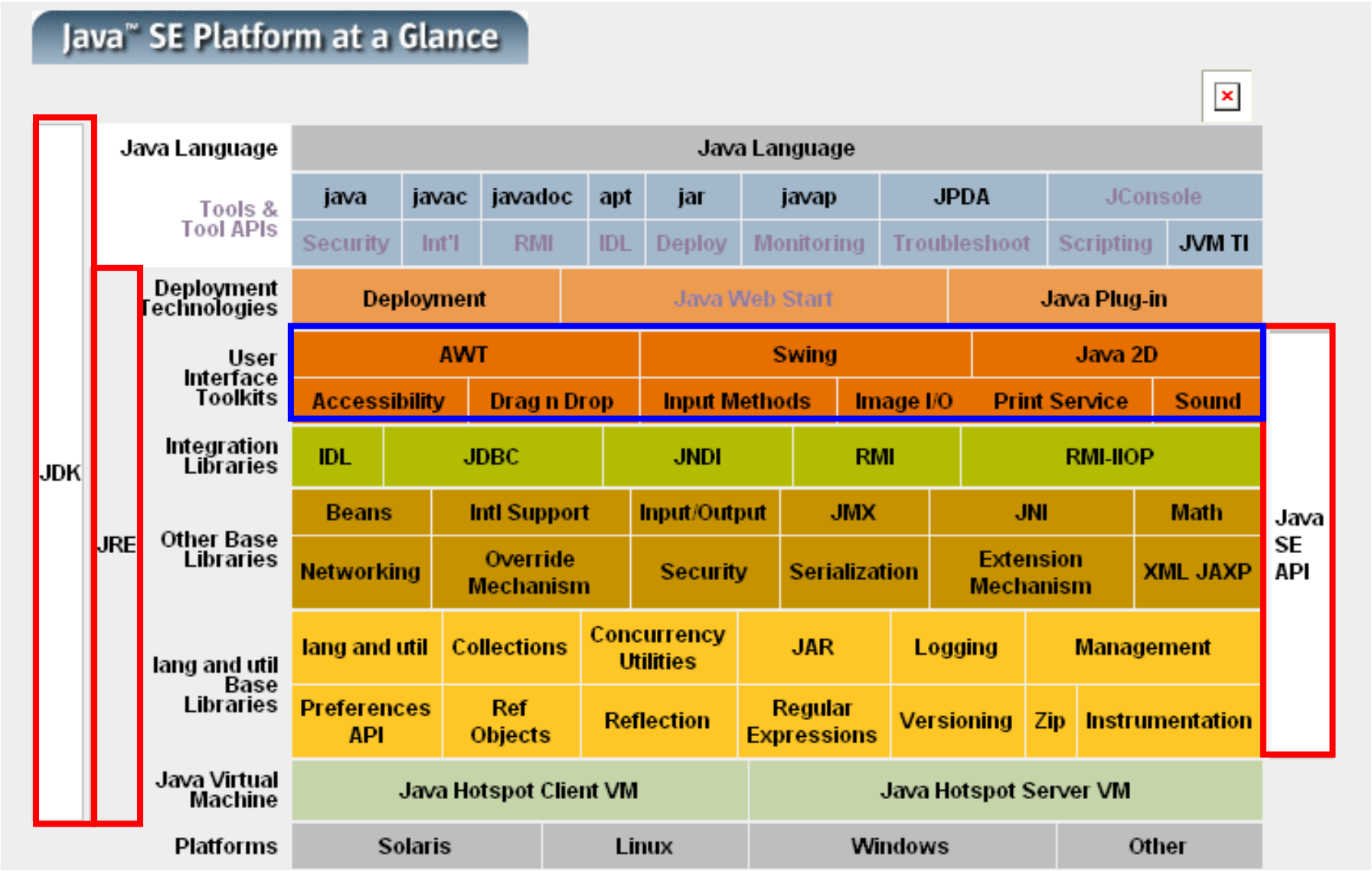
- JFC（Java Foundation Class）

  包括五类主要的API：AWT,Swing,Accessibility，java 2D,Drag n Drop

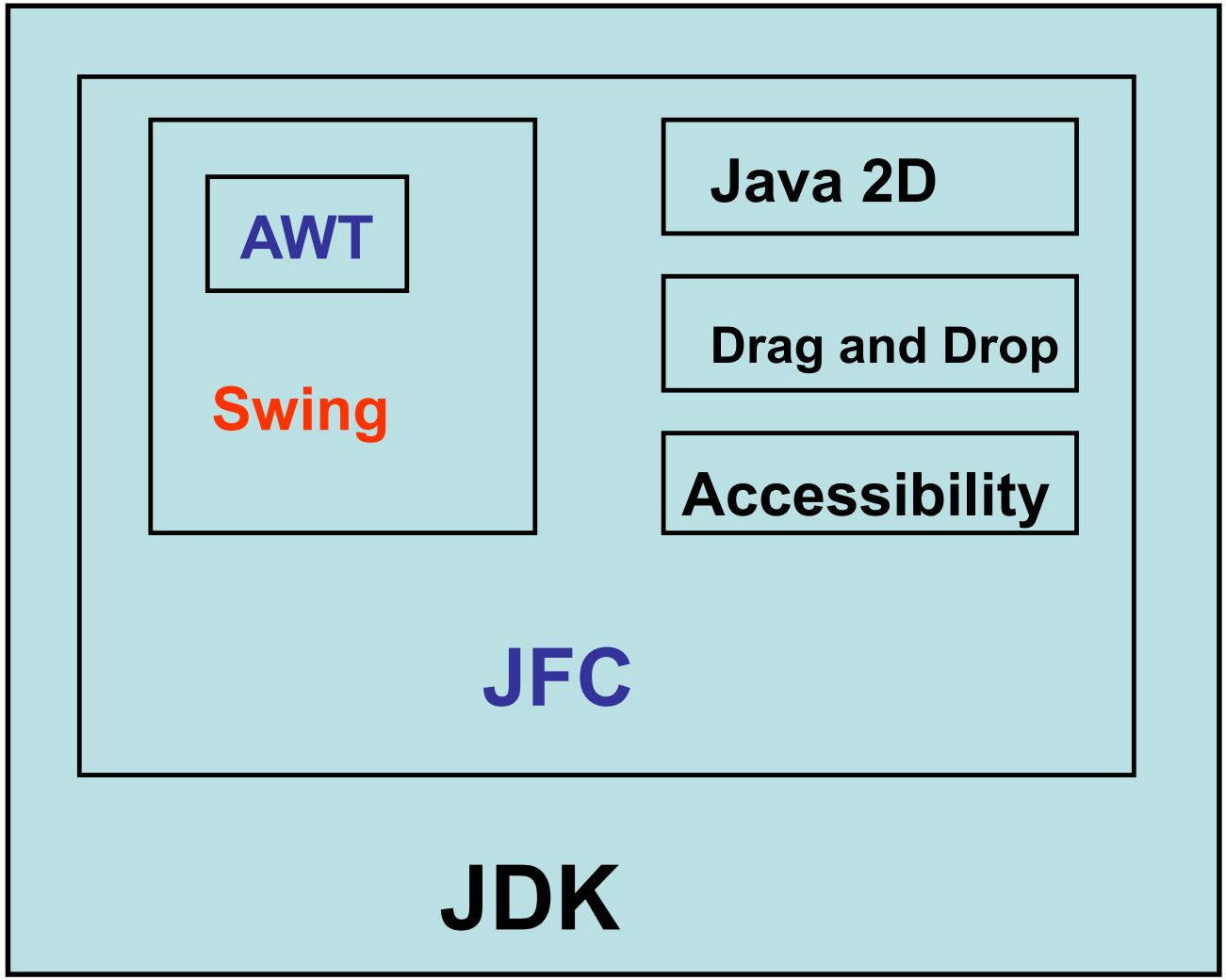
  AWT(Abstract Window Toolkit)

  Swing

2:四者在Java SE中的位置



3.四者之间的关系：



- 6.1 JFC
- **6.2 AWT与Swing**
- 6.3 Java 创建GUI步骤
- 6.4 AWT创建GUI
- 6.5 Swing创建GUI

6.2 AWT与Swing

- 1. AWT (Abstract Window Toolkit)
- 2. Swing
- 3. 初识Swing编程

1 AWT (Abstract Window Toolkit)

- **AWT—是JFC的基石，是JDK1.0的核心库之一。**

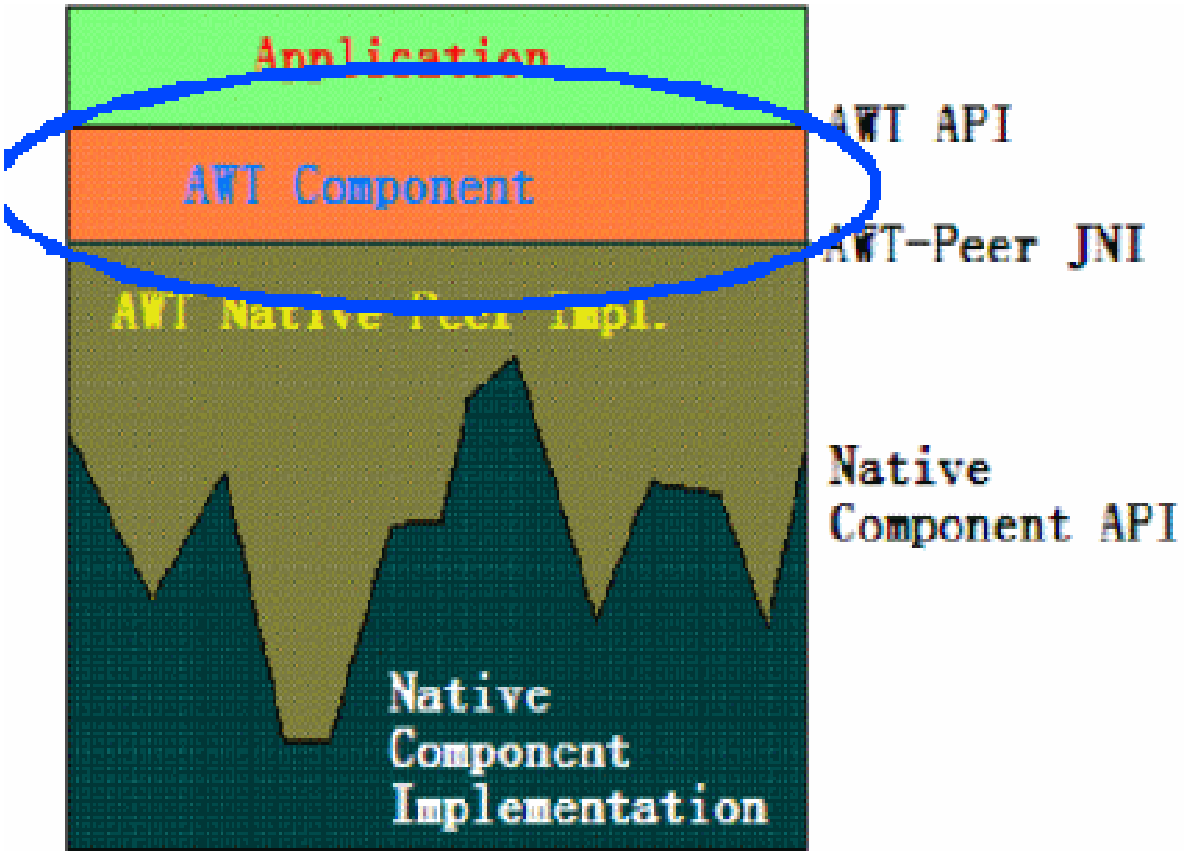
  ----相关的类在java.awt包中。

  ----java.awt包括AWT组件类、组件布局类、组件事件类和事件监听器类等。

  ----AWT组件运行时需要一个本地OS的对等组件为之服务。所以AWT组件也称为重量组件

  -----AWT提供的组件数目和功能有限

关于重量组件：


Application
AWT Component
AWT Native Peer Impl.
AWT API
AWT-Peer JNI
Native
Component API
Native
Component
Implementation

2.Swing

✓Swing

----是AWT的扩展

-----是JDK1.2后JFC的核心库，完全用java编写

---- 相关的类放在javax.swing包中

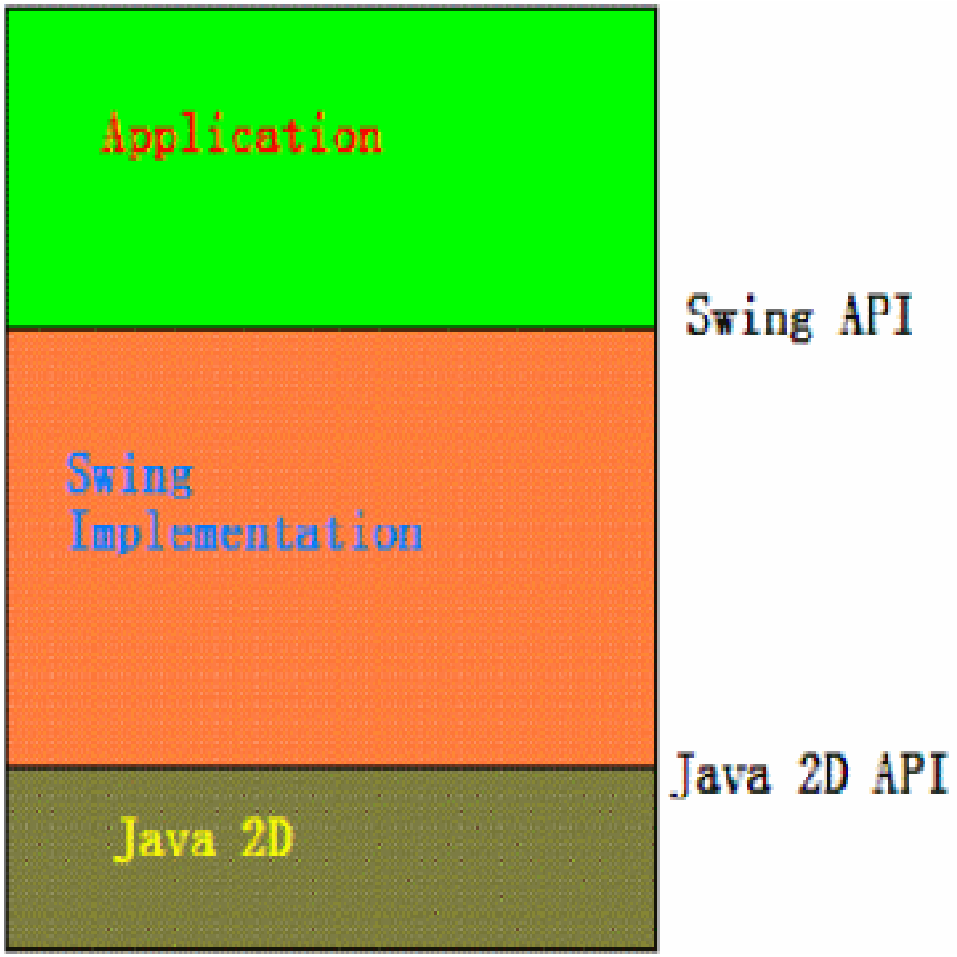
----- 提供了250多个类，40多个组件，相当于AWT的4倍

-----组件都是轻量组件

-----提供可插入式外观（Pluggable Look And Feel）功能，UI组件能动态改变外观，使用当前平台OS风格来显示组件

-----新增加了表格、树、定制对话框等组件

关于轻量组件(light-weight)



3. 初识Swing创建GUI程序

---参看 JDK Installation\demo\jfc\Swingset2示例

注意：

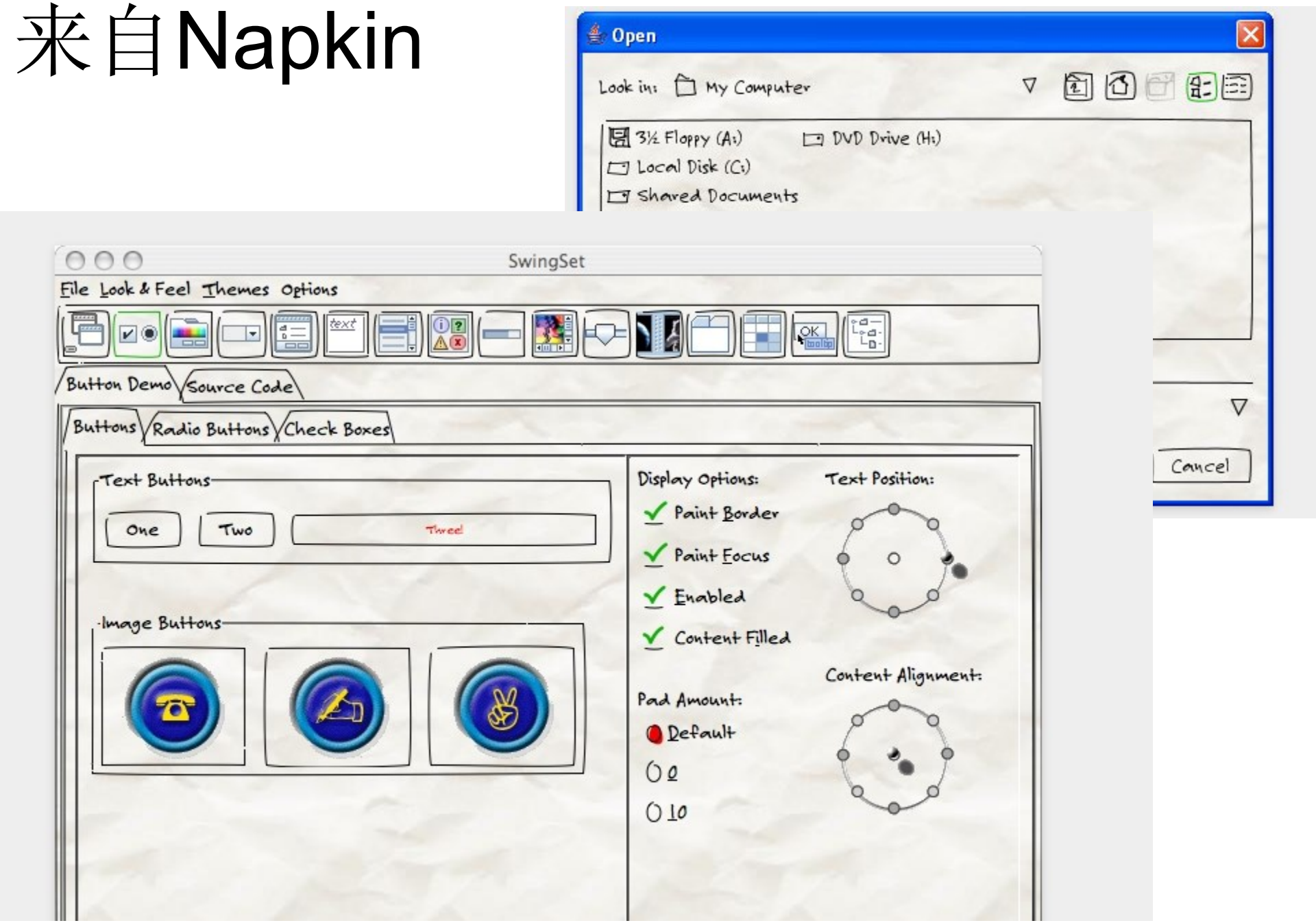
swing的PLAF的特性.

GUI组件在针对不同OS设置时的不同外观.

推荐PLAF链接：

http://napkinlaf.sourceforge.net/

来自Napkin


Open
Look in: My Computer
3½ Floppy (A:)
DVD Drive (H:)
Local Disk (C:)
Shared Documents
Cancel
SwingSet
File Look & Feel Themes Options
Button Demo
Source Code
Buttons
Radio Buttons
Check Boxes
Text Buttons
One
Two
Three!
Image Buttons
Display Options:
Paint Border
Paint Focus
Enabled
Content Filled
Pad Amount:
Default
0
10
Text Position:
Content Alignment:

- 6.1 JFC
- 6.2 Swing与AWT
- 6.3 Java 创建GUI步骤
- 6.4 AWT创建GUI
- 6.5 Swing创建GUI

6.3. JFC创建GUI的步骤：

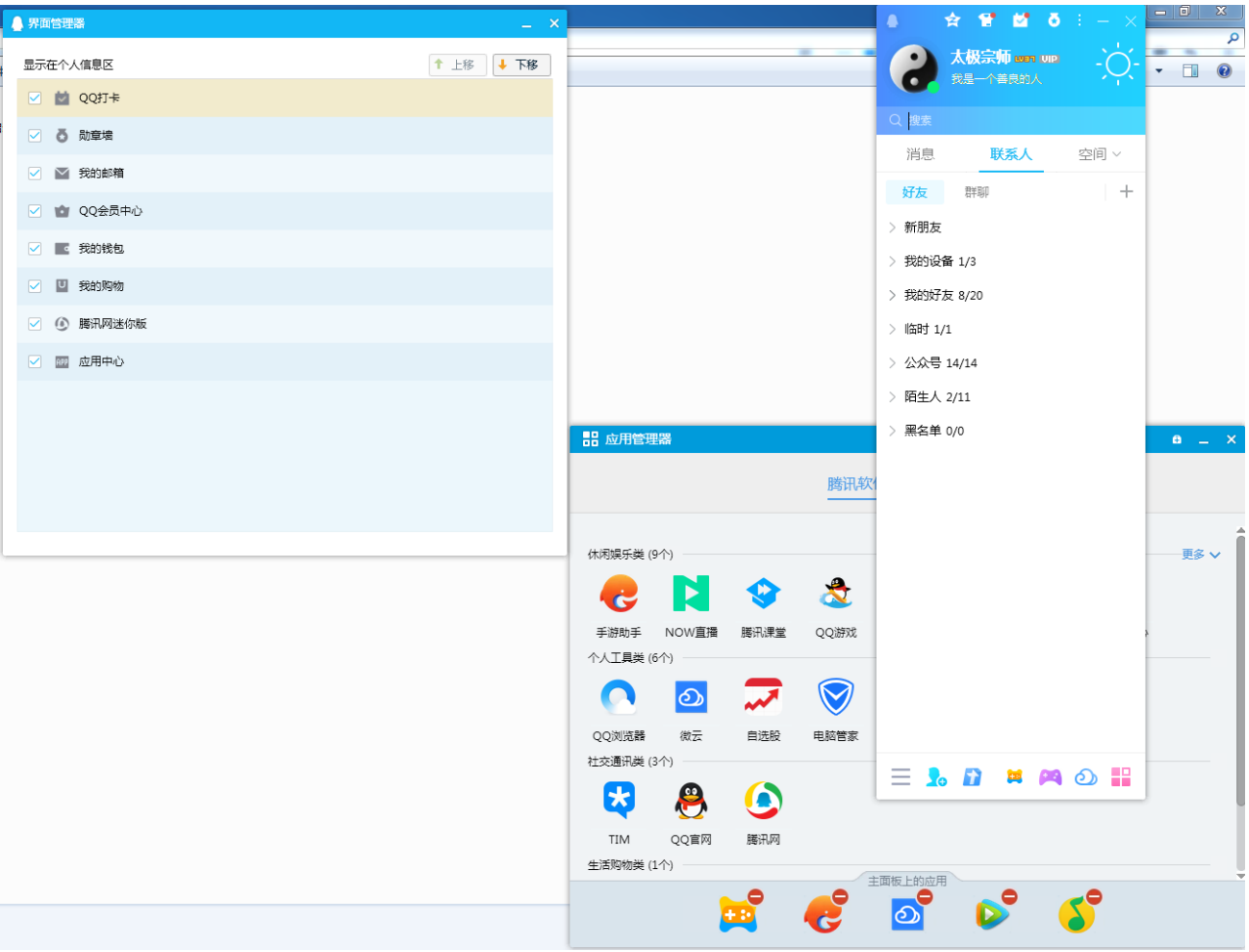
➢创建GUI外观

- (1)选择合适的组件
- (2)选择合适的容器
- (3)选择合适的布局管理器

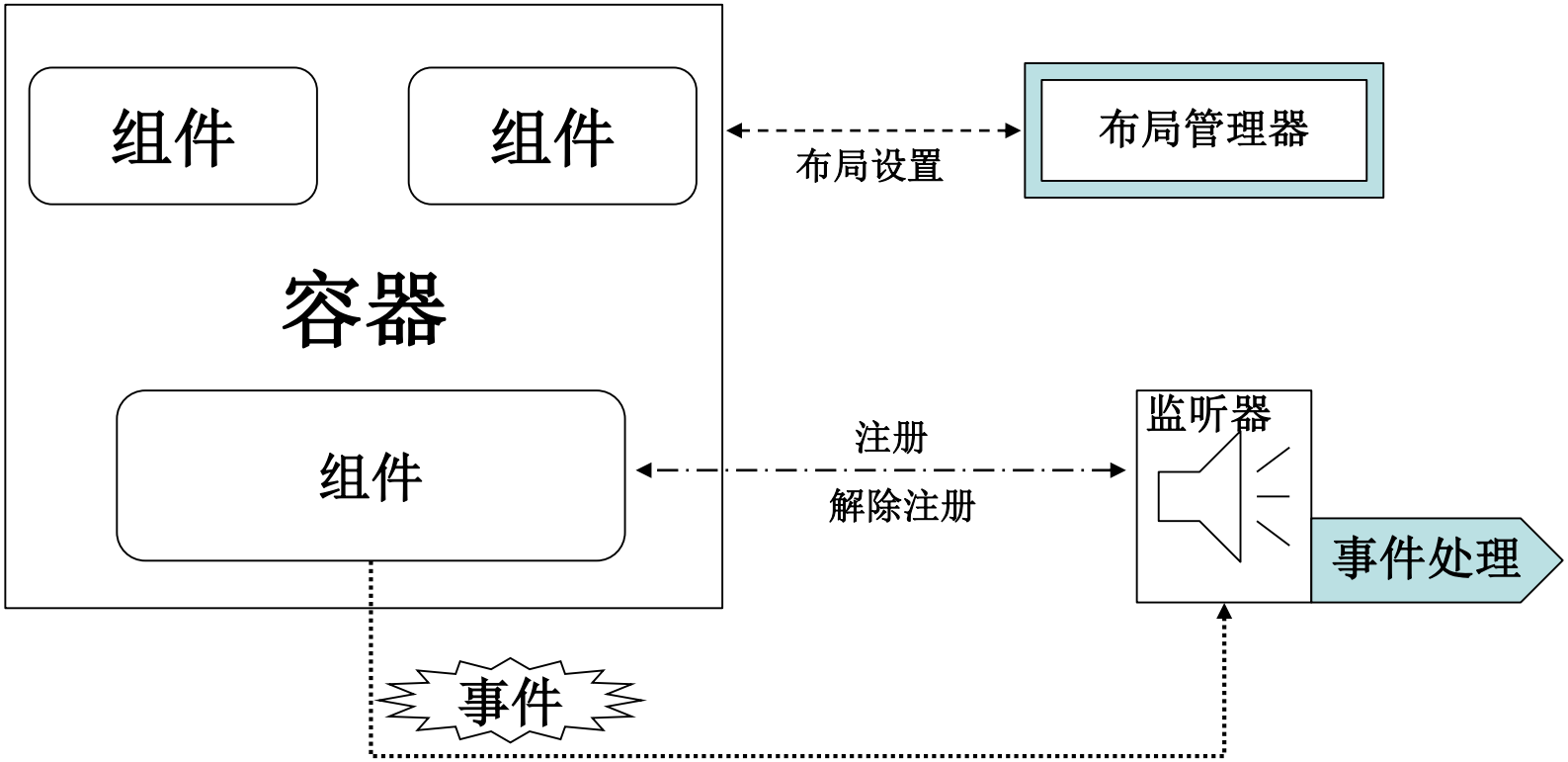
➢创建GUI的感觉

- (1)事件研究
- (2)使用监听器

QQ的GUI外观和GUI感觉



各部分关系



Java GUI中的组件,容器,布局管理器,事件和监听器的关系

总结：---1 创建GUI外观 (java.awt/javax.swing)

- 组件(Component):

  --GUI基本组成部分，例如按钮，滚动条等

  --不能单独显示，必须放到一个容器中

- 容器(Container):

  --是容纳其他组件的组件

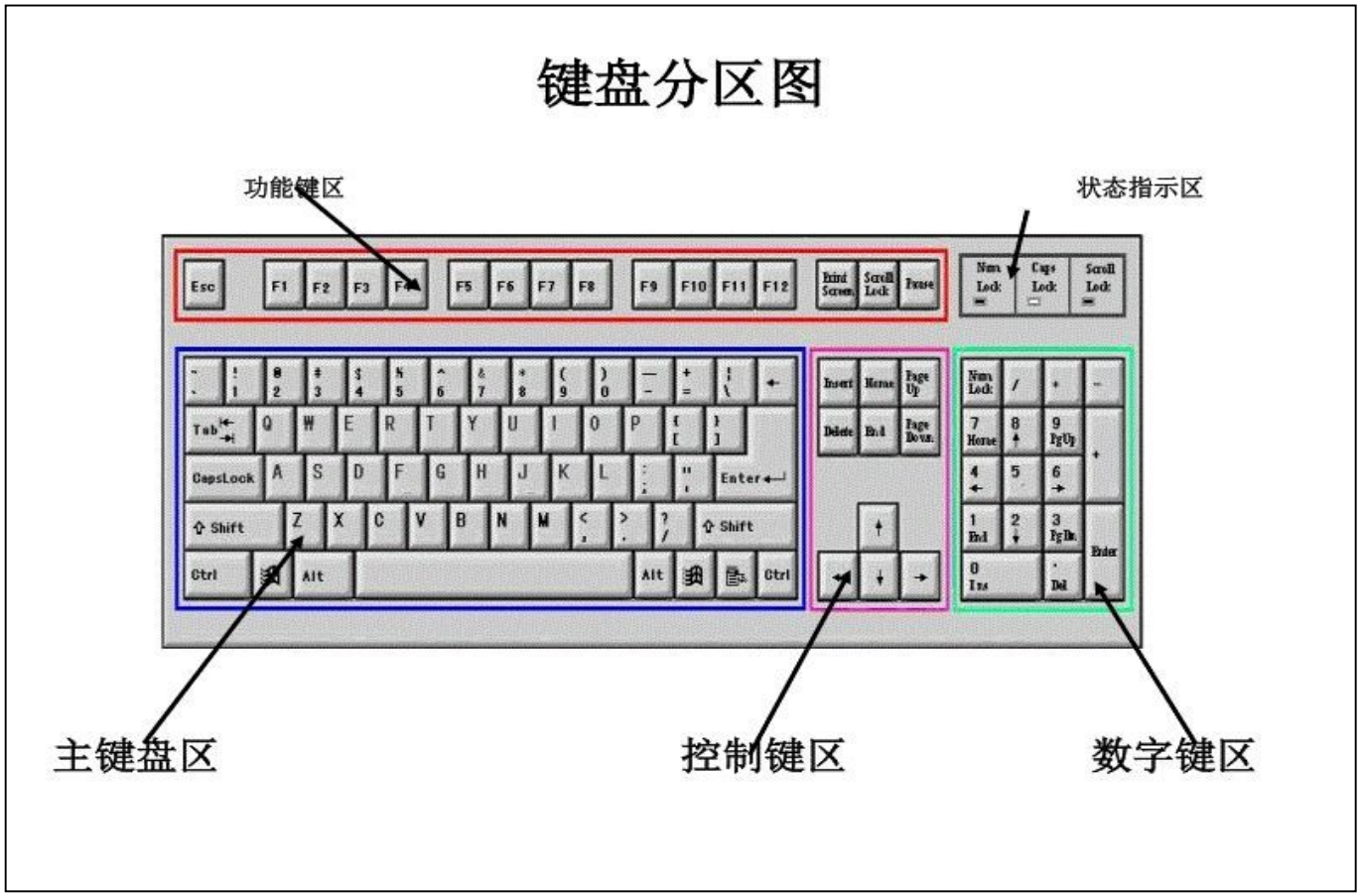
- 布局管理器(LayoutManager)

  --容器的布局

  --排列容器的组件，包括大小和位置

创建GUI外观——类比

- 零件
- 分组
- 排布



总结：---2 创建GUI感觉

(java.awt.event/javax.swing.event)

- 事件(Event)

--一个事件类对象，用户在相应组件上进行操作时会触发相应组件的相应事件

- 事件源(Event Source):

--触发事件的组件 例如按钮，文本框等

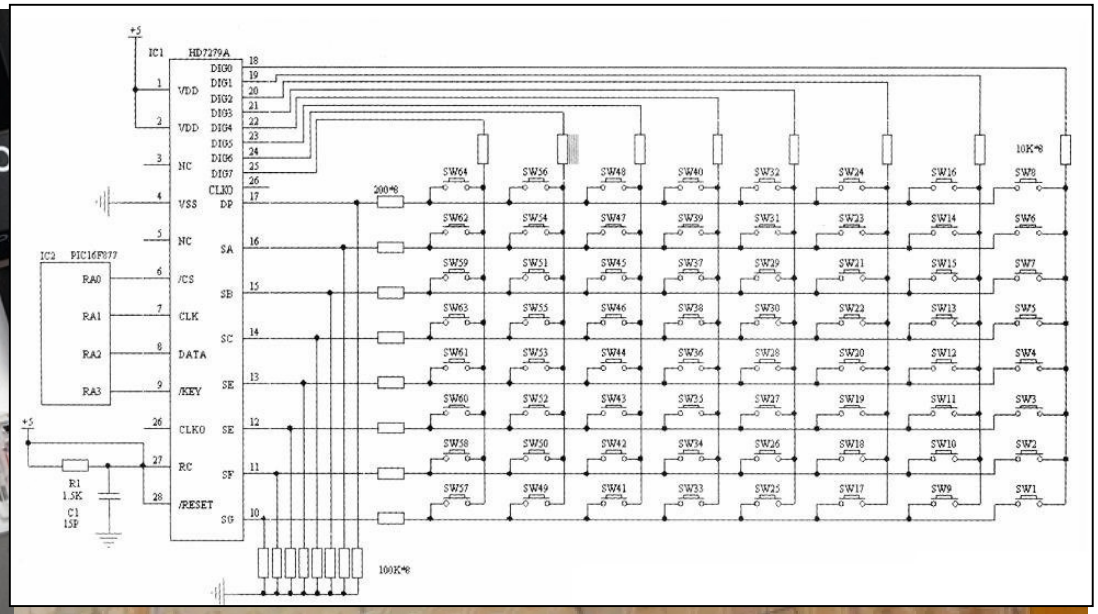
- 事件处理方法(Event Handler):

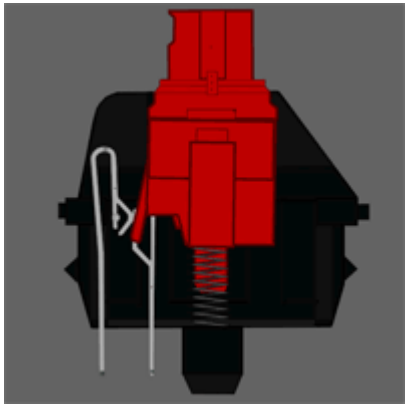
--能够接收，处理事件类对象，实现与用户交互功能的方法

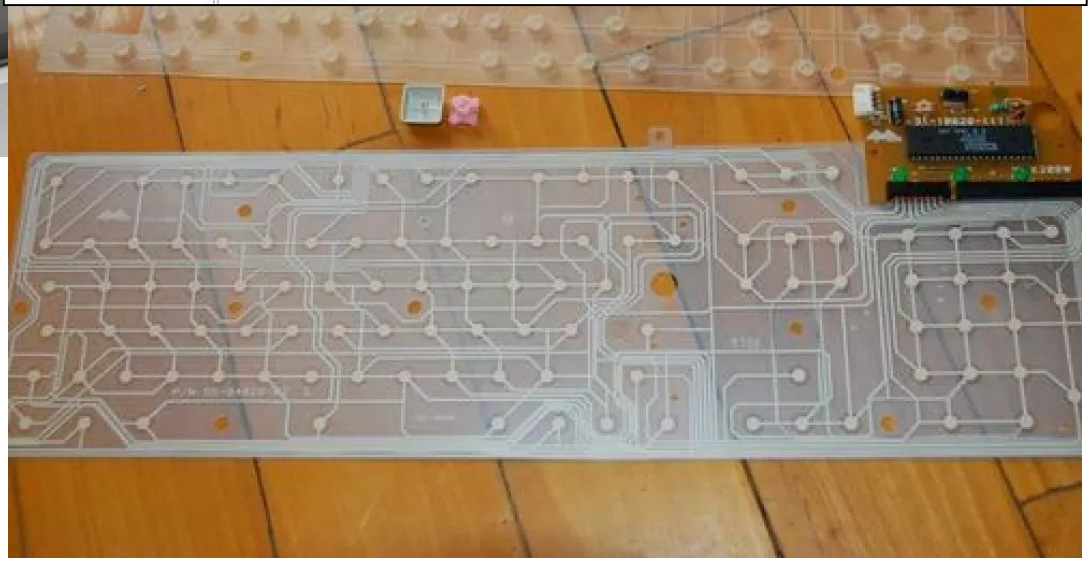
- 事件监听器(Event Listener)

--调用事件处理方法的对象

创建GUI感觉——类比


IC1
HD7279A
IC2
PIC16F877
RA0
RA1
RA2
RA3
/CS
CLK
DATA
/KEY
CLKO
RC
/RESET
R1
1.5K
C1
15P
200*8
100K*8
10K*8



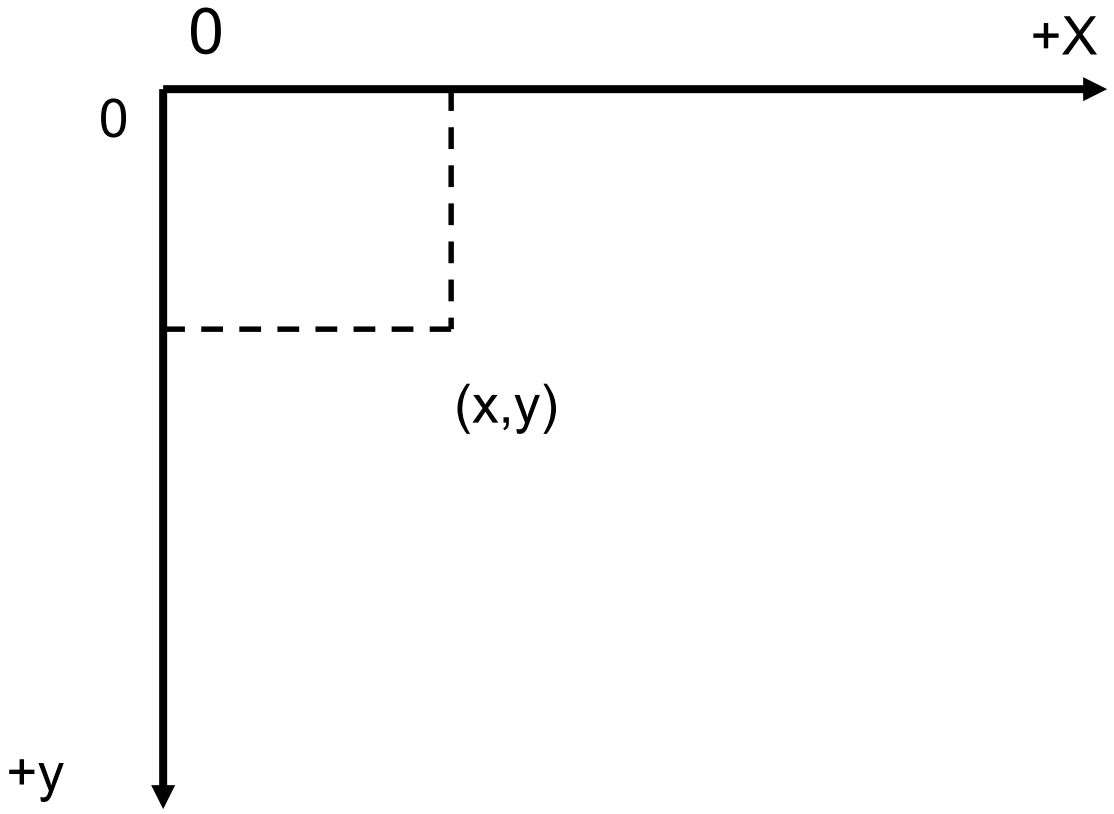


- 6.1 JFC
- 6.2 Swing与AWT
- 6.3 Java 创建GUI步骤
- 6.4 AWT创建GUI
- 6.5 Swing创建GUI

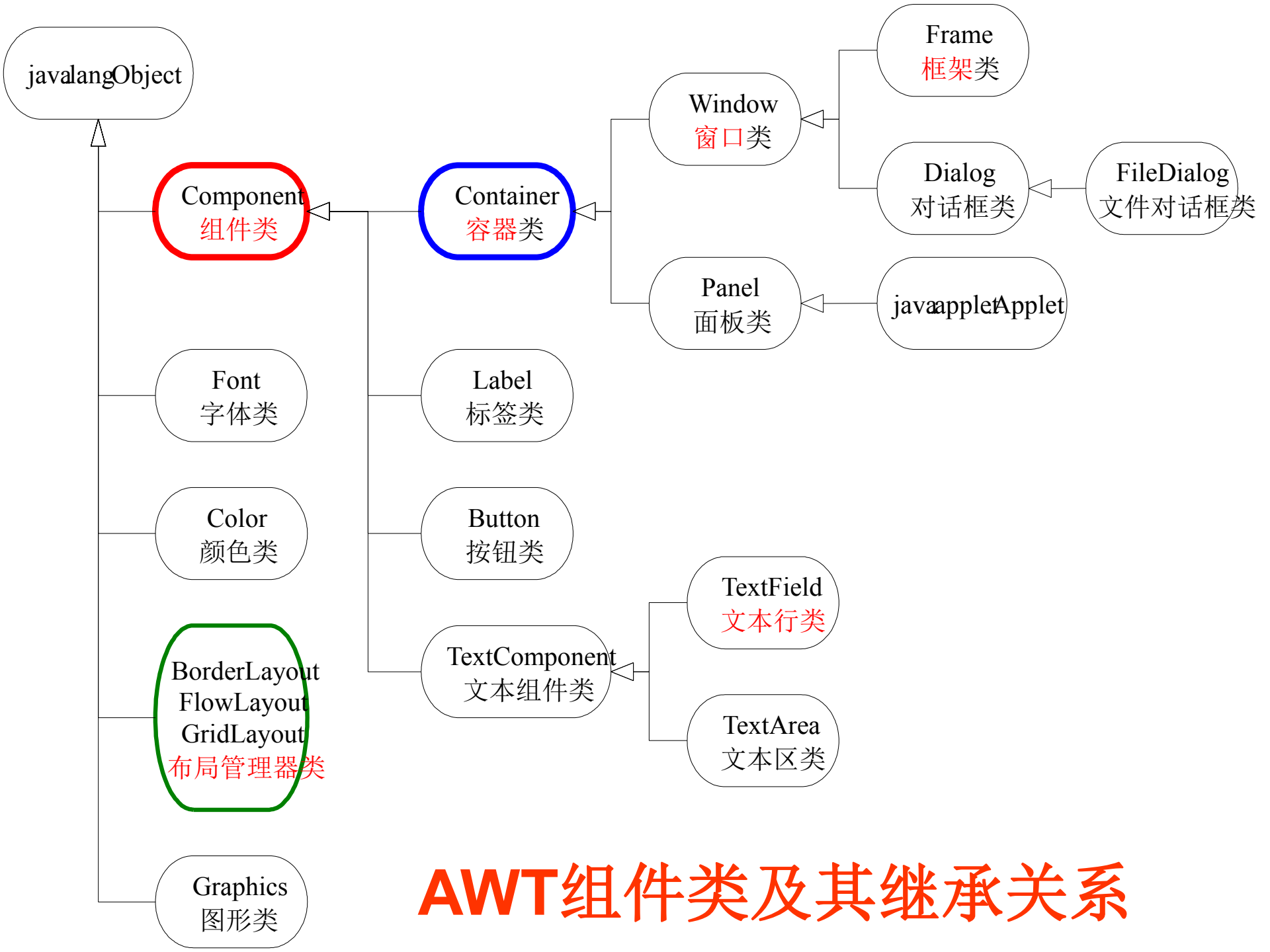
1. java.awt 包

包名	内容
java.awt	构建图形界面，图像和图形等
java.awt.color	使用关于颜色的类
java.awt.dnd	用于拖放操作的类和接口
java.awt.event	用于响应不同事件的类和接口
java.awt.font	用于高级字体操作的类和接口
java.awt.geom	用于绘制**java2D**的类和接口
java.awt.image	创建图像,闪烁图像和使用颜色模型的类和接口
java.awt.print	用于打印操作的类和接口

2. AWT画图基础

- 

AWT的坐标定位系统



AWT组件类及其继承关系

解释：

- **Component：**

  java.awt.Component 是所有组件的抽象父类

- **Container：**

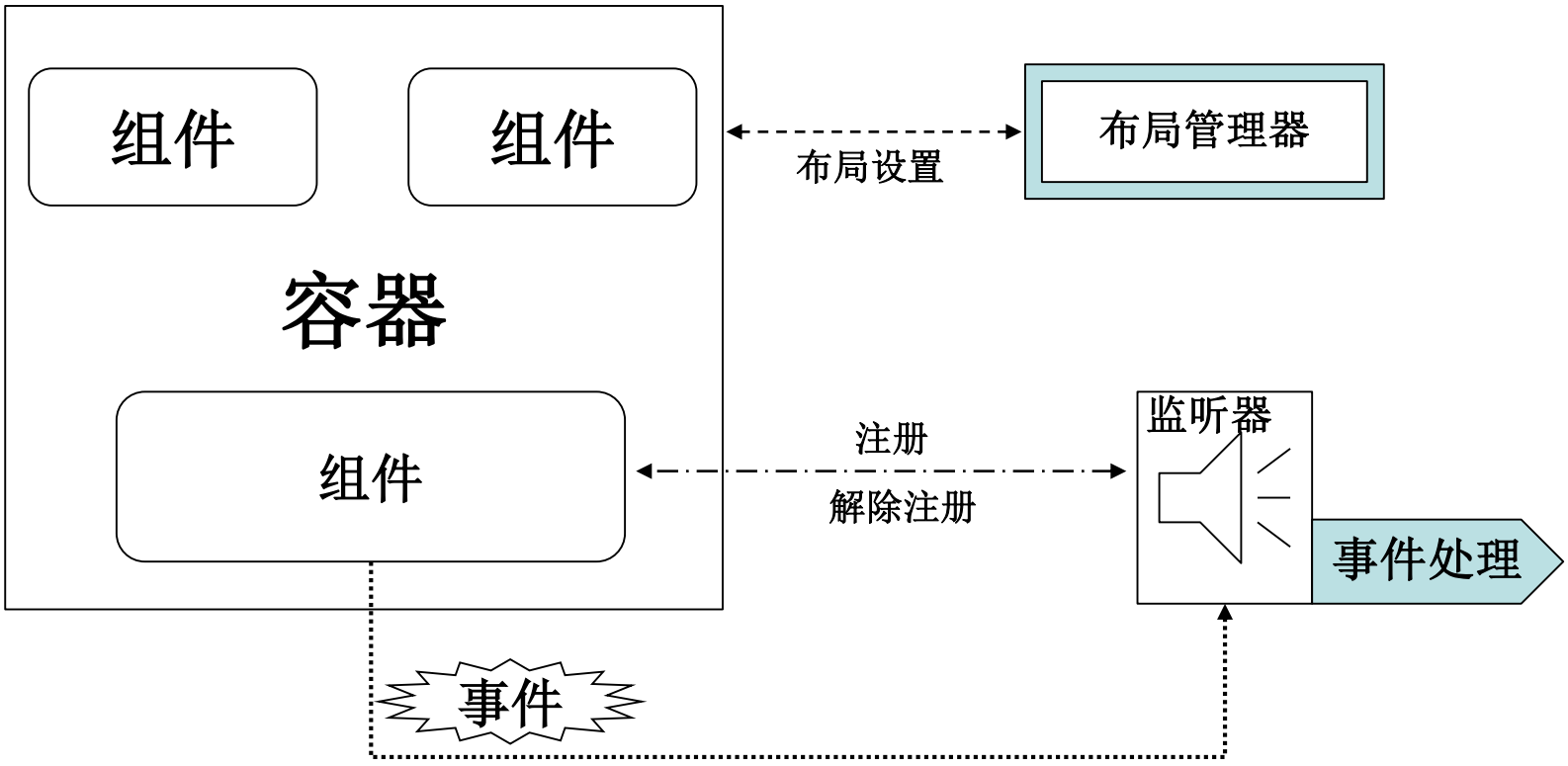
  Window是可以自由停泊的顶级窗口

  Panel 是可以用来容纳其他组件，但是不能单独存在，必须放到其他容器中（**Frame**等）

- **XXXXLayout：**

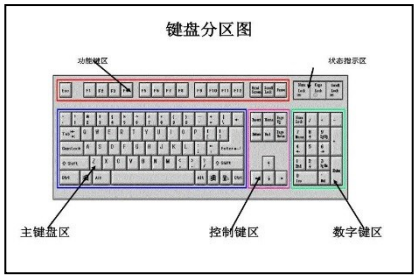
  每一种容器都有自己默认的缺省的布局管理器类

第一阶段:创建GUI外观



- (1)选择合适的AWT组件
- (2)选择合适的AWT容器
- (3)选择合适的AWT布局管理器

- 零件
- 分组
- 排布

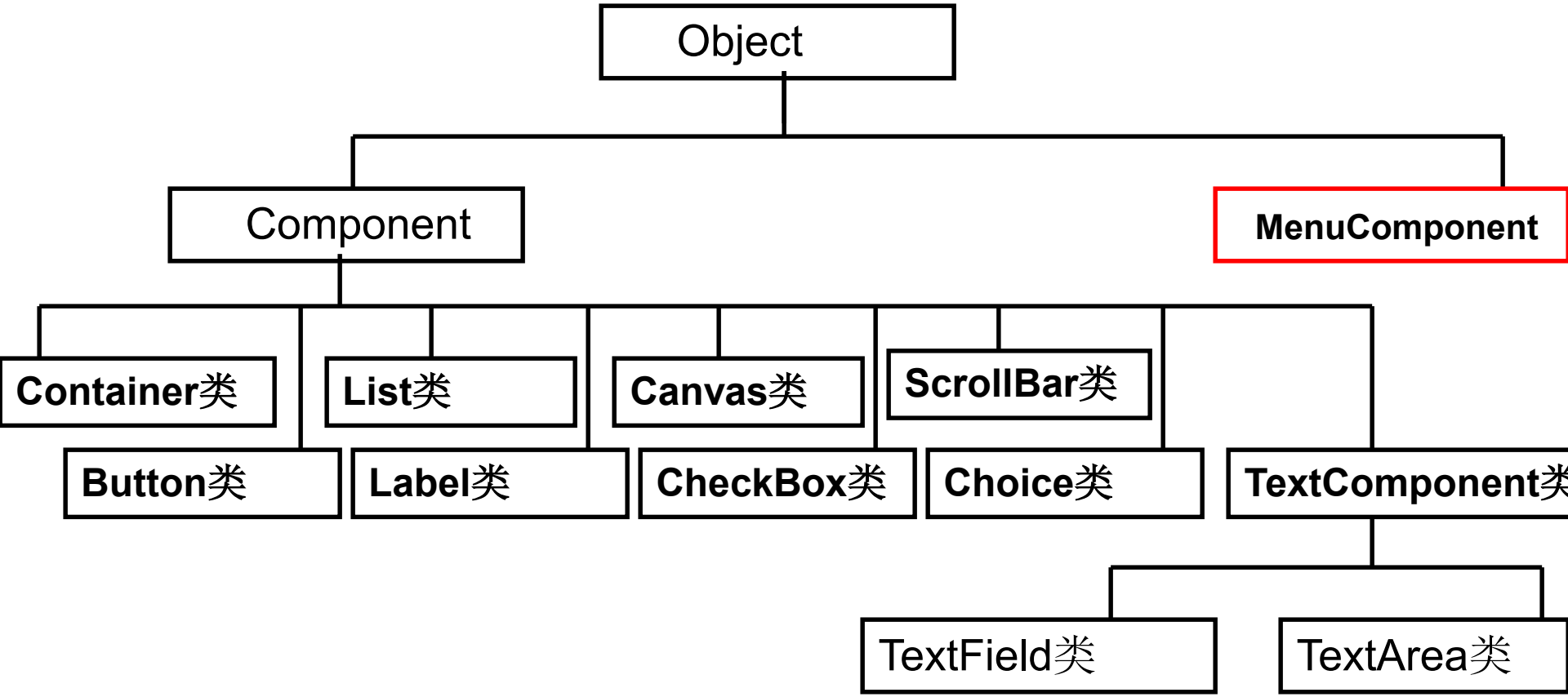


(1)选择合适的AWT组件:

- 组件是GUI的基本成分和核心元素.
- 组件在Java中是以一个类来表示的.
- AWT组件类的父类是Component抽象类

```
public abstract class Component extends Object
      implements ImageObserver, MenuContainer, Serializable
{
   public void setLocation(int x, int y)                    //设置组件位置
   public void setSize(int width, int height)
                                                            //设置组件的宽度和高度
   public void setVisible(boolean b)                        //设置组件是否显示

   ........
}
```

A: 组件使用


Object
Component
MenuComponent
Container类
List类
Canvas类
ScrollBar类
Button类
Label类
CheckBox类
Choice类
TextComponent类
TextField类
TextArea类

组件方法：

getBackground()-----返回一个**Color**对象，它包含组件的背景颜色

getCursor()-----返回一个**Cursor**对象，标识组件的光标

getFont()-------返回一个**Font**对象，它包含组件的字体,(如果组件显示文本，则 该字体用来显示文本)

getForeground()---返回组件的前景色

getGraphics()---返回一个**Graphics**子类对象，代表组件的图像文本

setBackground(Color c)---设置组件的背景色

setForeground(Color c)---设置组件的前景色

setCursor(Cursor c)------设置组件的光标

getSize()-----返回一个**Dimension**对象，标识组件的宽度和高度

setSize(Dimension d)---设置组件的高度和宽度

例：**Button b=new Button(“Hand cursor”);**

b.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));

1).非菜单组件：

- **Button：**

➢ 构造函数：

- **Button();**
- **Button(String label);**

➢ 方法使用举例：

```
Button b=new Button();
Button b1=new Button("OK");
b.setLabel("Cancel");
System.out.println(b1.getLabel());

 //b.setPreferredSize(new Dimension(200, 50)); //
 //b.setBounds(10, 10, 100, 50);
////b.setSize(new Dimension(50, 20)); // f.setLayout(null);
////b.setLocation(20, 20);
```

组件尺寸设置

- 1、setPreferredSize需要在使用布局管理器的时候使用
- 2、setSize,setLocation,setBounds方法需要在不使用布局管理器的时候使用
- 3、setSize（包括setLocation）在绝对布局中才能生效。
- 4、setPreferredSize一般先获取容器（如Panel）的空间大小，控件的大小即为容器的大小。
- 5、setPreferredSize设置此组件的首选大小

- **Label：**

➢ 构造函数：

- **Label();**
- **Label(String text);**
- **Label(String text,String alignment);**

---alignment(LEFT,RIGHT,CENTER)

➢ 方法使用：

```
Label l1=new Label();
Label l2=new Label("center aligned",Label.CENTER);
l1.setText("Right aligned");
l1.setAlignment(Label.RIGHT);
System.out.println(l1.getAlignment());
System.out.println(l2.getText());
```

Checkbox和CheckboxGroup

Checkbox

- 构造函数：
- 见JDK(5个)
- 方法使用：

```
Checkbox cb1=new Checkbox("Married");
Checkbox cb2=new Checkbox();
System.out.println(cb1.getState());
cb2.setLabel("Single");
cb2.setState(true);
```

CheckboxGroup

- 构造函数：

```
CheckboxGroup()
```

- 方法使用：

```
CheckboxGroup cbg=new CheckboxGroup();
Checkbox cb1=new Checkbox(
    "Left Justify",true,cbg);
Checkbox cb2=new Checkbox(
    "Center Justify",false,cbg);
Checkbox cb3=new Checkbox(
    "Right Justify",false,cbg);
.....
if(cbg.getSelectedCheckbox()==cb1)
  cbg.setSelectedCheckbox(cb2);
```

举例：

```
import java.awt.*;
class CheckDemo extends Frame
{  CheckDemo(String title)
   {super(title);
   Checkbox cb1=new Checkbox("Married");
   Checkbox cb2=new Checkbox();
   cb2.setLabel("Single");
   cb2.setState(true);
   Panel p1=new Panel();
   p1.add(cb1);
   p1.add(cb2);
   CheckboxGroup cbg=new    CheckboxGroup();
   Checkbox c1=new Checkbox( "Left Justify",true,cbg);
  Checkbox c2=new Checkbox("Center Justify",false,cbg);
  Checkbox c3=new Checkbox("Right Justify",false,cbg);
  Panel p2=new Panel();
  p2.add(c1);
  p2.add(c2);
  p2.add(c3);
  add(p1,BorderLayout.CENTER);
  add(p2,BorderLayout.SOUTH);
  setSize(400,400);
  setVisible(true);

}
```

```
public static void main(String[] args)
{ CheckDemo chd=new
CheckDemo("check");
}
}
```

- **Choice类**

➢ 构造函数

- **Choice()**

➢ 方法使用

```
Choice countries=new Choice();
countries.add("Albania");
countries.add("Algeria");
countries.add("Bahrain");
countries.add("China");
System.out.println(countries.getItemCount());
countries.insert("Canada",2);   --直接放在index参数所标识的位置之前插入
```

举例：

```
import java.awt.*;
class BCCDemo extends Frame
{
  BCCDemo(String title)
  {super(title);
   Label ln=new Label("姓名");
   TextField tx=new TextField();
   Label ls=new Label("性别");
   Choice c=new Choice();
   c.add("男");
   c.add("女");
   Button b1=new Button("确定");
   Button b2=new Button("取消");
   Panel p1=new Panel();
   Panel p2=new Panel();
   p1.add(ln);
   p1.add(tx);
   p1.add(ls);
   p1.add(c);
   p2.add(b1);
   p2.add(b2);
   add(p1,BorderLayout.CENTER);
   add(p2,BorderLayout.SOUTH);
   setSize(400,400);
   setVisible(true);
```

```
 public static void main (String[] args)
{  BCCDemo bc=new BCCDemo(“BCC”);
}
}
```

- **List类**

➢ 构造函数

List();

List(int rows);

List(int rows,boolean multipleMode)

➢ 方法使用

```
List magazines=new List();
magazines.add("bytes");
magazines.add("Dr.Dobbs");
magazines.add("JavaWorld");
System.out.println(magazines.getItemCount());
magazines.setMultipleMode(true);
```

举例：

```
import  java.awt.*;
class ListDemo extends Frame
{  ListDemo(String title)
   {  super(title);

   List ls=new List();
   ls.add("java语言");
   ls.add("C 语言");
   ls.add("操作系统");
   ls.add("编译原理");
   ls.add("系统结构");
   ls.add("数据结构");
   ls.add(" C#");
   ls.add("数值分析");
   ls.add(“网络");
  //记录较多时，会根据大小自动添加滚动条

   ls.setMultipleMode(true);
   add(ls,BorderLayout.CENTER);
   setSize(400,400);
   setVisible(true);
   }
```

```
 public static void main(String[] args)
{   ListDemo lsd=new     ListDemo("List");
}
}
```

- **Scrollbar类**

➢ 构造函数：

- 见JDK(3个)

➢ 使用方法：

```
Scrollbar sb=new Scrollbar(Scrollbar.HORIZONTAL,0,1,0,50);
//sb.setPreferredSize(new Dimension(200, 20));
 System.out.println(sb.getOrientation());
 System.out.println(sb.getValue());
 System.out.println(sb.getVisibleAmount());
 System.out.println(sb.getMinimum());
 System.out.println(sb.getMaximum());
 Scrollbar sb2=new Scrollbar(Scrollbar.VERTICAL);
 //sb2.setPreferredSize(new Dimension(20, 200));
 sb2.setValues(0,1,0,50);
 //sb2.setValues(20,1,0,50);
```

举例：

```
import java.awt.*;
class ScrollDemo extends Frame
{  ScrollDemo(String title)
   {super(title);
     Scrollbar sb=new Scrollbar(Scrollbar.HORIZONTAL,0,1,0,50);
      System.out.println(sb.getOrientation());
      System.out.println(sb.getValue());
      System.out.println(sb.getVisibleAmount());
      System.out.println(sb.getMinimum());
      System.out.println(sb.getMaximum());
      Scrollbar sb2=new Scrollbar(Scrollbar.VERTICAL);
      sb2.setValues(0,1,0,50);
      add(sb,BorderLayout.NORTH);
      add(sb2,BorderLayout.EAST);
      setSize(400,400);
      setVisible(true);
}
```

```
 public static void main(String[] args)
{ ScrollDemo sc=new ScrollDemo("Scroll");
}
}}
```

- **TextComponent类**
- **TextField类**
  - 构造函数
    见**JDK(共4个)**
  - 方法使用

```
TextField t=new TextField("enter the name . ",20);
t.setBackground(Color.cyan);
System.out.println(t.getText());
TextField password=new TextField(20);
password.setEchoChar('*');
```

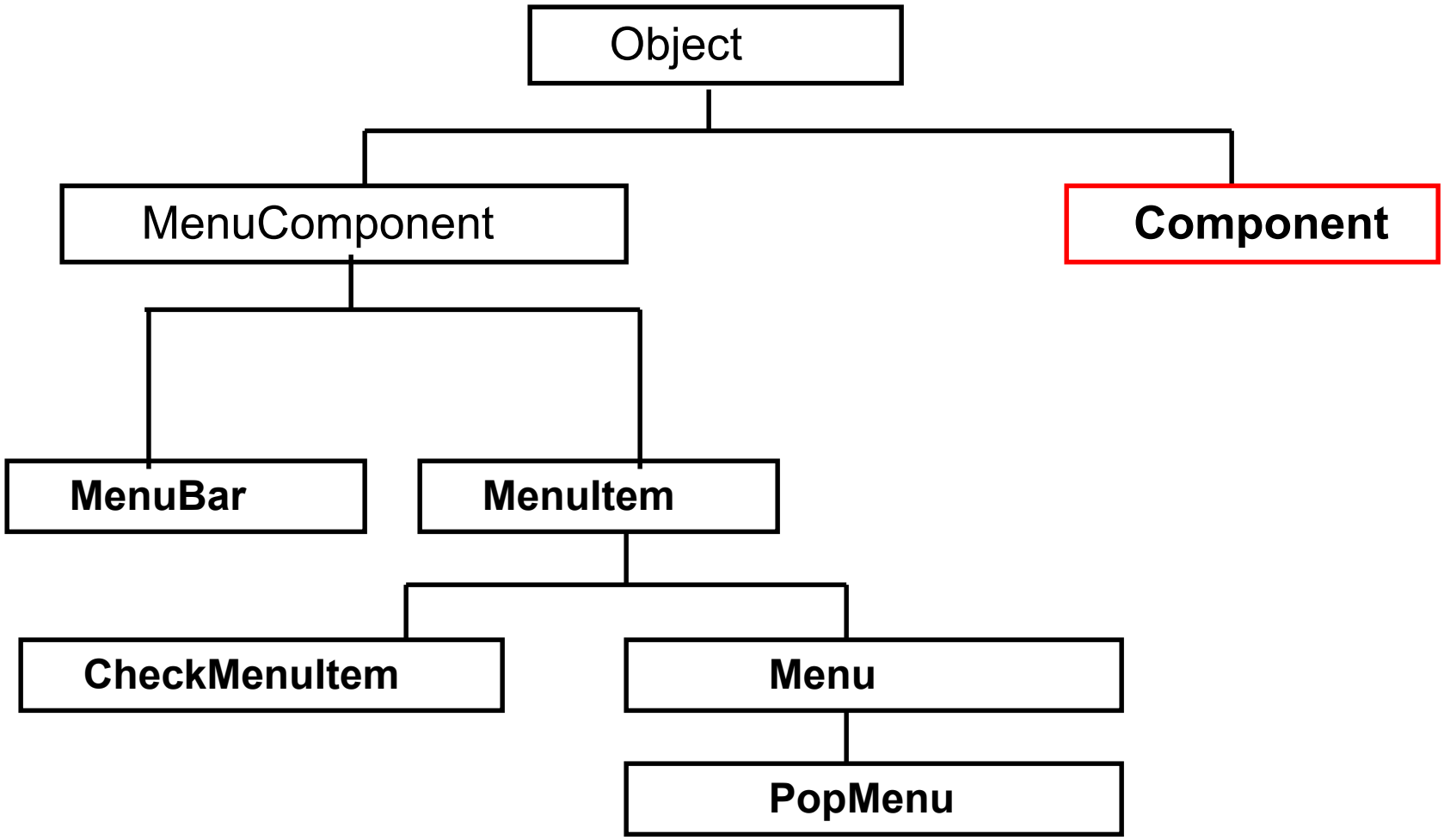
- **TextArea类**
  - 构造函数
    见**JDK(共5个)**
  - 方法使用

```
TextArea ta=new TextArea(5,5);
ta.setText("Initial text");
ta.insert("ized",7);
ta.setSelectionStart(4);
ta.setSelectionEnd(6);
```

举例：

```
import java.awt.*;
class TextDemo extends Frame
{ TextDemo(String title)
  {  super(title);
     TextField t=new TextField("enter the name .",20);
     t.setBackground(Color.cyan);
     System.out.println(t.getText());
     TextField password=new TextField(20);
     password.setEchoChar('*');
     add(t,BorderLayout.NORTH);
     add(password,BorderLayout.CENTER);
     TextArea ta=new TextArea(5,5);
     //TextArea ta=new TextArea(5,20);
     ta.setText("Initial text");
     ta.insert("ized",7);
     ta.setSelectionStart(4);
     ta.setSelectionEnd(6);
    add(ta,BorderLayout.SOUTH);
    setSize(300,300);
    setVisible(true);
public static void main(String[] args)
{ TextDemo txd=new TextDemo("text");
}
}
```

2).菜单组件


Object
MenuComponent
Component
MenuBar
MenuItem
CheckMenuItem
Menu
PopMenu

. MenuBar类

- 构造函数

  MenuBar();

- 方法使用

  add(Menu m);---------add the specified Menu to the menu bar

  Menu getMenu(int i);-----get the specified menu

  getMenuCounts()--------get the numbers of menus on the menu bars

  remove(index i)/remove(Menucomponent m)

  -----remove the menu from the menu bar

  SetHelpMenu(Menu m)----set the specified menu to be this menu bar's help menu

Menu类

- 构造函数

  见JDK(共3个)

- 方法使用

  add(String label)-----添加菜单项(默认在一个菜单的结尾)

  add(MenuItem mi)----添加一个复选框菜单项

  addSeparator()-----在当前位置插入一个分割符

  insert(MenuItem m,index i)/Insert(String label,int index)

  insert(int index);

  remove(….)

注：可以用**setActionCommand()**分配一个字符串命令标识符给一个菜单项，这个标识符唯一的标识这个菜单项。

CheckboxMenuItem

- 代表复选框菜单项。

MenuShortcut

- 代表菜单快捷键
- **Menushortcut(int key)　--KeyEvent类中可以找到key常量**
- **Menushortcut(int key,boolean useShiftModifier)**

菜单举例：

```
import java.awt.*;
import java.awt.event.*;
class TextDemo extends Frame
{ TextDemo(String title)
  {  super(title);
     TextField t=new TextField("enter the   name .",20);
     t.setBackground(Color.cyan);
    TextField password=new TextField(20);
     password.setEchoChar('*');
     add(t,BorderLayout.NORTH);
     add(password,BorderLayout.CENTER);
     TextArea ta=new TextArea(5,5);
     ta.setText("Initial text");
    add(ta,BorderLayout.SOUTH);
```

```
//the following code is about menu
     Menu file=new Menu("File");
     MenuItem mi=new
MenuItem("open..",new
MenuShortcut(KeyEvent.VK_O));
     file.add(mi);
     mi=new MenuItem("Save…",new
MenuShortcut(KeyEvent.VK_S));
    file.add(mi);
    file.addSeparator();
    mi=new MenuItem("Exit");
   file.add(mi);
   MenuBar mb=new MenuBar();
   mb.add(file);
   setMenuBar(mb);
     setSize(300,300);
}
```

```
public static void main(String[] args)
{ TextDemo txd=new TextDemo(“Menu");
}
}
```

PopupMenu类

- 构造函数

  PopupMenu()----empty Name

  PopupMenu(String lable)-----specified name for a new popmenu

- 方法使用

  show(Component origin,int x,int y)----

  在**(x,y)**位置处显示一个弹出式菜单，该位置相对于**origin**

注：**PopupMenu**继承了**Menu**类。所以拥有**Menu**类的所有方法

AWT Applet不支持菜单栏，但是**Applet**和**Application**都支持弹出式菜单

PopupMenu举例：

```
import java.awt.*;
import java.awt.event.*;
class TextDemo extends Frame
{ TextDemo(String title)
  {  super(title);
     TextField t=new TextField("enter the name .",20);
     t.setBackground(Color.cyan);
     TextField password=new TextField(20);
     password.setEchoChar('*');
     add(t,BorderLayout.NORTH);
      add(password,BorderLayout.CENTER);
     TextArea ta=new TextArea(5,5);
     ta.setText("Initial text");
     add(ta,BorderLayout.SOUTH);

     setSize(400,400);
      setVisible(true);
```

```
PopupMenu file=new PopupMenu();
MenuItem mi=new MenuItem("open..");
file.add(mi);
mi=new MenuItem("Save…");
file.add(mi);
file.addSeparator();
mi=new MenuItem("Exit");
file.add(mi);
add(file);
 file.show(t,20,20); //演示临时添加
}
 public static void main(String[] args)
 { TextDemo txd=new
TextDemo("Menu");}}
```

注：

PopupMenu.show(,,,)的origin参数中的组件和组件父类必须显示后才能调用。

PopupMenu必须使用事件和监听器。所以具体的实现稍后介绍

自己试验一下把显示语句放到后面看看

第一阶段:

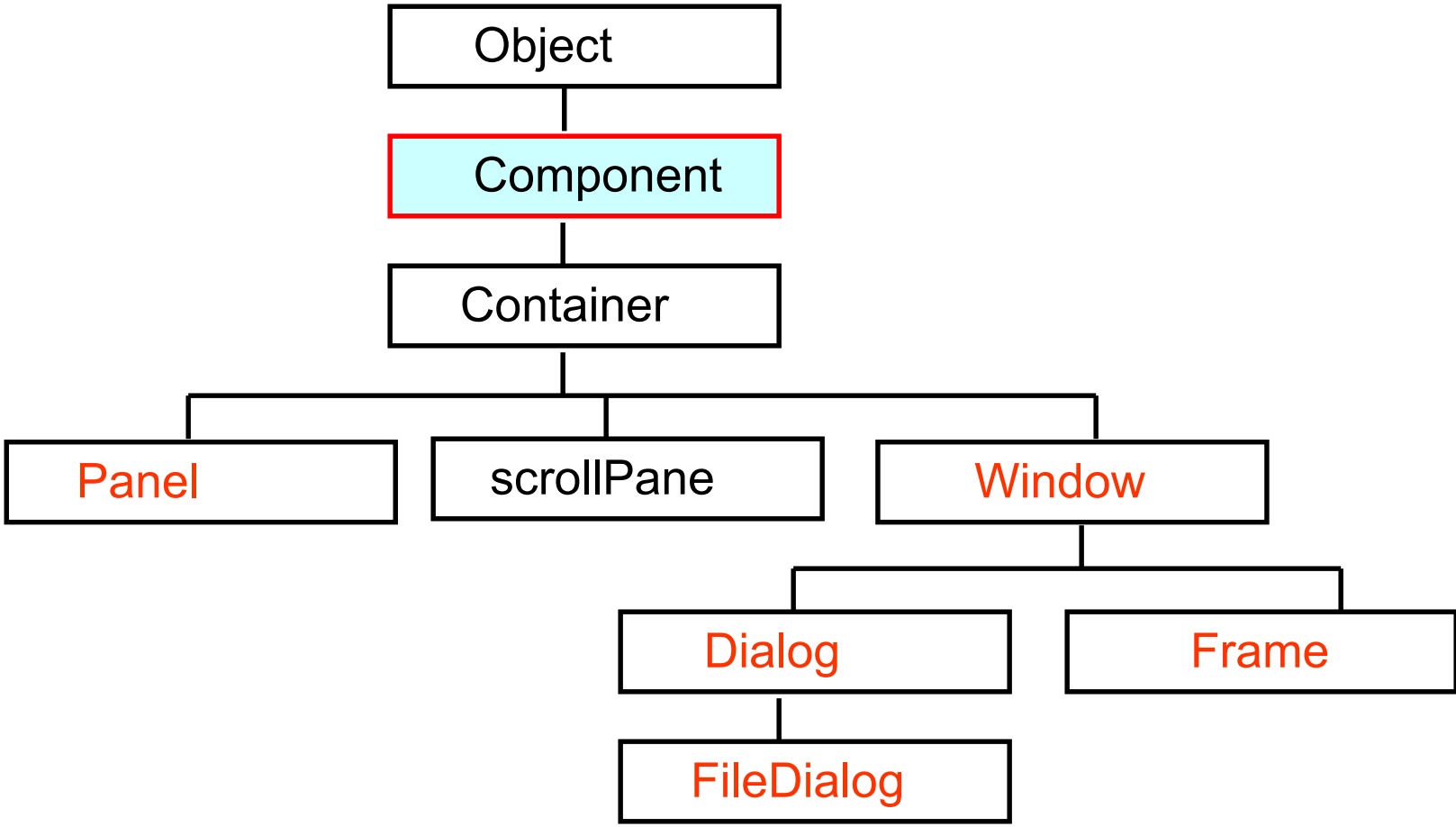
- (1)选择合适的AWT组件
- (2)选择合适的AWT容器
- (3)选择合适的AWT布局管理器

(2)选择合适的AWT容器

- 容器能容纳其他组件 ,是用来组织组件的.
- 容器也是一种特殊的组件.所以也能够放到其他的容器中

```
public class Container extends Component
{
    public void setLayout(LayoutManager mgr)        //设置布局管理器
    public Component add(Component comp)
                                          //在容器中添加一个组件comp
    ……..
}
```

B: 容器使用


Object
Component
Container
Panel
scrollPane
Window
Dialog
Frame
FileDialog

1.Panel类

- 一般用来被其他容器使用组织组件.没有可见边
- 构造函数

```
Panel();
Panel(LayoutManager layout);
```

- 方法使用

```
Panel p=new Panel();
p.add(new Button("Cancel"));
```

Panel举例：

```
public class PanelDemo extends Frame {
    public PanelDemo(String title) {
    super(title);
    Panel Panel11111=new Panel();
    Panel Panel22222=new Panel();
    Panel Panel33333=new Panel();
    Panel11111.setLayout(new CardLayout());
    Panel22222.setLayout(new FlowLayout());
    Panel33333.setLayout(new GridLayout());
    for (int i = 0; i < 5; i++)
    Panel11111.add(new Button("Button1-" + i));
    for (int i = 0; i < 5; i++)
    Panel22222.add(new Button("Button2-" + i));
    for (int i = 0; i < 5; i++)
    Panel33333.add(new Button("Button3-" + i));
    this.add(Panel11111,BorderLayout.NORTH);
    this.add(Panel22222,BorderLayout.CENTER);
    this.add(Panel33333,BorderLayout.SOUTH);
    setSize(250, 250);
    setVisible(true);
    }
    public static void main(String[] args) {
    PanelDemo pd = new PanelDemo("PanelDemo");}}
```

2.Frame类

- 是AWT应用程序的顶层容器。框架由标题栏，菜单栏和各种修饰物(最大化，最小化和关闭按钮)组成
- 构造函数

  见JDK(4个)
- 方法使用

  setTitle/getTitle（）-------设置和返回框架的标题

  setMenuBar/getMenuBar()------设置和返回框架的菜单栏

  setExtendedState()/getExtendedState()-----设置和返回框架的状态（Normal，Iconified，Maximized)

  setResizable(boolean bool)-----设置是否可以调整大小

Frame举例：

```
import java.awt.*;
 class FrameDemo1 extends Frame
  {
     FrameDemo1(String title)
     {  super(title);
        Button b=new Button("Hello");
        add(b,BorderLayout.WEST);
        Button b=new Button("GoodBye");
        add(b,BorderLayout.EAST);
        pack();                 //调整按钮,使他们达到首选的大小
        setResizable(false);    //让框架窗口不可调整大小
        setVisible(true);
      }
    public static void main(String[] args)
     { new FrameDemo1("Frame Demo");
     }
  }
```

3.Dialog类

- 对话框有两种方式：模态和非模态

  ---模态对话框要求用户在操作其他**GUI**组件之前,强迫用户关闭该对话框
  (只有关闭按钮，没有最小化和最大按钮**)**
  -----非模态对话框允许用户在操作对话框时,还可以对其他对话框进行操作。
  (有最大最小按钮**)**

- 构造函数

  见**JDK(**共**9**个**)**

- 方法使用

```
Dialog d=new Dialog(this, "are you sure? ",true);//假设是Frame容器中创建
d.add(new Button("yes"),"West");
d.add(new Button("No"));     //默认是center
d.add(new Button("Don’t know"),"East");
```

Dialog举例：

```
import java.awt.*;

class DialogDemo{
  public static void main(String[] args)
  {
    Frame f=new Frame("Dialog Demo");
    f.setSize(200,100);
    f.setVisible(true);
    Dialog d=new Dialog(f,"Is the sky blue?",true);
                                        // --创建一个模态对话框
    Panel p=new Panel();              //--创建一个Panel容器用来存放Button组件
    p.add(new Button("yes"));
    p.add(new Button("No"));
    d.add(p);
    d.setSize(200,100);
    d.setResizable(false);      //---设置d不可调整大小
    d.setVisible(true);                    //    ----使Dialog可见
  }
}
```

- 注：观察一下模态对话框和**Frame**之间的关系

4.FileDialog类

- ---用于文件选择的组件，是一种独立的，可移动的窗口。
- ---是Dialog的子类，分为打开(Open)和保存(Save)文件对话框
- 构造函数：

  见JDK(3个)
- 方法使用

public String getFile() --获取选择的文件

public void setFile(String file) ---设置文件名

public String getDirectory(String dir) -----获取选择的路径

public void setDirectory(String dir) ---设置文件的路径

注：构造的文件对话框默认是不可见的，需要调用**setVisible(boolean b)**来选择显示

FileDialog类

```
import java.awt.*;

 class DialogDemo{
   public static void main(String[] args)
   {
     Frame f=new Frame("Dialog Demo");
     f.setSize(200,100);
     f.setVisible(true);
     FileDialog d=new FileDialog(f,"this is a FileDialog",FileDialog.LOAD);
     //d.setDirectory("e:/qmx");
     //System.out.println(d.getDirectory());

     d.setSize(200,100);
     d.setResizable(false);
     d.setVisible(true);
     //f.add(d);  //本身就是Window不可添加到容器
   }
}
```

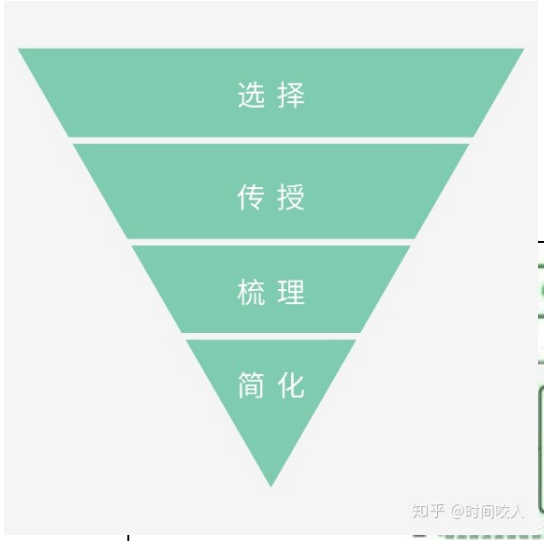
引申与总结:

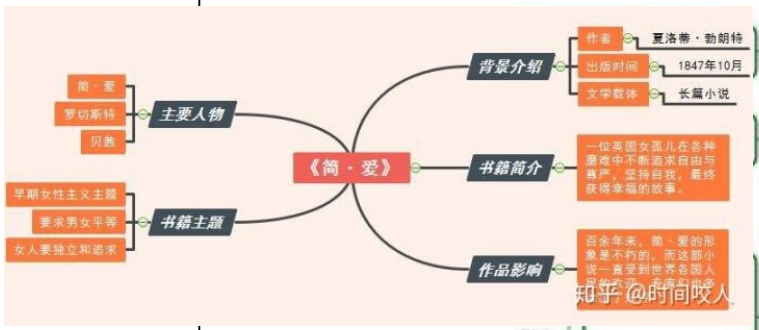
- **Container**和**Component**之间的关系就是设计模式之**Composite**（合成物）的实现

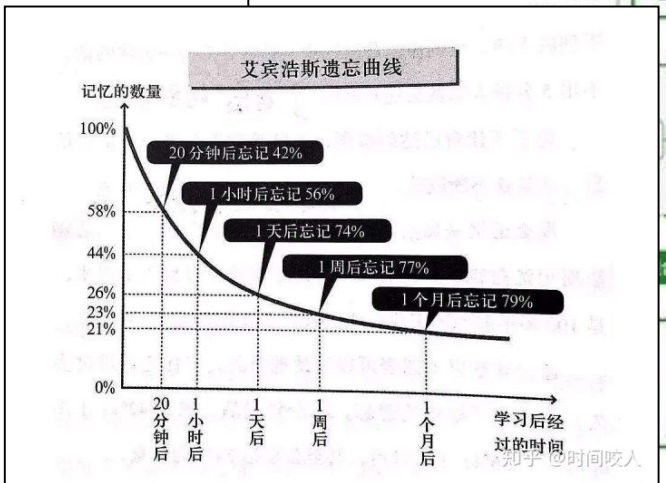
设计模式

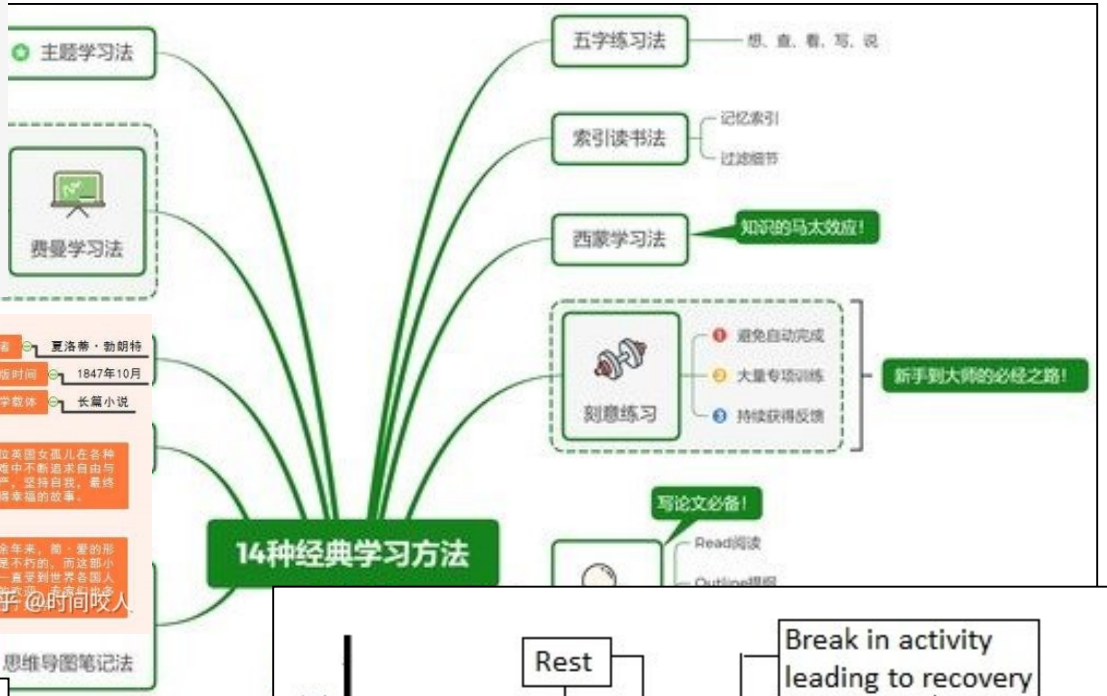
- 设计模式（Design Pattern）是前辈们对代码开发经验的总结，是解决特定问题的一系列套路。
- 它不是语法规定，而是一套用来提高代码可复用性、可维护性、可读性、稳健性以及安全性的解决方案。
- http://c.biancheng.net/design_pattern
- https://zhuanlan.zhihu.com/p/93770973
- 豆浆配油条

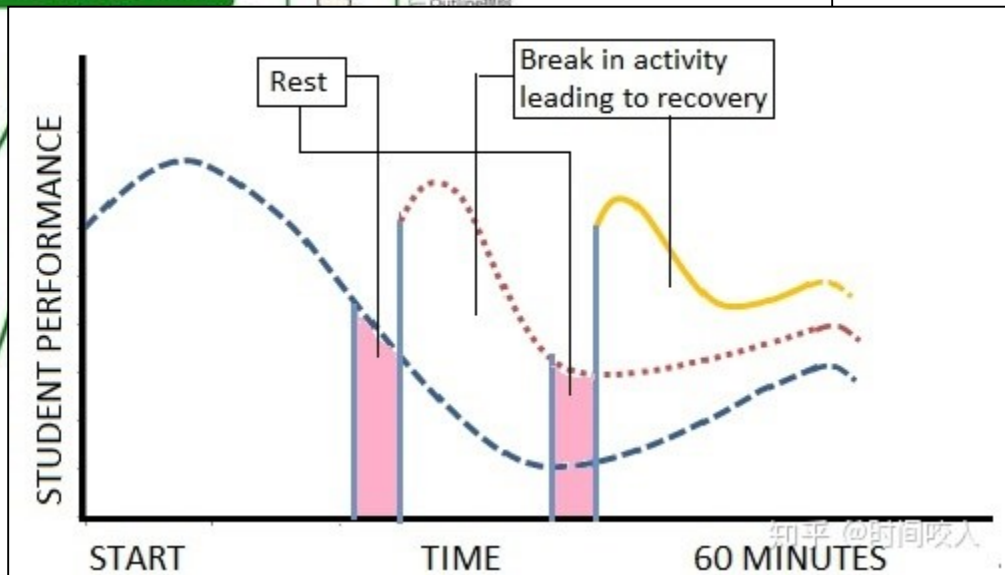
学习模式


选 择
传 授
梳 理
简 化


《简·爱》
主要人物
简·爱
罗切斯特
贝茵
书籍主题
早期女性主义主题
要求男女平等
女人要独立和追求
背景介绍
作者
夏洛蒂·勃朗特
出版时间
1847年10月
文学载体
长篇小说
书籍简介
一位英国女孤儿在各种磨难中不断追求自由与尊严，坚持自我，最终获得幸福的故事。
作品影响
百余年来，简·爱的形象是不朽的，而这部小说一直受到世界各国人
知乎 @时间咬人


艾宾浩斯遗忘曲线
记忆的数量
100%
58%
44%
26%
23%
21%
0%
20 分钟后忘记 42%
1 小时后忘记 56%
1 天后忘记 74%
1 周后忘记 77%
1 个月后忘记 79%
20 分钟后
1 小时后
1 天后
1 周后
1 个月后
学习后经过的时间
知乎 @时间咬人


14种经典学习方法
主题学习法
费曼学习法
思维导图笔记法
艾宾浩斯
康奈尔笔记法
合上课记录要点!
五字练习法
想、直、看、写、说
索引读书法
记忆索引
过滤细节
西蒙学习法
知识的马太效应!
刻意练习
避免自动完成
大量专项训练
持续获得反馈
新手到大师的必经之路!
写论文必备!
Read阅读
知乎 @时间咬人


Rest
Break in activity leading to recovery
STUDENT PERFORMANCE
START
TIME
60 MINUTES
知乎 @时间咬人

设计模式分类清单

- （1）创建型模式，共五种：工厂方法模式、抽象工厂模式、单例模式、建造者模式、原型模式。
- （2）结构型模式，共七种：适配器模式、装饰器模式、代理模式、外观模式、桥接模式、组合模式、享元模式。
- （3）行为型模式，共十一种：策略模式、模板方法模式、观察者模式、迭代子模式、责任链模式、命令模式、备忘录模式、状态模式、访问者模式、中介者模式、解释器模式。

设计模式
之间的关系

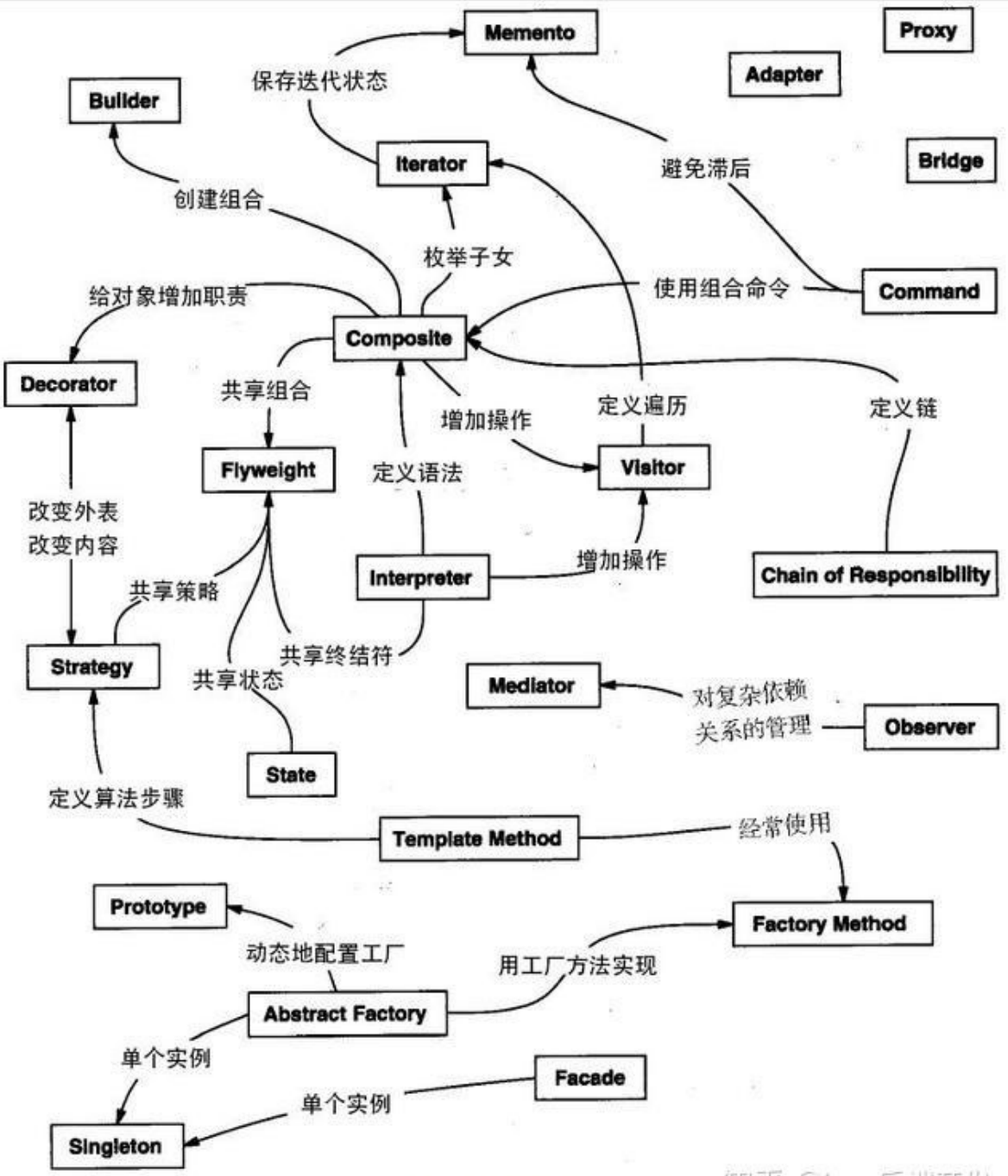
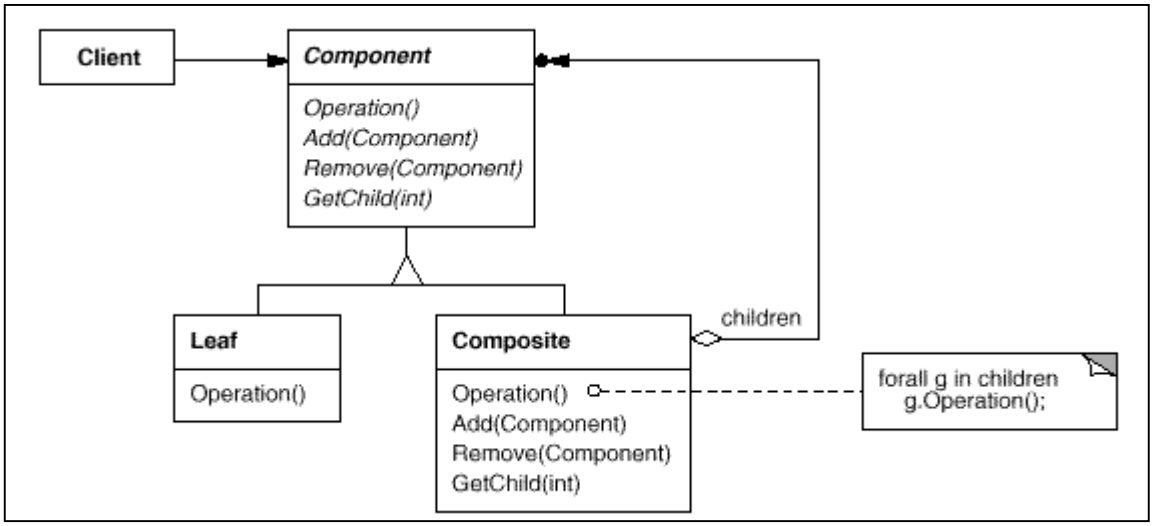
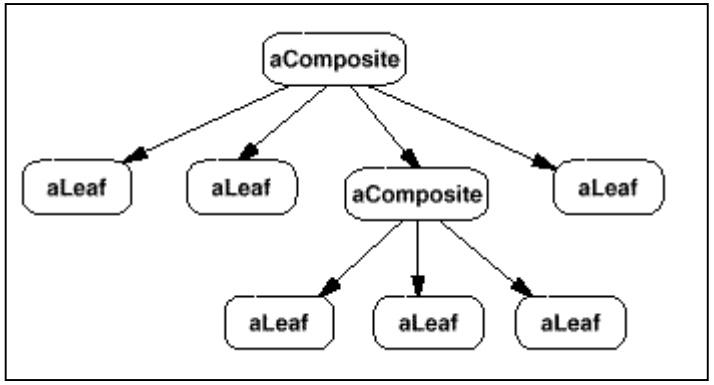


图　设计模式之间的关系

设计模式之组合模式（Composite ）

- **Container和Component之间的关系**





https://www.jianshu.com/p/685dd6299d96

第一阶段:

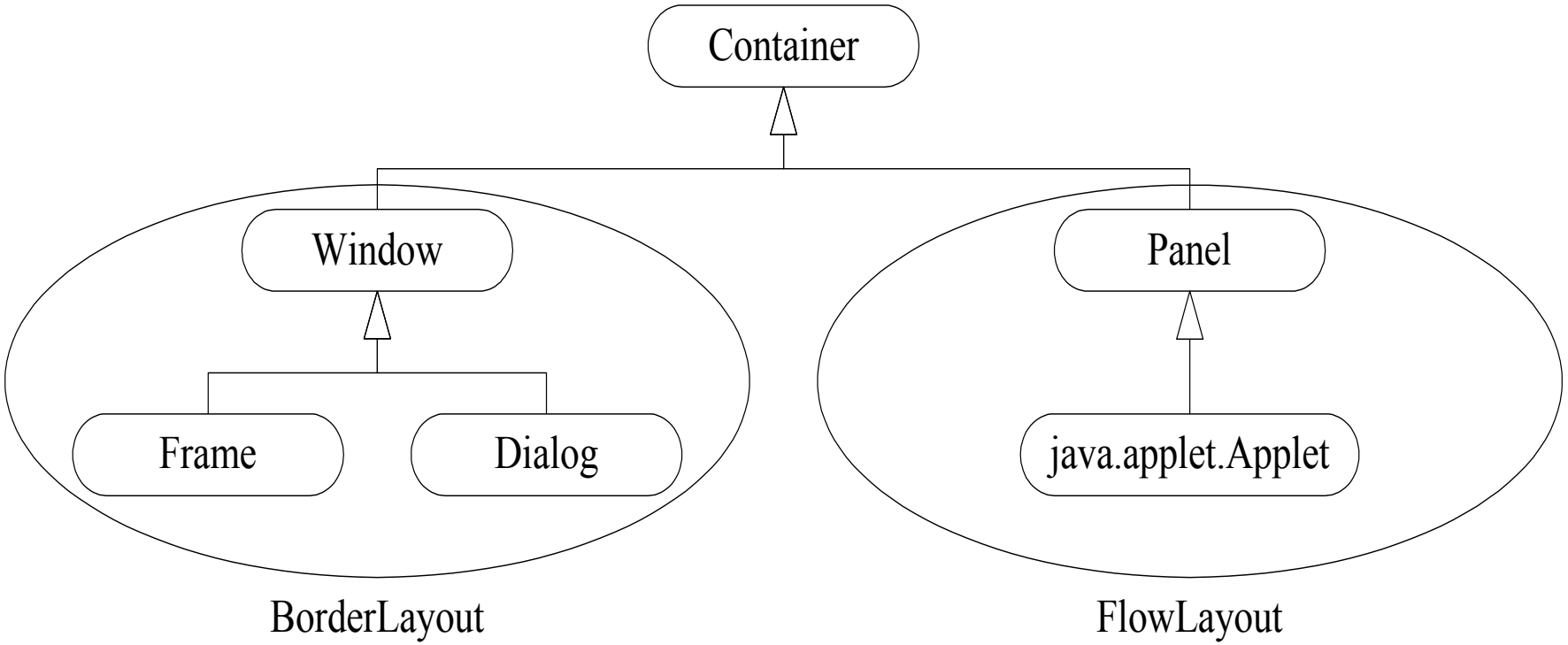
- (1)选择合适的AWT组件
- (2)选择合适的AWT容器
- (3)选择合适的AWT布局管理器

(3)选择合适的AWT布局管理器

- Java采用布局管理器(layout manager)对GUI中的组件进行相对定位并自动改变组件大小,合理分布组件格局.
- Java提供多种风格的布局管理器.
- 每一种布局管理器都有一个布局管理器类.
- 布局管理器是指容器的布局管理器.
- 每一种容器有一种默认的布局管理器.

C: 布局管理器使用

- 布局管理器和容器紧密相连(滑块没有布局管理器)
- 每个容器都有一个缺省的布局管理器，但是可以调**setLayout**来设置相应的布局管理器


Container
Window
Panel
Frame
Dialog
java.applet.Applet
BorderLayout
FlowLayout

容器的默认布局管理器

1.BorderLayout

- 一个边框布局管理器将它的容器分为五个区域：

  南部，北部，东部，西部，中部

- 在每一个区域只能放一个组件(但是可以通过添加一个容器(包括多个组件)到这个区域可以打破这种限制)
- 是对话框，文件对话框，框架和窗口的缺省布局管理器
- 构造函数

  见JDK（2个）
- 方法使用

  add(new Button(“ok”),BorderLayout.SOUTH);

BorderLayout举例：

```
import java.awt.*;
class BLDemo extends Frame{
    final static int NOGAPS=0;
    final static int GAPS=1;
    BLDemo(String title,int gapsOption)
   { super(title);
     if(gapsOption==GAPS)
      setLayout(new   BorderLayout(10,10));
     else
       setLayout(new BorderLayout());
    add(new Button("North"),"North");
   add(new Button("South"),"South");
   add(new Button("Center"),"Center");
   add(new
Button("West"),BorderLayout.WEST);
   add(new Button("East"),"East");
   setSize(250,250);
   setVisible(true);
   }
public static void main(String[] args)
{ int option=NOGAPS;
  for(int i=0;i<args.length;i++)
    if(args[i].equalsIgnoreCase("GAPS"))
       option=GAPS;
    else if(args[i].equalsIgnoreCase("NOGAPS"))
    option=NOGAPS;
    new BLDemo("BorderLayout Demo",option);
}

}
```

2.FLowLayout

- 流布局管理器可以将它的容器分成一行或多行，并且按从上到下从左到右的方式添加组件到这些行中。
- 组件可以在容器的左边，右边和中间对齐
- 流布局管理器是面板(Panel)的缺省布局管理器
- **new FlowLayout()**创建一个流布局管理器，缺省的垂直和水平间隙是**5**个像素，并且是左对齐。
- **FlowLayout(int alignment)-----**改变对齐方式**(LEFT,RIGHT,CENTER)**
- **FlowLayout(int alignment,int hgap,int vgap)**

FlowLayout举例：

```
import java.awt.*;
class FLDemo extends Frame
{  final static int NOGAPS=0;
   final static int GAPS=1;
   FLDemo(String title,int gapsOption,int alignOption)
    { super(title);
      if(gapsOption==GAPS)
        setLayout(new FlowLayout(alignOption));
      else
        setLayout(new FlowLayout(alignOption,0,0));
      for(int i=0;i<10;i++)
       add(new Button("Button"+i));
      setSize(250,250);
      setVisible(true);
}
 public static void main(String[] args)
  {  int option1=NOGAPS;
     int option2=FlowLayout.LEFT;
     for(int i=0;i<args.length;i++)
if(args[i].equalsIgnoreCase("LEFT"))
    option2=FlowLayout.LEFT;
else
  if(args[i].equalsIgnoreCase("RIGHT"))
     option2=FlowLayout.RIGHT;
 else
 if(args[i].equalsIgnoreCase("CENTER"))
    option2=FlowLayout.CENTER;
 else
  if(args[i].equalsIgnoreCase("GAPS"))
     option1=GAPS;
  else
   if(args[i].equalsIgnoreCase("NOGAPS"))
     option1=NOGAPS;
   new FLDemo("FlowLayout Demo",option1,option2);
}
}
```

3.GridLayout

- 网格布局管理器将它的容器分成行列的网格，每行和每列的交点称为cell单元。一个组件放在它的cell中，并且每个cell大小都相同
- 行列的数量是0基础的。
- 构造函数

GridLayout()------创建一个网格布局管理器（若干行和若干列)，没有间隙

GridLayout(int rows，int columns)-------创建一个rows*columns个cell，没有间隙

GridLayout(int rows,int columns,int hgap,int vgap)

GridLayout举例：

```
import java.awt.*;
class GLDemo extends Frame
{  final static int NOGAPS=0;
   final static int GAPS=1;
   GLDemo(String title,int gapsOption)
   {  super(title);
      if(gapsOption==GAPS)
        setLayout(new GridLayout(3,3,10,10));
      else setLayout(new GridLayout(3,3));
      for(int i=0;i<9;i++)
        add(new Button("Button"+i));
      setSize(250,250);
      setVisible(true);}
  public static void main(String[] args)
   {  int option=NOGAPS;
     for(int i=0;i<args.length;i++)
       if(args[i].equalsIgnoreCase("GAPS"))
          option=GAPS;
      else
       if(args[i].equalsIgnoreCase("NOGAPS"))
          option=NOGAPS;
      new GLDemo("GridLayout Demo",option);}
```

4.GridBagLayout

- 格包布局管理器是一个复杂的布局管理器，这个布局管理器能够创建一个网格(如同**GridLayout)**
- 但是格包管理器中每个单元格并不是同样的大小，一个单元可以占用多行和多列
- 见**JDK**

5.CardLayout

- 插件布局管理器将一个容器当作一系列的插件，仅有一个组件可以被放置在一个插件中，但是可以添加一个容器(包含多个组件)来打破这种限制
- 构造函数

  见**JDK**（共**2**个）
- 方法使用

```
Frame f=new Frame();
f.SetLayout(new CardLayout());
f.add(new Button("OK"), "card1");  ----添加按钮给一个名为Card1的插件
(CardLayout)getLayout().show(f, "card1");----显示card1插件
```

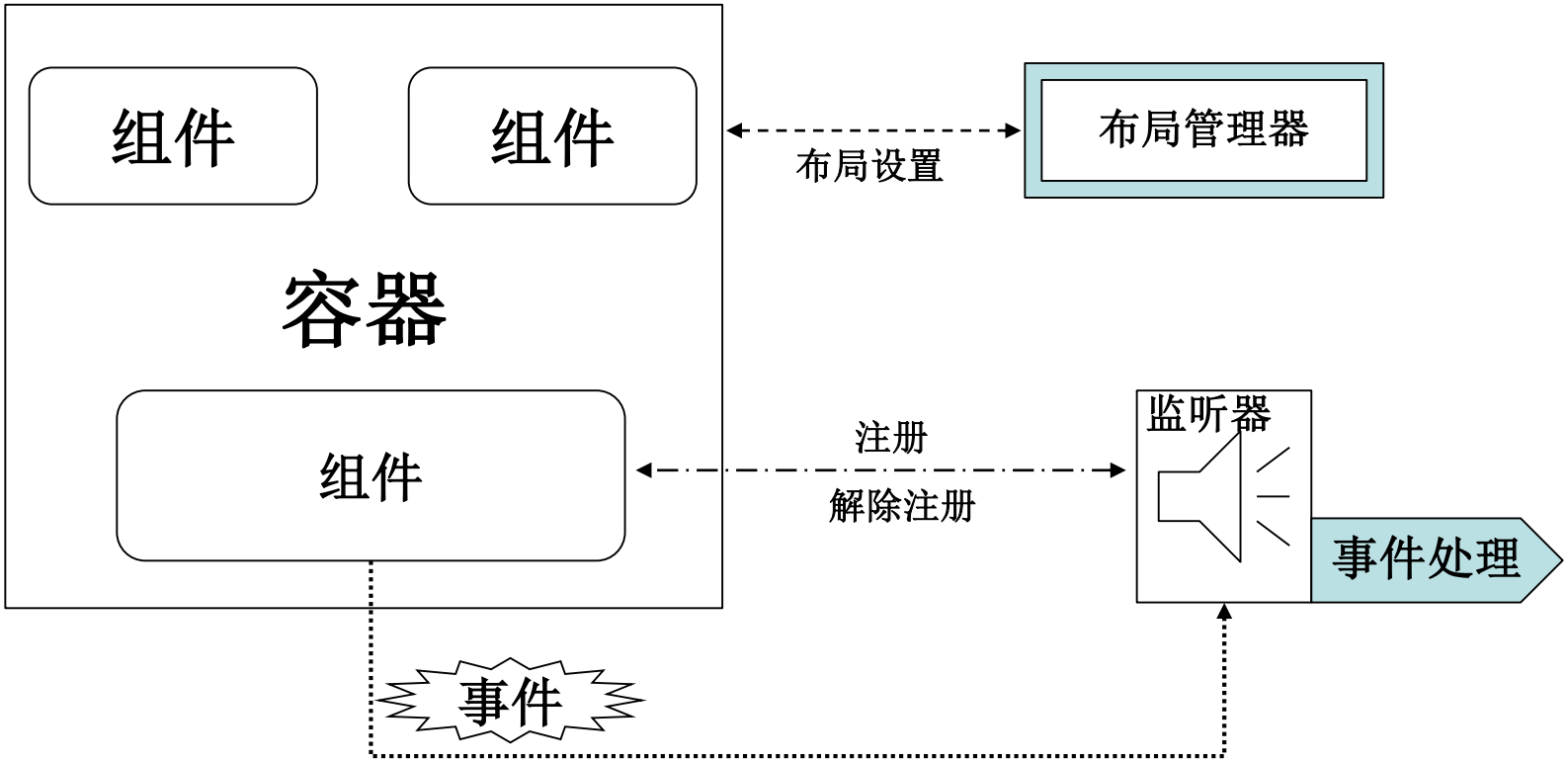
合并布局管理器举例：

```
import  java.awt.*;
 import java.applet.Applet;
 public class Calc extends Applet
{  public void init()
   { setLayout(new BorderLayout());     //BorderLayout
     add(new Label("0",Label.RIGHT),BorderLayout.NORTH);
     Panel p=new Panel();
     p.setLayout(new GridLayout(4,4)); //GridLayout
     String[ ] btnNames={"7","8","9","/","4","5","6","X","1","2","3","-
","0",".","C","+"};
     for (int i=0;i<btnNames.length;i++)
        p.add(new Button(btnNames[i]));
     add(p,"Center");
   }
 }
```

引申与思考

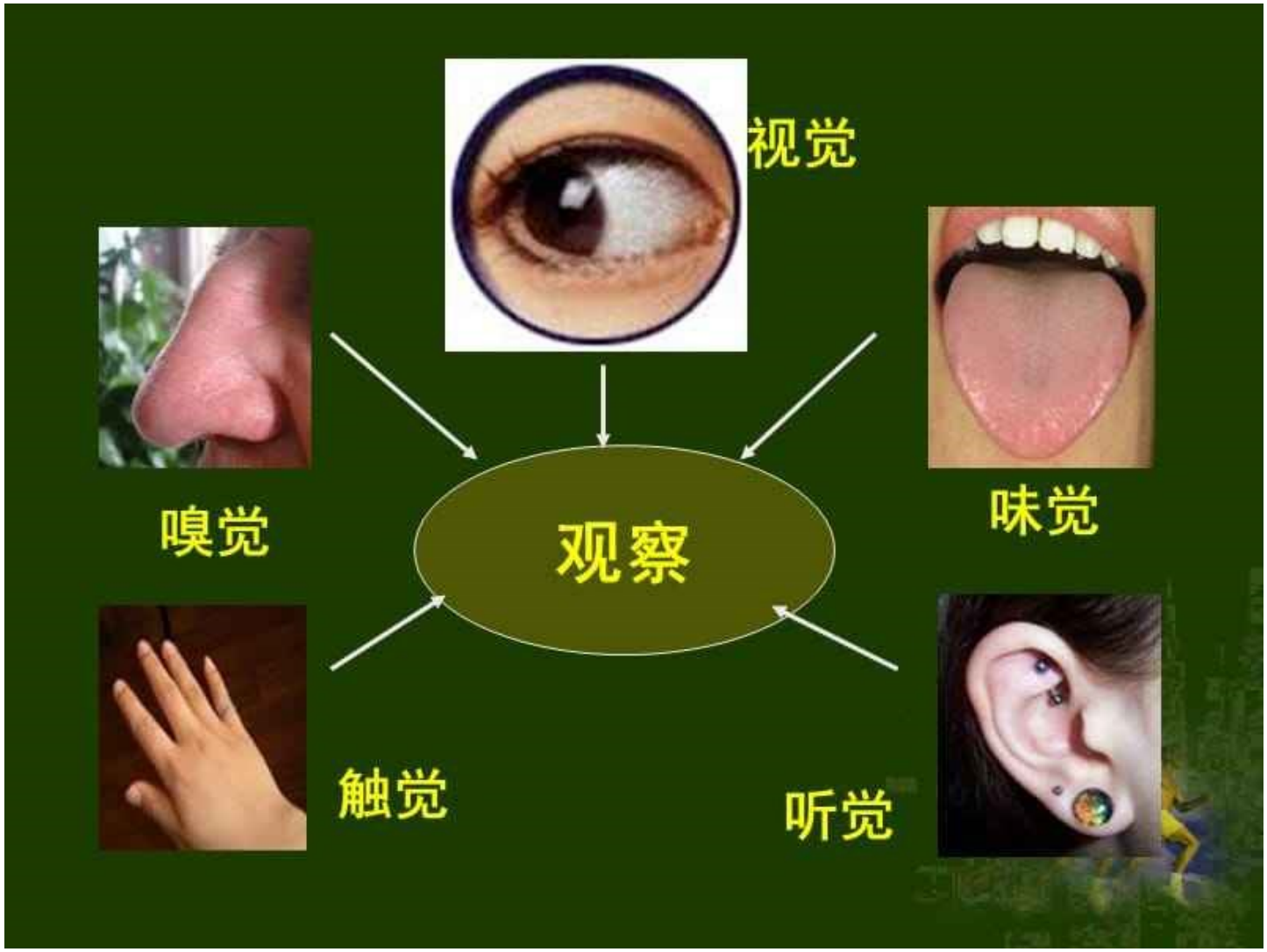
- **AWT的LayoutManager 就是Stragety设计模式的Java 应用**

第二阶段:创建GUI感觉（事件处理）



- **AWT事件**
- **AWT事件监听器**　　addXXXListener( ) removeXXXListener( )

人体的感觉系统



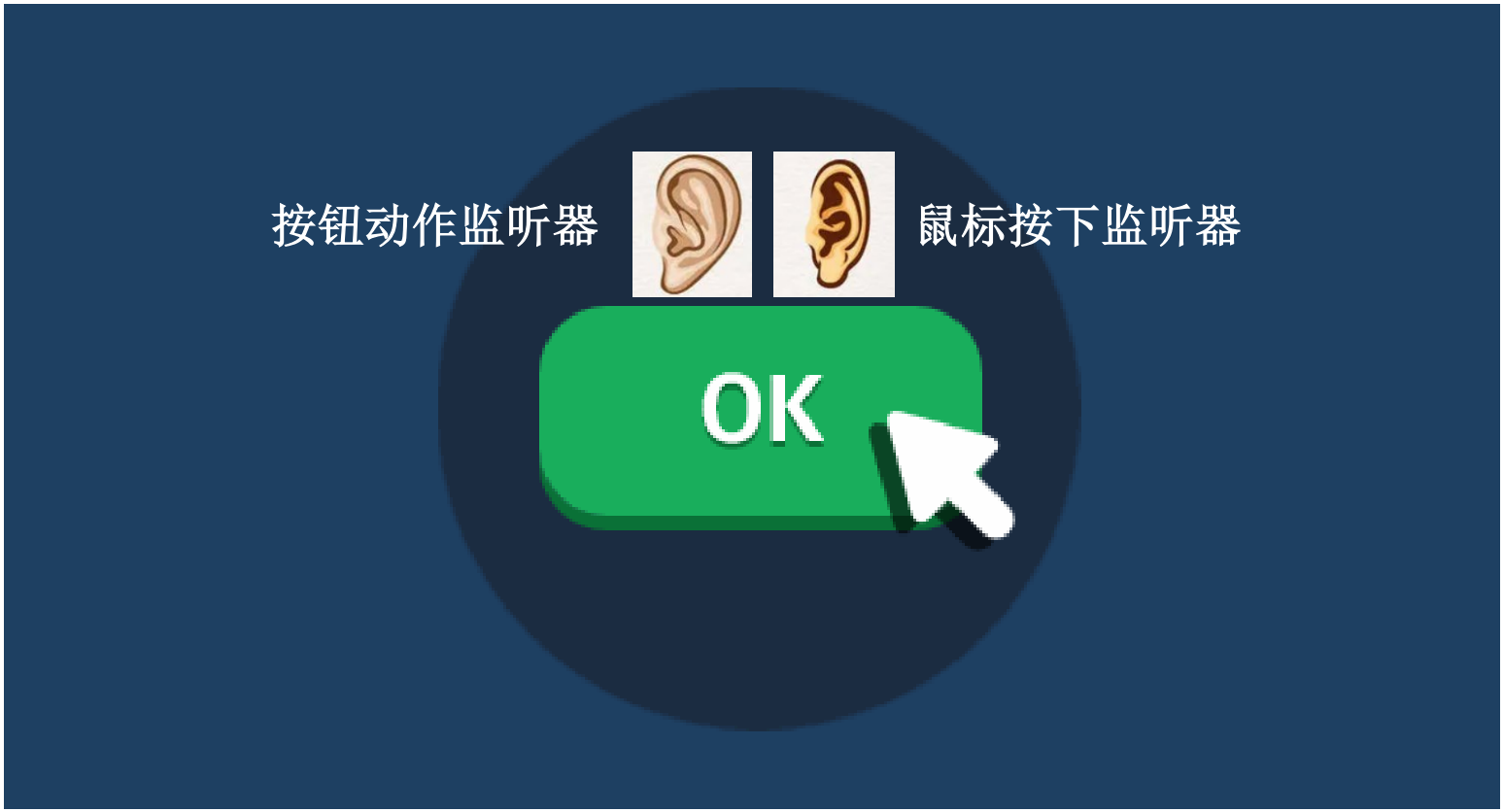
人造感觉系统（监听器和事件）

注意：仅在需要的地方安装合适的监听器

——需要的地方指能够产生对应事件的地方

——合适的监听器指能够监听对应事件的监听器

GUI程序的感觉



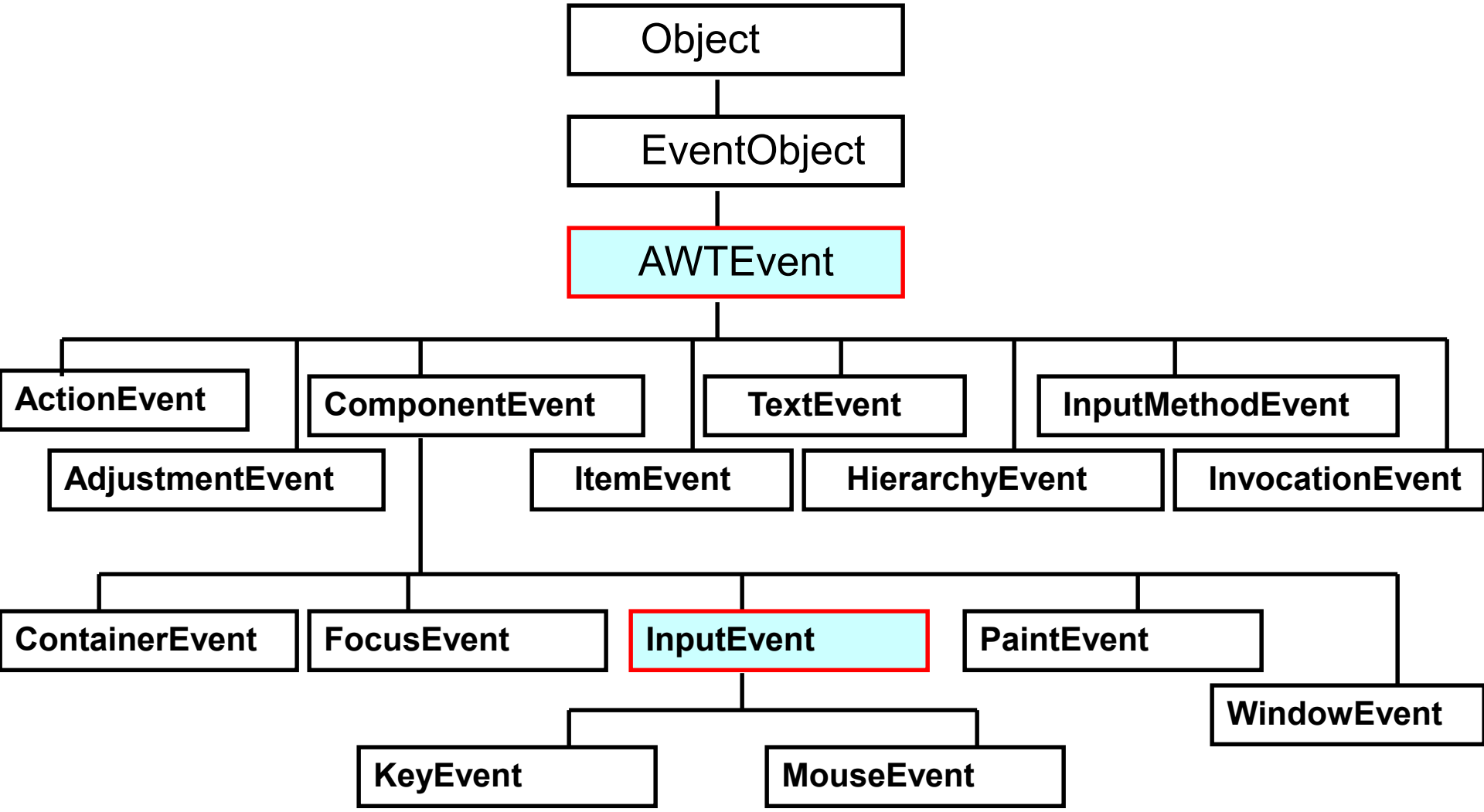
注意：按钮按下和鼠标按下不是一回事（键盘也可），虽然GUI程序多数情况下是一回事

AWT Event：

- 事件是一个状态的改变或一个活动的发生.
- Java将不同的事件封装成不同的事件类
- 产生事件的事件源(组件)需要事件监听器来负责响应什么事件,以及对响应事件进行处理.

- **(A)：事件研究：**

- **Java**中组件等的一系列相关的动作将激发相应的事件。
- 窗口工具集将事件和对象相联，提供了事件发生的描述性信息。该对象是从抽象**AWTEvent**类的子类中创建的。
- 理解**AWTEvent**和它的子类是创建**GUI**的第四步


Object
EventObject
AWTEvent
ActionEvent
ComponentEvent
TextEvent
InputMethodEvent
AdjustmentEvent
ItemEvent
HierarchyEvent
InvocationEvent
ContainerEvent
FocusEvent
InputEvent
PaintEvent
WindowEvent
KeyEvent
MouseEvent

1.行为事件（ActionEvent）

- 当组件指定行为发生时，组件产生行为事件。
- 从**ActionEvent类(java.awt.event)**中创建的对象表示行为对象
- 当产生一个行为事件时，用户可能按住**Alt**、**Ctrl**、或**Shift**键。
  ActionEvent.getModifiers()返回这几个修改键的整数值。
  可以用**&**(位与)判断是否按下了修改键
  if(e.getModifiers()&ActionEvent.ALT_MASK)
- 按下相应的菜单项也会激发行为事件

2.焦点事件（FocusEvent）

- 当一个组件得到或者失去焦点时，发生了焦点事件
- 从FocusEvent类中创建的对象表示焦点事件

3.项目事件（ItemEvent）

- 当用户选择一个复选框、复选框菜单项、选择列表或列表项时触发项目事件
- 从ItemEvent类中创建的对象表示项目事件
- **getSource()　----**返回事件的源
- **getselectedItems()----**返回所选项目的数组
- **.SELECTED/.DESELECTED----**表示选定或者是撤销选定
- **getStateChange()----**判断某个项目是否被选定或撤销了选定

4.击键事件（KeyEvent）

- 当用户按下或者释放一个键时，发生击键事件，分为：键按下、键释放、键输入
- 键按下/释放由一个**keycode(32**位整数)来表征。
- 当按下一个或者多个键产生一个**Unicode**字符，产生键输入事件；对于每一个键输入事件都至少对应一个键按下事件
- 从**KeyEvent**类中创建的对象表示键事件
- 键按下判断可以采用

  if(e.getKeyCode()==KeyEvent.VK_ENTER)
- 键输入判断可以采用

  if(e.getKeyChars()==‘A’)

5.鼠标事件（MouseEvent）

- 当用户按下鼠标、释放鼠标、移动鼠标时发生鼠标事件。
- 从MouseEvent类中创建的对象代表鼠标事件
- getX()/getY()----返回鼠标指针的坐标(相对于事件源的坐标)

- PopupMenu未完待续（关联鼠标）
- 对于OS设定的弹出式菜单操作：

```
if(e.isPopupTrigger())
                //获取弹出式菜单触发器的设置(激活一个弹出式菜单时，为真
{  popup.show(e.getComponent(),e.getX(),e.getY());
 }
```

6.文本事件（TextEvent）

- 当一个文本框或者文本域的内容发生改变时，触发文本事件
  (在文本区域按下键或者用setText()方法时都会触发)
  从TextEvent类中创建的对象表示文本对象

7.窗口事件（WindowEvent）

- 当一个窗口被激活，撤销激活，打开，关闭，图标化或撤销
- 图标化发生窗口事件
- 从WindowEvent类中创建的对象代表窗口事件
- 要标识激发事件的容器类窗口，如对话框，文件框或框架，需要调用WindowEvent的getWindow()方法

第二阶段:事件处理

- AWT事件
- AWT事件监听器

- 事件处理方法(Event Handler):

  --能够接收，处理事件类对象，实现与用户交互功能的方法

- 事件监听器(Event Listener)

  --调用事件处理方法的对象

B)：监听器研究(Event Listener)

- 创建并注册监听器是创建GUI的最后一步
- 一个监听器对象使用一个源来注册自己，然后当事件发生时，这个源调用该监听器
- 要成为一个源的注册监听器，对象必须从一个实现监听器界面的类中创建，并且必须调用一个源注册方法

1.行为监听器

- 行为监听器监听行为事件，由实现**ActionListener**接口的对象来表示
- **ActionListener**中有一个方法，必须由行为监听器类来实现

  public abstract void actionPerformed(ActionEvent e)

  {….}

- 例一个行为监听器：

```
…..
public class ActionListenerDemo
                                implements   ActionListener
{ public void regListener()
 {
     Button b=new Button(“ok”);
     b.addActionListener(this);
  }
public void actionPerformed(ActionEvent e)
  { ……….}
….}
```

行为监听器举例：

```
import  java.awt.*;
 import java.awt.event.*;
 import java.applet.Applet;
 public class Calc extends Applet
                          implements ActionListener
{    Label l;
   public void init()
   { setLayout(new BorderLayout());
      l=new Label("0",Label.RIGHT);
     add(l,  BorderLayout.NORTH);
     Panel p=new Panel();
     p.setLayout(new GridLayout(4,4));
     String[ ]
btnNames={"7","8","9","/","4","5","6","X","1","2","3","-
","0",".","C","+"};
     for (int i=0;i<btnNames.length;i++) {
       Button b=new Button(btnNames[i]);
       b.setActionCommand(" "+i);   //思考
       p.add(b);
       b.addActionListener(this); }
       add(p,”Center”);     }
```

```
public void actionPerformed(ActionEvent e)
     { String id=e.getActionCommand();
       l.setText(id);
//       Button b = (Button)e.getSource();
//       l.setText(b.getLabel());//获取按钮属性
       }
}
```

用什么进行计算？

ActionCommand

界面、逻辑相分离思想

2.焦点监听器

- 焦点监听器监听焦点事件。他们由实现**FocusListener**接口的对象表示
- **FocusListener**指定了两个方法，它们必须由监听器类来实现。方法为：

```
public abstract void focusGained(FocusEvent e)
public abstract void focusLost(FocusEvent e)
```

- 要注册一个焦点源，焦点监听器必须调用源的**addFcousListener**

焦点监听器举例：

```
import java.awt.*;
import java.awt.event.*;
class FocusListenerDemo extends Frame implements FocusListener
{  TextField numField;
   FocusListenerDemo(String title)
    {  super(title);

       Panel p=new Panel();
       p.add(new Label("Please enter a number"));
       numField=new TextField(20);
       numField.addFocusListener(this);
       p.add(numField);
       add(p,"North");

       p=new Panel();
       p.add(new Label("Please enter a name"));
       TextField nameField=new TextField(20);
       nameField.addFocusListener(this);
       p.add(nameField);
       add(p,"South");

       setSize(350,100);
       setVisible(true);
    }
```

```
    public void focusGained(FocusEvent e)
        {
          Component c=(Component)e.getSource();
          System.out.println("GAIN:"+c.getName());
        }
      public void focusLost(FocusEvent e)
        {  Component c=(Component)e.getSource();
           System.out.println("LOST:"+c.getName());
           if(e.getSource()==numField)
            try{
               int i=Integer.parseInt(numField.getText());
            }
            catch(NumberFormatException e2)
            {
               numField.requestFocus();                    //让焦点停留在numField上
            }
        }
      public static void main(String[] args)
      {
        new FocusListenerDemo("Focus Listner Demo");
      }
    }
```

3.项目监听器

- 项目监听器监听项目事件。
- 由实现**ItemListener**接口的对象表示，此接口指定了一个简单的方法，该方法必须由项目监听器的类实现。

  public abstract void itemStateChanged(ItemEvent e)
- 当一个复选框、复选框菜单项、选择或列表的状态发生改变时，调用**itemStateChanged**
- 要注册一个项目的源，项目监听器必须调用源的**addItemListener**

4.击键监听器

- 击键监听器监听击键事件。由实现**KeyListener**接口的对象表示
- **KeyListener**指定了**3**个方法：

public abstract void keyPressed(KeyEvent e) 按下某个键
public abatract void keyReleased(KeyEvent e) 释放某个键
public abstract void keyTyped(KeyEvent e) 键入某个键

- 注册一个键源，必须调用源的**addKeyListener**

注意:中文状态下进行输入时，keyPressed方法不触发，keyReleased仍然触发，keyTyped在中文输入完成统一触发。

击键监听器举例：

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

class KeyListenerDemo extends Frame implements KeyListener
{
  KeyListenerDemo(String title)
   {  super(title);
      addKeyListener(this);
      setSize(300,300);
      setVisible(true);
   }
  public void keyPressed(KeyEvent e)
   { System.out.println("Key Pressed:virtual key code="+e.getKeyCode());
   }
  public void keyReleased(KeyEvent e)
   { System.out.println("Key Released:virtual key code="+e.getKeyCode());
   }
  public void keyTyped(KeyEvent e)
    { System.out.println("Key typed:character="+e.getKeyChar());
    }
  public static void main(String[] args)
    { new KeyListenerDemo("Key Listener Demo");
    }
```

5.鼠标监听器

- 鼠标监听器监听鼠标事件。鼠标按钮相关事件由实现MouseListener的对象实现
- MouseListener指定5个方法：

```
public abstract void MouseClicked(MouseEvent e)
public abstract void MouseEntered(MouseEvent e)
public abstract void MouseExited(MouseEvent e)
public abstract void MousePressed(MouseEvent e)
public abstract void MouseReleased(MouseEvent e)
```

- 要注册一个鼠标源，鼠标监听器必须调用源的addMouseListener
- 鼠标移动的相关事件由实现MouseMotionListener的对象实现.方法：

```
public abstract void MouseDragged(MouseEvent e)
public abstract void MouseMoved(MouseEvent e)
```

- 要注册一个鼠标移动源，鼠标监听器必须调用源的addMouseMotionListener

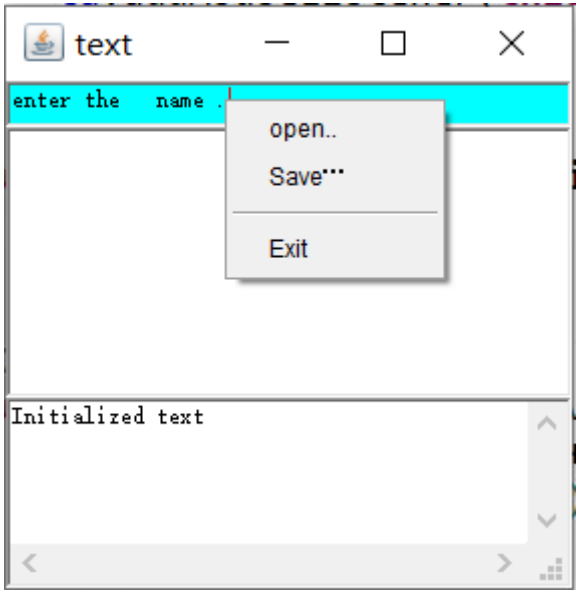
- 2.5D操作、图形化界面利器

鼠标监听器举例：

```
import java.awt.*;
import java.awt.event.*;
class MouseListenerDemo extends Frame implements
MouseListener,MouseMotionListener
{
  PopupMenu popup;
  MouseListenerDemo(String title)
   {
    super(title);
    popup=new PopupMenu("Hello");
    popup.add(new MenuItem("First"));
    popup.add(new MenuItem("Second"));
    popup.addSeparator();
    popup.add(new MenuItem("Third"));

    add(popup);
    addMouseListener(this);
    addMouseMotionListener(this);
    setSize(100,100);
    setVisible(true);
   }

```

```
    public void mouseEntered(MouseEvent e)
        {
          System.out.println("Mouse Entered");     }
      public void mouseExited(MouseEvent e)
        {  System.out.println("Mouse Exited"); }
      public void mousePressed(MouseEvent e)
        { System.out.println("Mouse Pressed");
          if(e.isPopupTrigger())
            popup.show(e.getComponent(),e.getX(),e.getY());      }
      public void mouseReleased(MouseEvent e)
        { System.out.println("Mouse Released");
          if(e.isPopupTrigger())
            popup.show(e.getComponent(),e.getX(),e.getY()); }
      public void mouseClicked(MouseEvent e)
        { System.out.println("Mouse Clicked"); }
      public void mouseMoved(MouseEvent e)
        { System.out.println("Mouse Moved");     }
      public void mouseDragged(MouseEvent e)
        { System.out.println("MouseDragged");}
      public static void main(String[] args)
       {new MouseListenerDemo("Mouse Listener Demo");
       }
    }
```

PopupMenu鼠标加持



6.文本监听器

- 文本监听器监听文本事件。它们由实现**TextListener**接口的对象实现。此接口中包含一个方法：

  public abstract void textValueChanged(TextEvent e)

- 要注册一个文本源，必须调用源的**addTextListener**
- 当用户输入或者调用**setText()**方法改变文本时，调用**textValueChanged()**

文本监听器举例：

```
import java.awt.*;
import java.awt.event.*;
class TextListenerDemo extends Frame implements TextListener
{  TextListenerDemo(String title)
  { super(title);
    Panel p=new Panel();
    p.setLayout(new FlowLayout(FlowLayout.LEFT));
    p.add(new Label("Please enter a name(8 chars maximum):"));
    TextField nameField=new TextField(10);
    nameField.addTextListener(this);
    p.add(nameField);
    add(p,"North");
    setSize(400,400);
    setVisible(true);  }
  public void textValueChanged(TextEvent e)
   { TextField tf=(TextField)e.getSource();
     String text=tf.getText();
     if(text.length()>8)     //text.length=9也是一样的道理
      { int cp=tf.getCaretPosition();
        tf.setText(text.substring(0,text.length()-1));
        tf.setCaretPosition(cp);   }}
  public static void main(String[] args)
    { new TextListenerDemo("Text Listener Demo");
    }
}
```

7.窗口监听器

- 窗口监听器监听窗口事件。包括打开一个窗口、关闭一个窗口、窗口关闭、图标化窗口、撤销窗口图标、激活一个窗口和释放一个窗口
- 监听器由实现**WindowListener**接口的对象表示，此接口共**7**个方法：

```
public abstract void WindowClosing(WindowEvent e)
public abstract void WindowClosed(WindowEvent e)
public abstract void WindowIconified(WindowEvent e)
public abstract void WindowDeiconified(WindowEvent e)
public abstract void WindowActivated(WindowEvent e)
public abstract void WindowDeactivated(WindowEvent e)
```

- 要注册一个窗口源，窗口监听器必须调用源的addWindowListener

表 11-1　AWT 事件及监听器接口

事件类别	接　　口	方法及参数
ActionEvent	ActionListener	actionPerformed(ActionEvent)
ItemEvent	ItemListener	itemStateChanged(ItemEvent)
AdjustmentEvent	AdjustmentListener	adjustmentValueChanged(adjustmentEvent)
ComponentEvent	ComponentListener	componentHidden(ComponentEvent)
		componentMoved(ComponentEvent)
		componentResized(ComponentEvent)
		componentShown(ComponentEvent)
MouseEvent	MouseListener	mouseClicked(MouseEvent)
		mouseEntered(MouseEvent)
		mouseExited(MouseEvent)
		mouseReleased(MouseEvent)
		mousePressed(MouseEvent)
MouseEvent	MouseMotionLiStener	mouseDragged(MouseEvent)
		mouseMoved(MouseEvent)
WindowEvent	WindowListener	windowActivated(WindowEvent)
		windowDeactivated(WindowEvent)
		windowOpened(WindowEvent)
		windowClosed(WindowEvent)
		windowClosing(WindowEvent)
		windowIconfied(WindowEvent)
		windowDeIconfied(WindowEvent)
KeyEvent	KeyListener	keyPressed(KeyEvent)
		keyReleased(KeyEvent)
		keyTyped(KeyEvent)
ContainerEvent	ContainerListener	componentAdded(containerEvent)
		componentRemoved(containerEvent)
TextEvent	TextListener	textValueChanged(TextEvent)
FocusEvent	FocusListener	focusGained(FocusEvent)
		focusLost(FocusEvent)

补充：适配器(Adapter) ---java.aw.event

- 适配器(Adapter)是实现一个监听器接口的类。这个类中的所有方法都被空操作实现。

- 要使用一个适配器，必须创建一个子类并且覆盖感兴趣的方法。然后从子类中创建一个对象并把它的引用传给一个合适的注册方法。

举例：

例：

```
…
  class MyWindowAdapter extends WindowAdapter
   { public void windowClosing(WindowEvent e)
         {System.exit(0);}
   }
   class WindowDemo extends Frame
   {  WindowDemo(String title)
     { super(title);
       MyWindowAdapter mwa=new MyWindowAdapter();
      addWindowListener(mwa);
      …….
     }
   }
```

AWT适配器

- ① **ComponentAdapter** 组件适配器。
  ② **ContainerAdapter** 容器适配器。
  ③ **FocusAdapter** 焦点适配器。
  ④ **KeyAdapter** 键盘适配器。
  ⑤ **MouseAdapter** 鼠标适配器。
- ⑥ **MouseMotionAdapter** 鼠标移动适配器。
  ⑦ **WindowAdapter** 窗口适配器。

引申—Java 类库对设计模式的应用

- 这种类的结构和定义就是设计模式中的**Adapter**设计模式的应用.

匿名内部类

- 必须引进一个新类，并且保证与已经定义的类名不重名。这也很麻烦。这样的情况下可以使用一个匿名内部类。
- 内部类是一个类可以定义在另一个类中的类。

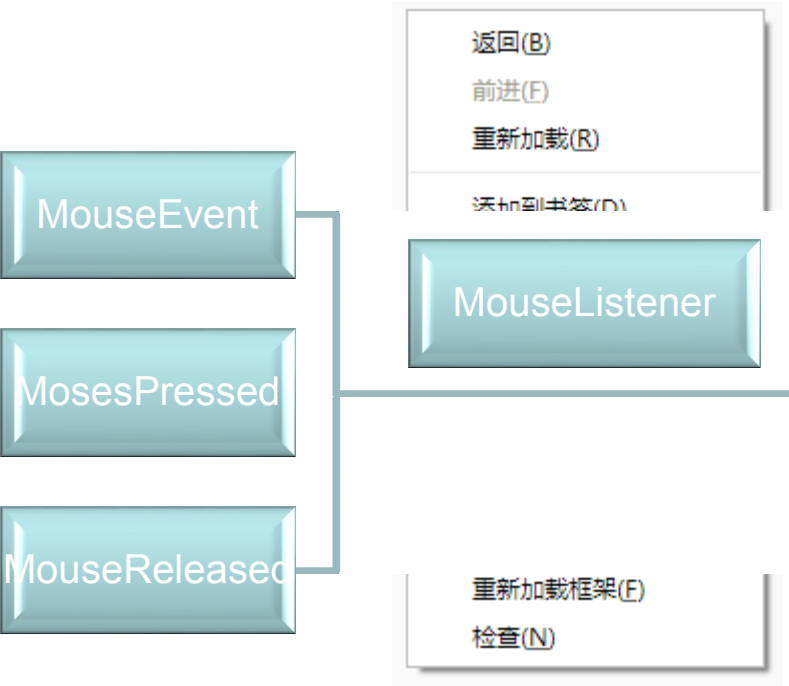
例：。。。。。

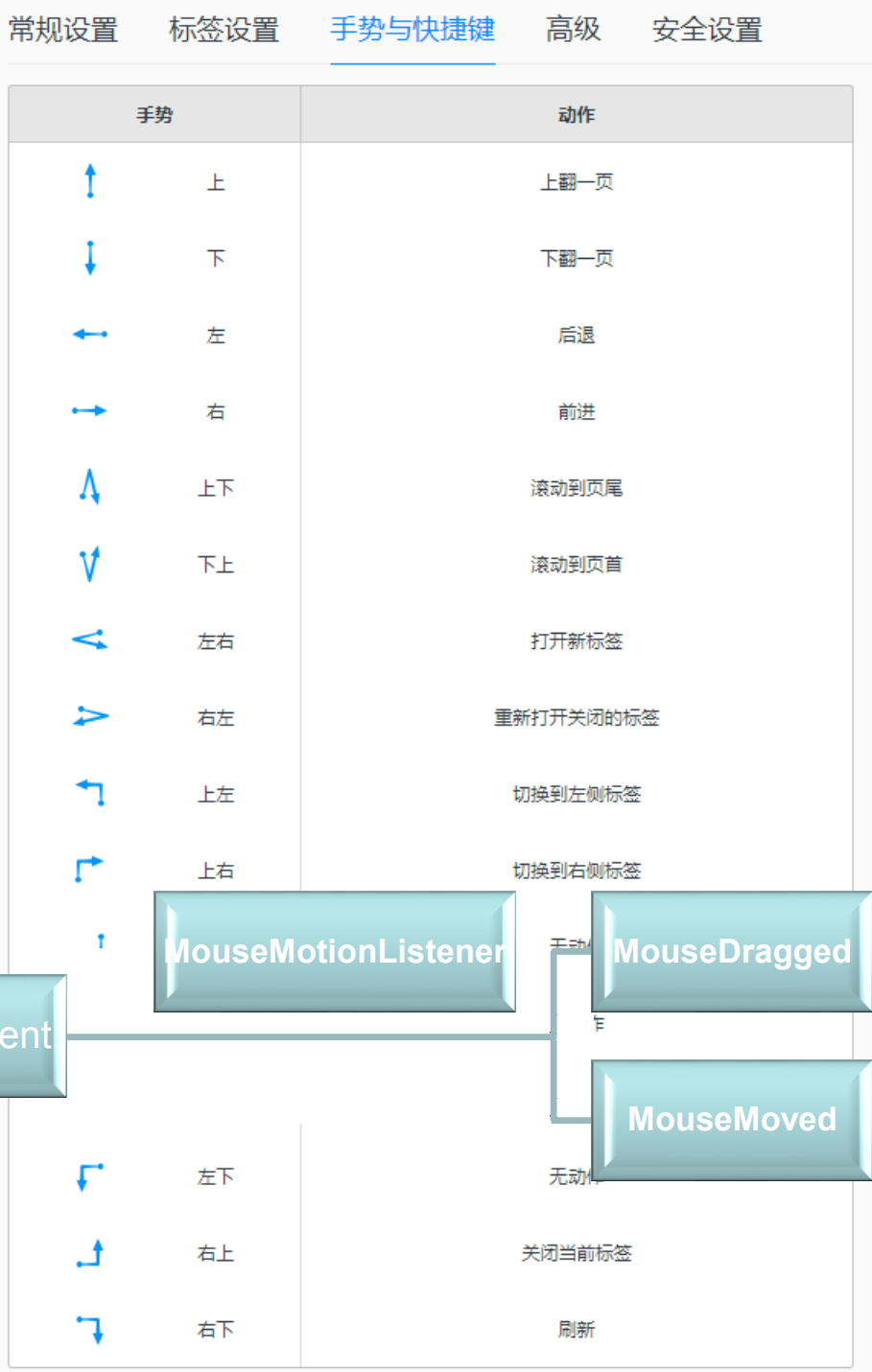
```
 class WindowAdapterDemo1 extends Frame
{  WindowAdapterDemo1(String title)
 {super(title);
  addWindowListener(new WindowAdapter()
                          {  public void windowClosing(WindowEvent e)
                            {System.exit(0);}
                          });
   setSize(500,500);
   setVisible(true);
}
public static void main…… ..
```

单组件配多监听器

组件 —— 监听器

1 —— N





单一监听器绑定多事件源
多事件源注册同一监听器

监听器 — 事件源
1 — N

MouseEvent e

Yes

No

Cancel

MouseListener

MouseEvent

```
mouseClicked(){
    if……yes_Function()
    if…… no_Function()
    if…… cancel_Function()
}
mousePressed……
mouseReleased ……
```

事件源组件 — 事件监听器 — 事件

GUI感觉 总结

- 界面不同功能属性，需要不同的事件类型

  TextEvent、 MouseEvent、……

- 事件类型与事件监听器一对一匹配

  TextEvent ←→ TextListener (除MouseEvent)

- 事件监听器与处理方法一对多匹配

  MouseListener→mouseClicked、mousePressed、mouseReleased

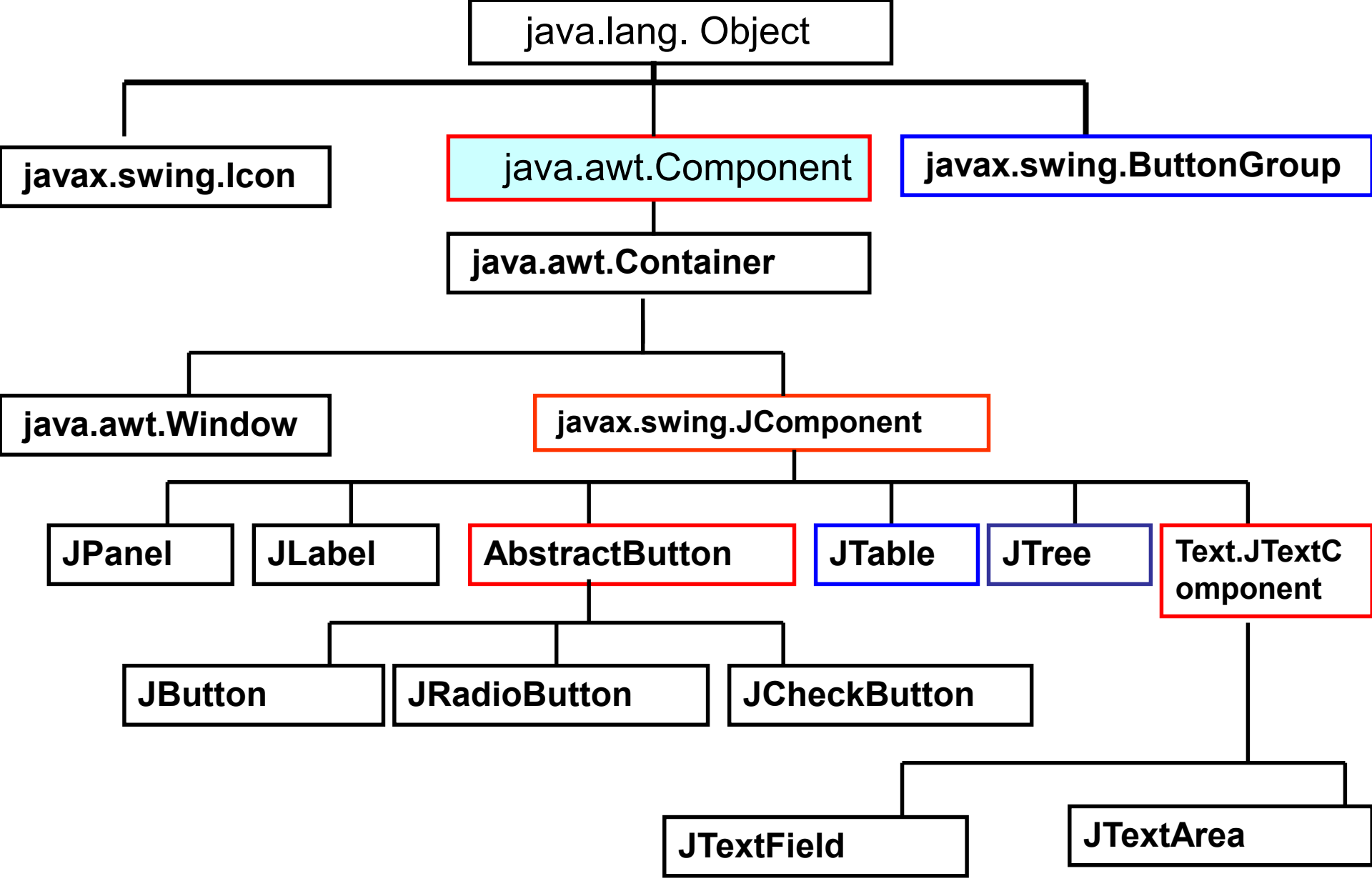
- 某个监听器可被多个组件注册，在监听处理方法通过事件携带的事件源对不同组件的匹配不同的处理方法

GUI外观与感觉的关系

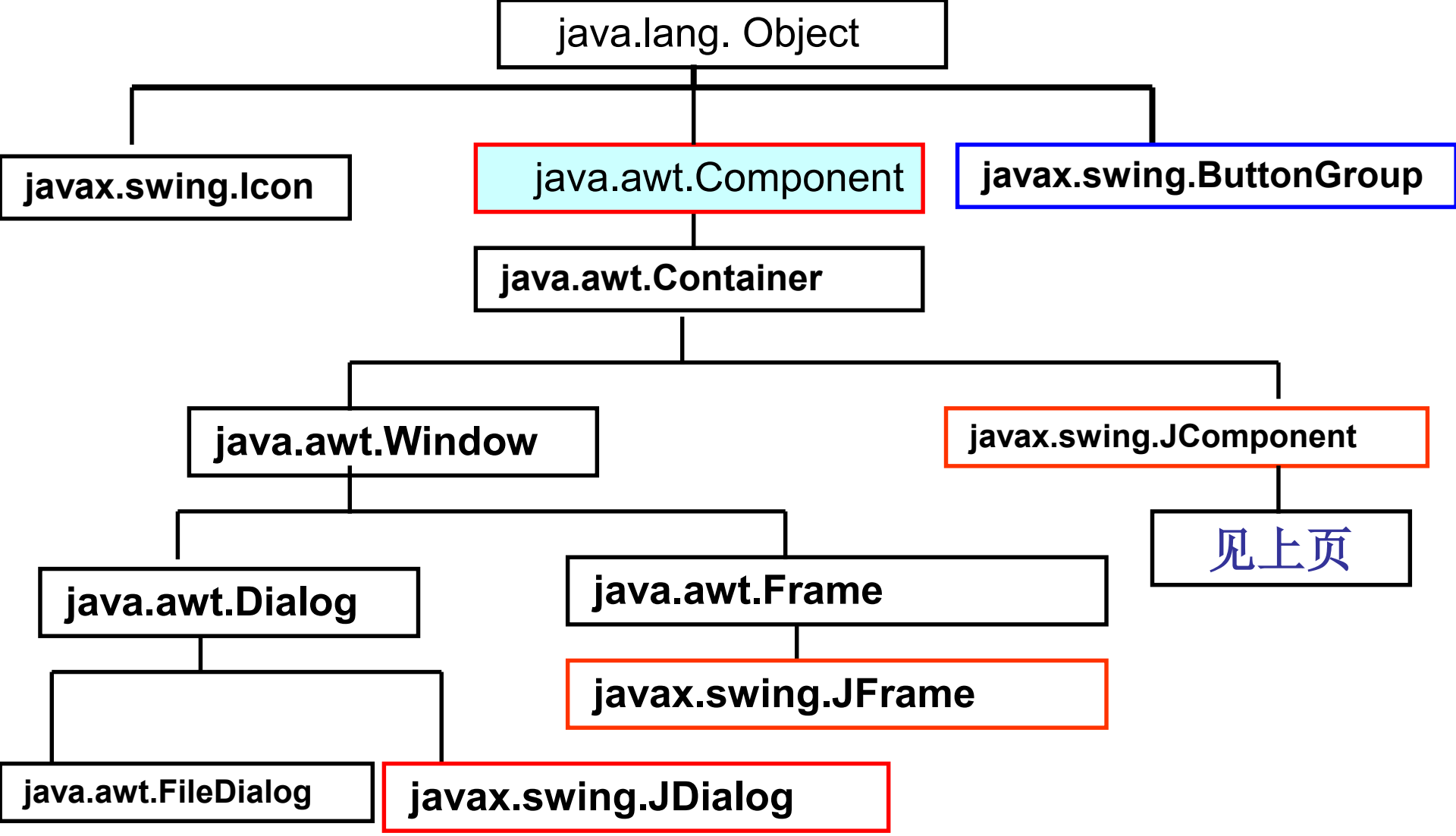
- 6.1 JFC
- 6.2 Swing与AWT
- 6.3 Java 创建GUI步骤
- 6.4 AWT创建GUI
- **6.5 Swing创建GUI**

6.5 Swing创建GUI

- Swing库是AWT库的扩展实现
- 相关的类在javax.swing包中
- 提供了250多个类，40多个组件
- Java 建议用Swing组件代替AWT组件
- Swing中的组件称为轻量组件



Swing组件关系概述（一）



Swing组件关系概述（二）

结论：

- JFrame继承自java.awt.Frame
- JDialog继承自java.awt.Dialog
- JComponent继承自java.awt.Container，所以swing组件都是容器。
- Jcomponent 是所有非窗口顶层容器组件的父类.

```
public abstract class Jcomponent implements Serializable
{ ..........
}
public class JLabel extends Jcomponent implements .....
{.........
}
```

1：窗口组件---关闭操作

- 关闭操作调用方法：

public void setDefaultCloseOperation(int operation)

解释：

设置用户在此窗体上发起 “close” 时默认执行的操作。（见JFrame或者JDialog）

方法来源：

```
public interface WindowConstants
{
    public static final int DO_NOTHING_ON_CLOSE = 0;
                                                    //什么也不做

    public static final int HIDE_ON_CLOSE = 1; //隐藏窗口

    public static final int DISPOSE_ON_CLOSE = 2;
                            //隐藏当前窗口，释放窗口占用的其他资源

    public static final int EXIT_ON_CLOSE = 3;
                            //结束程序运行
}
```

2:swing创建GUI:

- (1)选择组件:

  ------swing中的组件都是以JXXX开头. swing组件都是容器(javax.swing)

  (2)选择容器:

  -----swing窗体也是容器.

- (3)选择布局管理器:

  -----使用java.awt包中的layout manager

- (4)激发事件:

  -----java.awt.event包中的相应的Event与javax.swing.event互相补充.

  (5)事件监听器:

  -----使用java.awt.event.**Listener和javax.swing.event.**Listnener互相补充

3:swing举例一

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class JDialogDemo extends JFrame implements ActionListener
{
    public JDialogDemo(String s)
    {   super("User Login");
       JButton jb=new JButton("click me");
        JPanel jp=new JPanel();
        jp.add(jb);
        add(jp);
        jb.addActionListener(this);
        this.setSize(400,100);                    //设置框架尺寸
        this.setBackground(java.awt.Color.lightGray);
                                                  //设置框架背景颜色
        this.setLocation(300,240);                //框架显示在屏幕位置
        this.setVisible(true); //显示框架
    }
```

举例(续:)

```
 public void actionPerformed(ActionEvent e)
{
•       JDialog jd=new JDialog(this,"are you sure",false);
•       jd.setSize(100,100);
•       jd.setVisible(true);
•       jd.setDefaultCloseOperation(EXIT_ON_CLOSE);
•                                       //缺省值为HIDE_ON_CLOSE
}
•
 public static void main(String[] args)
{
•       new JDialogDemo();
}
}
```

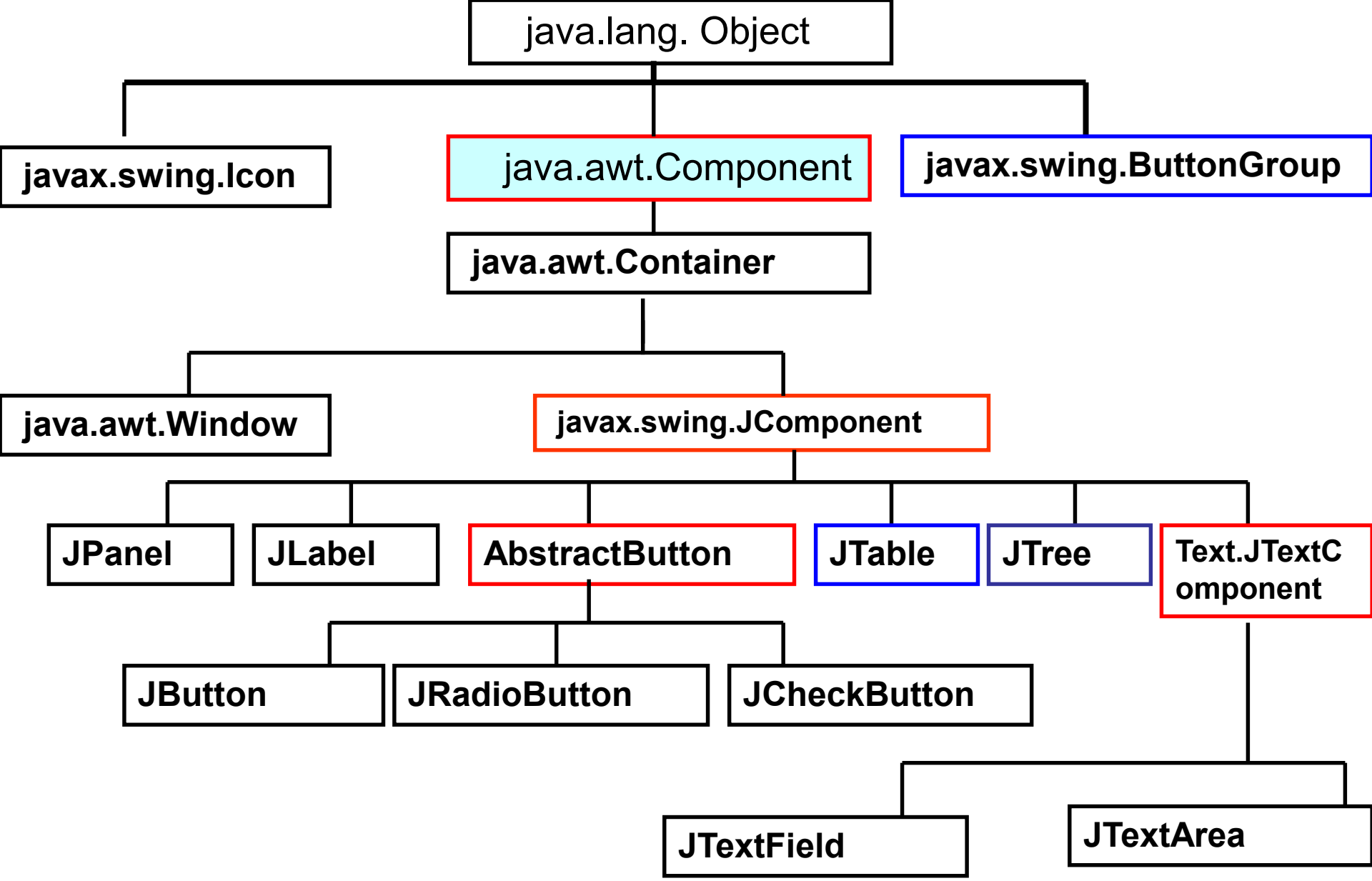
二、带图标的JButton示例:

```
import java.awt.*;
import javax.swing.*;
public class JButtonDemo
{   public JButtonDemo()
    {
                JFrame jf=new JFrame("这是我的一个实例");
                 ImageIcon icon=new ImageIcon("picture.jpg");
                 JButton jb=new JButton("图标按钮",icon);
                 JPanel jp=new JPanel();
                 jp.add(jb);
                 jf.add(jp);

                 jf.setSize(100,100);
                 jf.setVisible(true);
    }
    public static void main(String[] args)
    {  new JButtonDemo();    }
}
```

三:ButtonGroup的使用示例

- 使用ButtonGroup和JRadionButton可以实现一组按钮的单项选择.
- ButtonGroup是javax.swing包中的类,不是组件.所以不是Jcomponent的子类.

```
public class ButtonGroup extends Object implements Serializable
 {   public ButtonGroup()
     public void add(AbstractButton b)
     public void remove(AbstractButton b)
 }
```



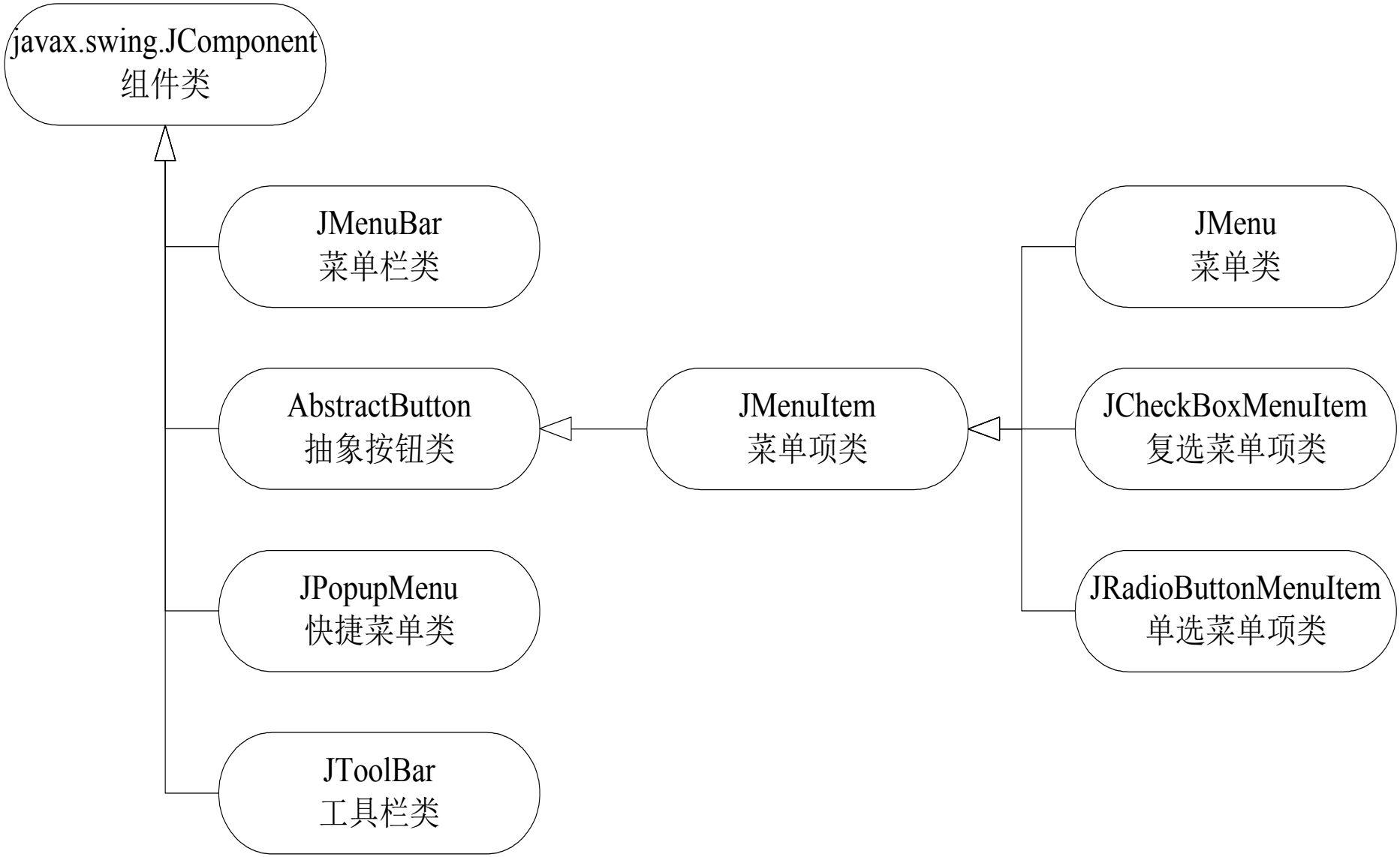
ButtonGroup与Swing关系

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class ButtonGroupAnimals extends JFrame
                        implements ActionListener {
   static String goatString = "goat";
   static String catString = "cat";
   static String dogString = "dog";
   static String parrotString = "parrot";
   static String pigString = "pig";
   JLabel picture;
   public ButtonGroupAnimals()
   {   super("显示动物举例");
      //Create the radio buttons.
      JRadioButton goatButton = new JRadioButton("山羊");
      goatButton.setActionCommand(goatString);
      goatButton.setSelected(true);
      JRadioButton catButton = new JRadioButton("小猫");
      catButton.setActionCommand(catString);
      JRadioButton dogButton = new JRadioButton("小狗");
      dogButton.setActionCommand(dogString);
      JRadioButton parrotButton = new JRadioButton("鹦鹉");
      parrotButton.setActionCommand(parrotString);
      JRadioButton pigButton = new JRadioButton("小猪");
      pigButton.setActionCommand(pigString);
      ButtonGroup group = new ButtonGroup();
      group.add(goatButton);
      group.add(catButton);
      group.add(dogButton);
      group.add(parrotButton);
      group.add(pigButton);
```

```
        goatButton.addActionListener(this);
        catButton.addActionListener(this);
        dogButton.addActionListener(this);
        parrotButton.addActionListener(this);
        pigButton.addActionListener(this);
       ImageIcon icon=new ImageIcon("images/"+ goatString+".jpg");
        picture = new JLabel(icon);
      JPanel radioPanel = new JPanel(new GridLayout(0, 1));
        radioPanel.add(goatButton);
        radioPanel.add(catButton);
        radioPanel.add(dogButton);
        radioPanel.add(parrotButton);
        radioPanel.add(pigButton);
        add(radioPanel, BorderLayout.LINE_START);
        add(picture, BorderLayout.CENTER);
        this.setSize(600,600);
        this.setVisible(true);
    }
```

```
 public void actionPerformed(ActionEvent e) {
            ImageIcon icon=new ImageIcon("images/"+
e.getActionCommand()+".jpg");
        picture.setIcon(icon);
    }




    public static void main(String[] args) {
            new ButtonGroupAnimals();

            }


}
```

四:JMenu示例


javax.swing.JComponent
组件类
JMenuBar
菜单栏类
AbstractButton
抽象按钮类
JPopupMenu
快捷菜单类
JToolBar
工具栏类
JMenuItem
菜单项类
JMenu
菜单类
JCheckBoxMenuItem
复选菜单项类
JRadioButtonMenuItem
单选菜单项类

引申:Java 类库对设计模式的应用

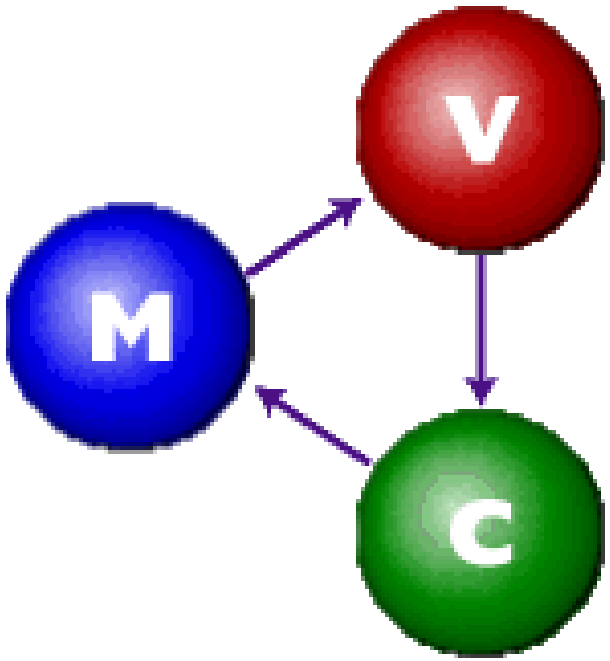
- **Swing中的Action是对设计模式的Command设计模式最良好的应用**

Swing高级组件使用：

- JTable的使用

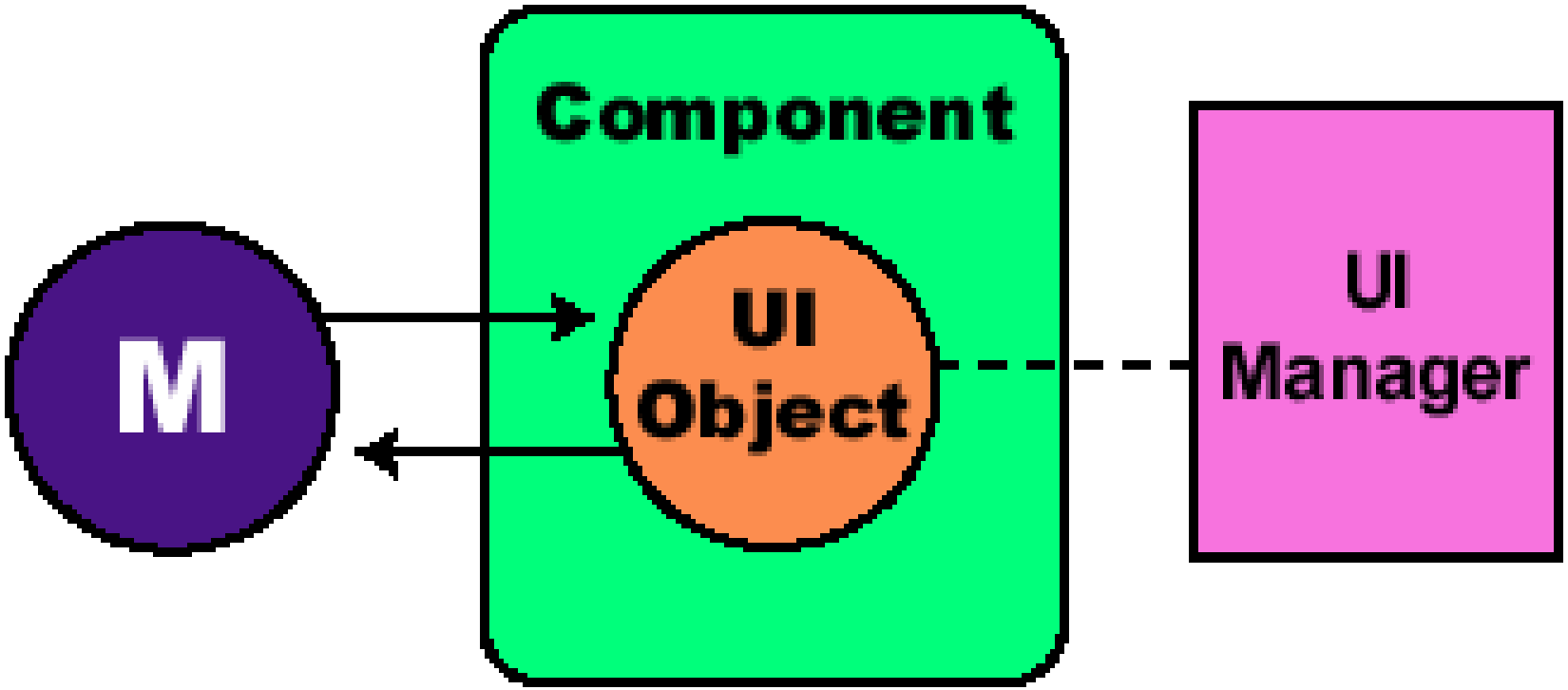
JTable组件的使用

- A　MVC



- **A *model* that represents the data for the application.**
- **The *view* that is the visual representation of that data.**
- **A *controller* that takes user input on the view and translates that to changes in the model.**

B *Swing's quasi-MVC*



类比：机关枪

The following table shows the component-to-model mapping for Swing

Component	Model Interface	Model Type
JButton	ButtonModel	GUI
JToggleButton	ButtonModel	GUI/data
JCheckBox	ButtonModel	GUI/data
JRadioButton	ButtonModel	GUI/data
JMenu	ButtonModel	GUI
JMenuItem	ButtonModel	GUI
JCheckBoxMenuItem	ButtonModel	GUI/data
JRadioButtonMenuItem	ButtonModel	GUI/data
JComboBox	ComboBoxModel	data
JProgressBar	BoundedRangeModel	GUI/data
JScrollBar	BoundedRangeModel	GUI/data
JSlider	BoundedRangeModel	GUI/data
JTabbedPane	SingleSelectionModel	GUI
JList	ListModel	data
JList	ListSelectionModel	GUI
JTable	TableModel	data
JTable	TableColumnModel	GUI
JTree	TreeModel	data
JTree	TreeSelectionModel	GUI
JEditorPane	Document	data
JTextPane	Document	data
JTextArea	Document	data
JTextField	Document	data
JPasswordField	Document	data

B：JTable组件

- ***JTable ---- View***

  JTable 用来显示和编辑规则的二维单元表

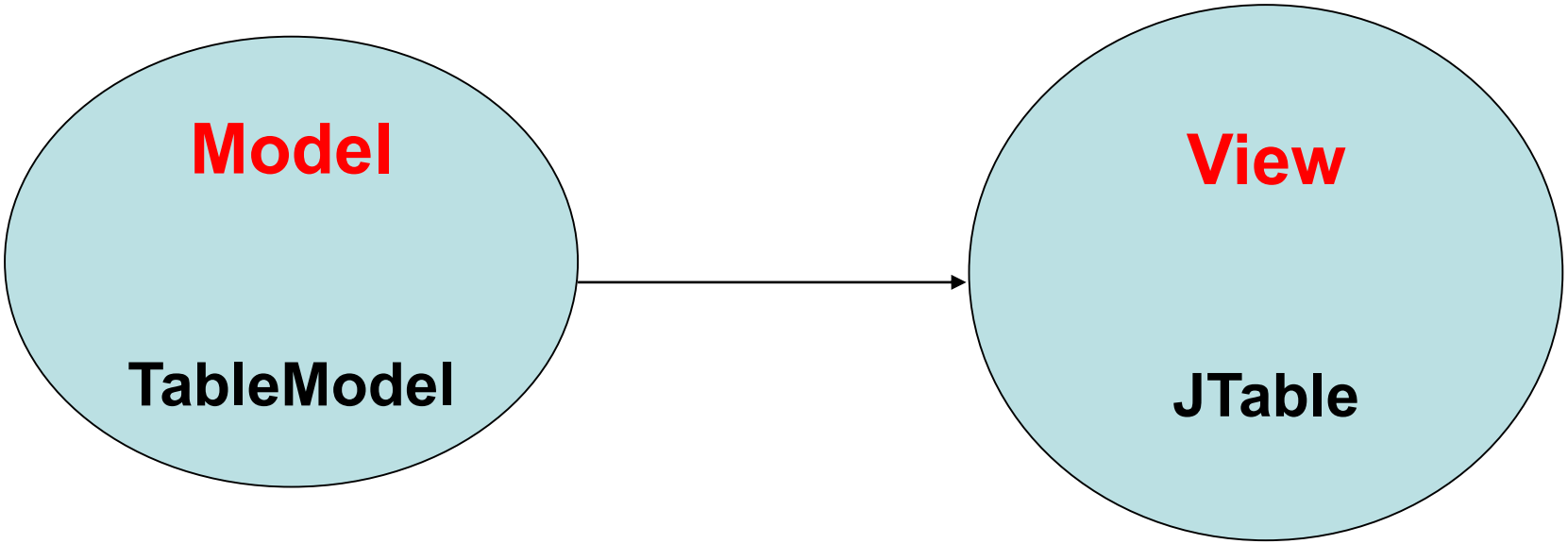
  JTable不包含也不缓存数据,只是数据的一个视图

- ***TableModel----Model***

  每一个JTable都有一个TableModel,将数据放到TableModel进行组织，然后再创建JTable显示数据。

  JTable是从一个实现了TableModel接口的类中获得数据

JTable工作原理:



【演示】txt带制表位文本复制到Excel

TableModel:

- Java提供了两个实现了TableModel的类

  DefaultTableModel

  AbstractTableModel

注:

- 位于javax.swing.table包中.
- AbstractTableModel是一个抽象类,使用时必须继承创建一个自定义的TableModel类.