



哈爾濱工業大學(深圳)
HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

規格嚴格 功夫到家
1920 — 2017

面向对象的软件构造导论

第九章 Swing图形用户界面

计算机科学与技术学院
哈尔滨工业大学（深圳）



课程回顾 (第八章)

- 流
- 输入输出流
- Java流继承框架
- 操作文件
- 对象输入/输出流与序列化
- 数据访问对象模式

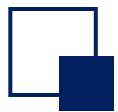


读取控制台输入

- ❑ Java的控制台输入由System.in完成。
- ❑ 为了获得一个绑定到控制台的字符流，你可以把System.in包装在一个BufferedReader对象中来创建一个字符流。
- ❑ 下面是创建BufferedReader的基本语法：

```
BufferedReader br = new BufferedReader(new  
    InputStreamReader(System.in));
```

- ❑ BufferedReader对象创建后，我们便可以使用read()方法从控制台读取一个字符，或者用readLine()方法读取一个字符串。



从控制台读取多字符输入

□ 下面的程序示范了用**read()**方法从控制台不断读取字符直到用户输入q。

```
//使用 BufferedReader 在控制台读取字符
import java.io.*;
public class BRRead {
    public static void main(String[] args) throws IOException {
        char c;
        // 使用 System.in 创建 BufferedReader
        BufferedReader br = new BufferedReader(new
            InputStreamReader(System.in));
        System.out.println("输入字符, 按下 'q' 键退出。");
        // 读取字符
        do {
            c = (char) br.read();
            System.out.println(c);
        } while (c != 'q');
    }
}
```

System.out包含方法:

- **print()**: 不换行
- **println()**: 换行
- **write()**: 很少用



从控制台读取多字符输入

□ 以上实例编译运行结果如下：

```
1.输入字符, 按下 'q' 键退出。  
2.runoob  
3.r  
4.u  
5.n  
6.o  
7.o  
8.b  
9.  
10.  
11.q
```



从控制台读取字符串

- 下面的程序用`readline()`方法读取和显示字符行直到输入单词“end”。

```
//使用 BufferedReader 在控制台读取字符
import java.io.*;
public class BRReadLines {
    public static void main(String[] args) throws IOException {
        // 使用 System.in 创建 BufferedReader
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        String str;
        System.out.println("Enter lines of text.");
        System.out.println("Enter 'end' to quit.");
        do {
            str = br.readLine();
            System.out.println(str);
        } while (!str.equals("end"));
    }
}
```



从控制台读取字符串

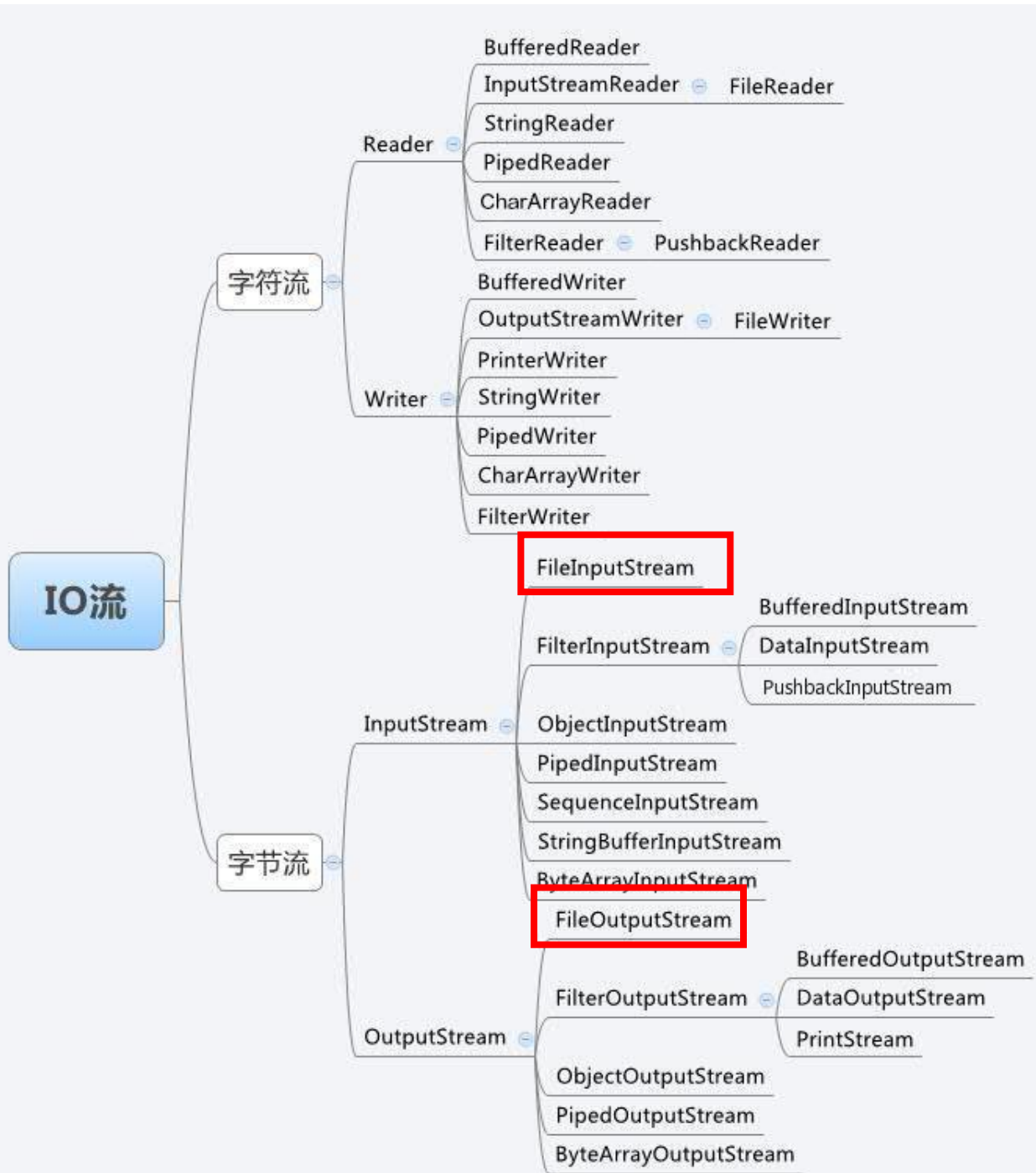
□ 以上示例编译运行结果如下：

```
1.Enter lines of text.  
2.Enter 'end' to quit.  
3.This is line one  
4.This is line one  
5.This is line two  
6.This is line two  
7.end
```



文件的输入输出

下面将要讨论的两个重要的流是 **FileInputStream** 和 **FileOutputStream**。





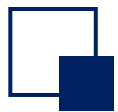
文件输入输出实例

```
1.import java.io.*;
2.public class FileStreamTest {
3.    public static void main(String[] args) throws IOException {
4.        File f = new File("a.txt");
5.        FileOutputStream fop = new FileOutputStream(f);
6.        // 构建FileOutputStream对象,文件不存在会自动新建
7.        OutputStreamWriter writer = new OutputStreamWriter(fop, "UTF-8");
8.        // 构建OutputStreamWriter对象,参数可以指定编码,默认为操作系统默认编码
9.        writer.append("中文输入");
10.       // 写入到缓冲区
11.       writer.append("\r\n");
12.       // 换行
13.       writer.append("English");
19.       writer.close();
20.       // 关闭写入流,同时会把缓冲区内容写入文件,所以上面的注释掉
21.       fop.close();
22.       // 关闭输出流,释放系统资源
```



文件输入输出实例

```
19.      FileInputStream fip = new FileInputStream(f);
20.      // 构建FileInputStream对象
21.      InputStreamReader reader = new InputStreamReader(fip, "UTF-8");
22.      // 构建InputStreamReader对象,编码与写入相同
23.
24.      StringBuffer sb = new StringBuffer();
25.      while (reader.ready()) {
26.          sb.append((char) reader.read());
27.          // 转成char加到StringBuffer对象中
28.      }
35.      System.out.println(sb.toString());
36.      reader.close();
37.      // 关闭读取流
38.      fip.close();
39.      // 关闭输入流,释放系统资源
40.  }
41. }
```



完整的流家族

- Java所有的流类位于java.io包中，都分别继承自以下四种抽象流类型(四大家族)

| | 字节流 | 字符流 |
|-----|--------------|--------|
| 输入流 | InputStream | Reader |
| 输出流 | OutputStream | Writer |



读写文件（读）

- ❑ **Files**是操作文件的工具类,包含了大量的方法
- ❑ 可以用下面的方式很容易地读取文件的所有内容:
 - **byte**[] bytes = Files.readAllBytes(path);
- ❑ 我们还可以以如下的方式从文本文件中读取内容:
 - var content = Files.readString(path, charset), 例如:
- ❑ 但是如果希望将文件当作行序列读入, 那么可以调用:
 - List<String> lines = Files.readAllLines(path, charset);

```
String content = Files.readString(Paths.get("/path/to/file.txt"), StandardCharsets.ISO_8859_1);
```

Charset表示字符集, 默认为UTF-8



读写文件（写）

- ❑ 如果希望写出一个字符串到文件中，可以调用：
 - `Files.writeString(path, content.charset);`
- ❑ 向指定文件追加内容，可以调用：
 - `Files.write(path, content.getBytes(charset), StandardOpenOption.APPEND);`
- ❑ 还可以用下面的语句将一个行的集合写出到文件中：
 - `Files.write(path, lines, charset);`

```
// 写入二进制文件:
```

```
byte[] data = ...
```

```
Files.write(Paths.get("/path/to/file.txt"), data);
```

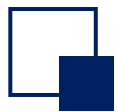
```
// 写入文本并指定编码:
```

```
Files.writeString(Paths.get("/path/to/file.txt"), "文本内容...", StandardCharsets.ISO_8859_1);
```

```
// 按行写入文本:
```

```
List<String> lines = ...
```

```
Files.write(Paths.get("/path/to/file.txt"), lines);
```



创建文件和目录

□ 创建新目录可以调用:

- `Files.createDirectory(path);`
- 其中，路径中除了最后一个部件外，其他部分都必须是已存在的。

□ 可以使用下面的语句创建一个空文件:

- `Files.createFile(path);`
- 如果文件已经存在了，那么这个调用就会抛出异常。



复制、移动和删除文件

- ❑ 将文件从一个位置**复制**到另一个位置可以直接调用：
 - `Files.copy(fromPath, toPath);`
- ❑ **移动**文件(即复制并删除原文件)可以调用：
 - `Files.move(fromPath, toPath);`
- ❑ 如果目标路径已经存在，那么复制或移动将失败。如果想要覆盖已有的目标路径，可以使用**REPLACE_EXISTING**选项。如果想要复制所有的文件属性，可以使用**COPY_ATTRIBUTES**选项。也可以像下面这样同时选择这两个属性：
 - `Files.copy(fromPath, toPath, standardCopyOption.REPLACE_EXISTING,
standardCopyOption.COPY_ATTRIBUTES);`



复制、移动和删除文件

- ❑ 可以将移动操作定义为**原子性**的，这样就可以保证要么移动操作成功完成，要么源文件继续保持在原来位置。具体可以使用 **ATOMIC_MOVE** 选项来实现：
 - `Files.move(fromPath, toPath, standardCopyOption.ATOMIC_MOVE);`
- ❑ 最后，**删除文件**可以调用：
 - `Files.delete(path);`
- ❑ 如果要删除的文件不存在，这个方法就会抛出异常。因此，可转而使用下面的方法：
 - `boolean deleted = Files.deleteIfExists(path);`
 - 该删除方法还可以用来移除空目录。



对象输入输出与序列化

- Java语言支持一种称为**对象序列化(object serialization)**的通用机制，它可以将任何对象写出到输出流中，并在之后将其读回。
- 序列化与反序列化
 - 把对象转换为字节序列的过程称为对象的序列化(**serialize**)。
 - 把字节序列恢复为对象的过程称为对象的反序列化(**deserialize**)。



序列化案例

□ 定义了如下的Person类，该类实现了Serializable 接口

```
public class Person implements Serializable {  
    private String name;  
    private int age;  
    public Person(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
    //为了方便查看对象, 重写默认的toString()  
    @Override  
    public String toString(){  
        return "Person{" + "name=" + name + "\" +  ", age=" + age + "}";  
    }  
}
```



序列化案例

□ 调用 `ObjectOutputStream` 对象的 `writeObject` 输出可序列化对象

```
public class SeriDemo {  
    public static void main(String[] args) {  
        // ObjectOutputStream 流  
        try {  
            Person p1 = new Person("zhangsan", 30);  
            System.out.println(p1);  
            ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("person.dat"));  
            oos.writeObject(p1);  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```



序列化案例

□调用ObjectInputStream对象的readObject()得到序列化的对象

```
//创建一个ObjectInputStream输入流
try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream("person.dat"))){
    System.out.println("readObject(): ");
    Person zhangsan = (Person) ois.readObject();
    System.out.println(zhangsan);
} catch (Exception e) {
    e.printStackTrace();
}
```

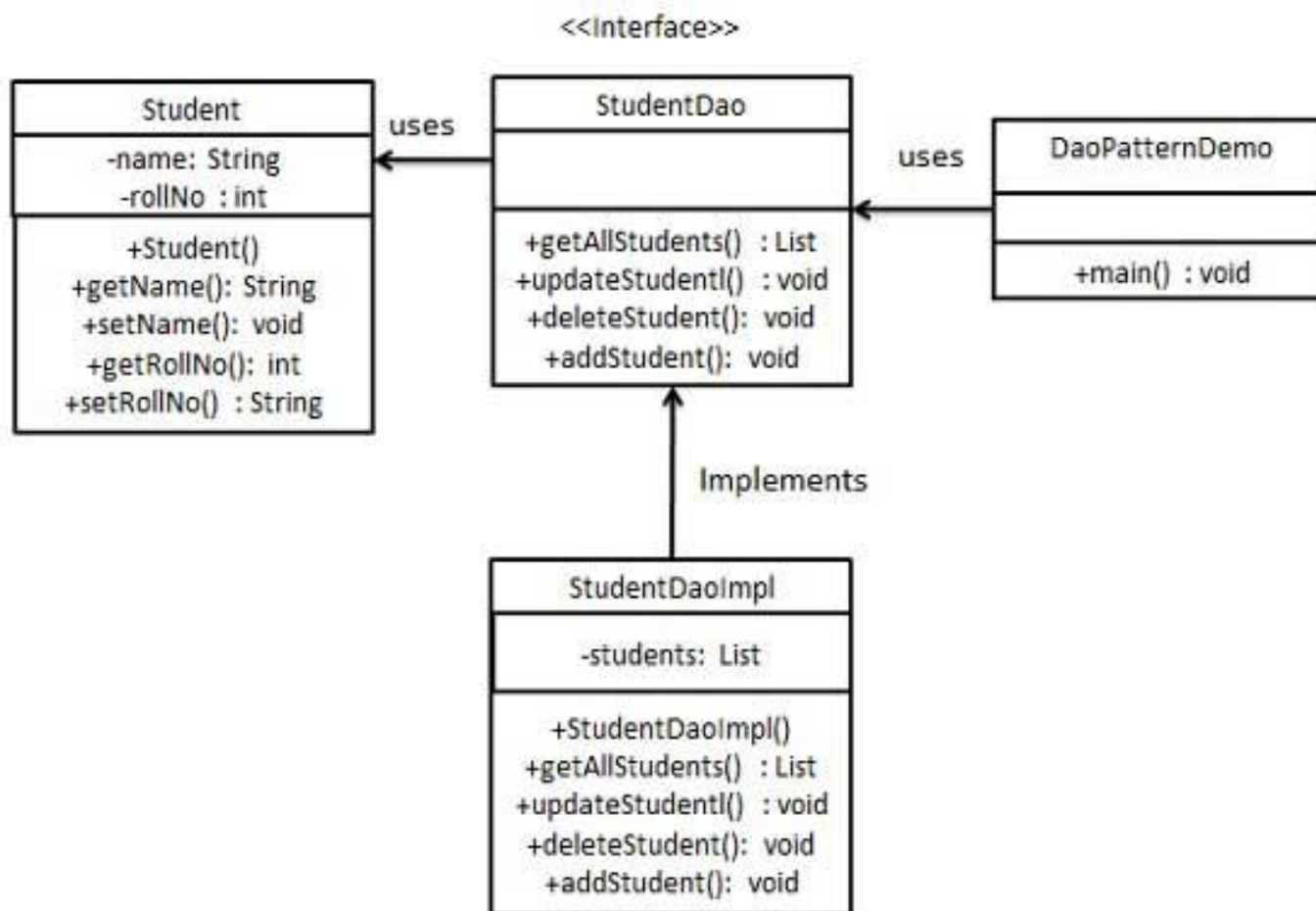


数据访问对象模式实现

❑ 我们将创建一个作为模型对象或数值对象的 **Student** 对象。**StudentDao** 是数据访问对象接口。**StudentDaoImpl** 是实现了数据访问对象接口的实体类。

❑ **DaoPatternDer** 模式的用法。

≡ 数据访问对象





課程內容

- Swing框架
- Swing图形处理、绘制颜色的原理
- 事件机制
- Swing基本用户组件
- MVC模式

JAVA GUI (Graphical User Interface) 简史-AWT

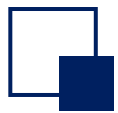
■ 在Java1.0刚刚出现的时候，包含了一个用于基本GUI程序设计的类库，名为抽象窗口工具(Abstract Window Toolkit, AWT).

■ 问题：

- AWT的初衷是用来开发小型的图形界面程序，没有丰富的用户界面组件集合
- 不同平台上的AWT用户界面库中存在着不同的bug

AWT → **Swing**

■ Swing是用于 Java GUI 编程（图形界面设计）的工具包（类库），采用纯 Java 实现，不再依赖于本地平台的图形界面。



JAVA GUI 简史 – Swing

- ❑ Sun公司和Netscape创建了一个名为Swing的用户界面库。
- ❑ Swing作为Java1.1的一个扩展，现已成为Java1.2标准库的一部分。
- ❑ Swing是用于 Java GUI 编程（图形界面设计）的工具包（类库）。
 - Swing并不是完全替代AWT，而是构建在AWT架构之上。
 - Swing提供了更加强大的用户界面组件，如绘制的用户界面类。
 - AWT提供了窗口工具包的底层机制，如事件处理。
 - Swing 使用纯粹的 Java 代码来模拟各种控件，没有使用本地操作系统的内在方法，所以 Swing 是跨平台的



JAVA GUI 简史—JavaFX

- ❑ Swing最早发布时，用户曾抱怨它的速度太慢。
- ❑ 而随着计算机变得更快，用户开始抱怨Swing太丑了。
- ❑ 2007年，引入了JavaFX. 它在Java虚拟机上运行，专门为实现动画和华丽效果做了优化。
- ❑ 从Java11开始，JavaFX将不再打包到Java中。

Swing框架

- ❑ Swing GUI包含了两种元素：**组件和容器**。
- ❑ 一个 Java 的图形界面，由各种不同类型的“元素”组成，例如：窗口、菜单栏、对话框、标签、按钮、文本框等等，这些“元素”统一被称为**组件**
- ❑ 组件按照不同的功能，可分为**顶层容器、中间容器、基本组件**。
- ❑ 组件和容器构成了包含层级关系。

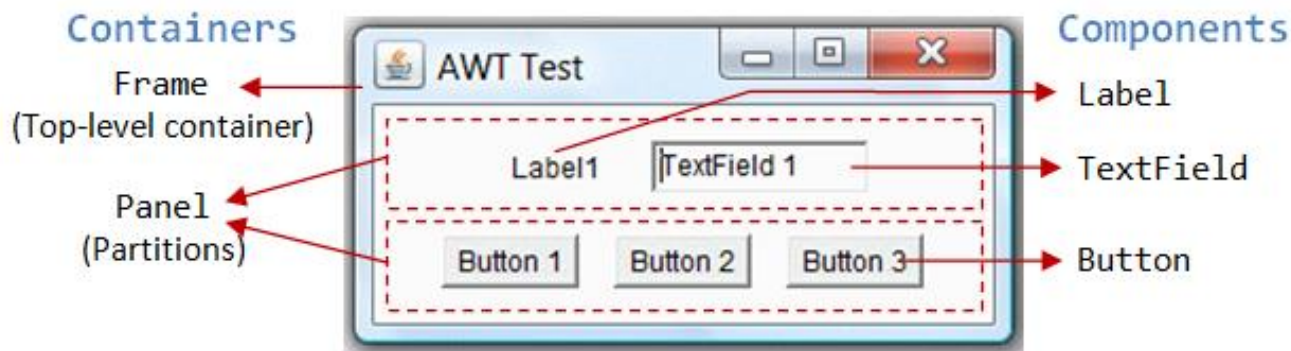
- 顶层容器

- 菜单栏

- 中间容器

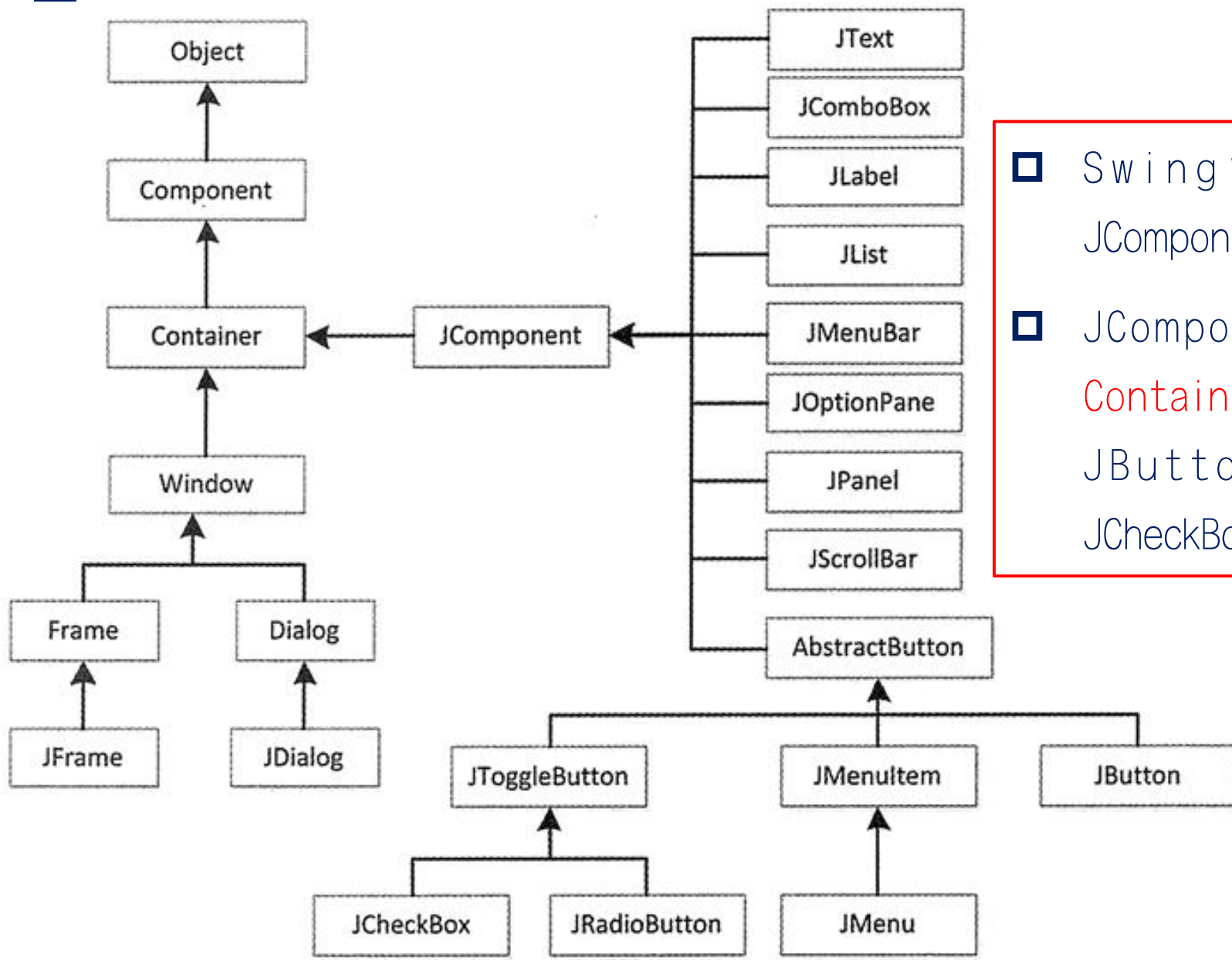
- 基本组件

- 基本组件

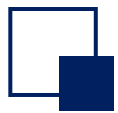




Swing框架



- Swing的组件继承于JComponent类，JComponent类提供了所有组件都需要的功能。
- JComponent继承于类Component及其子类Container。常见的组件有标签JLabel、按钮JButton、输入框JTextField、复选框JCheckBox、列表JList。

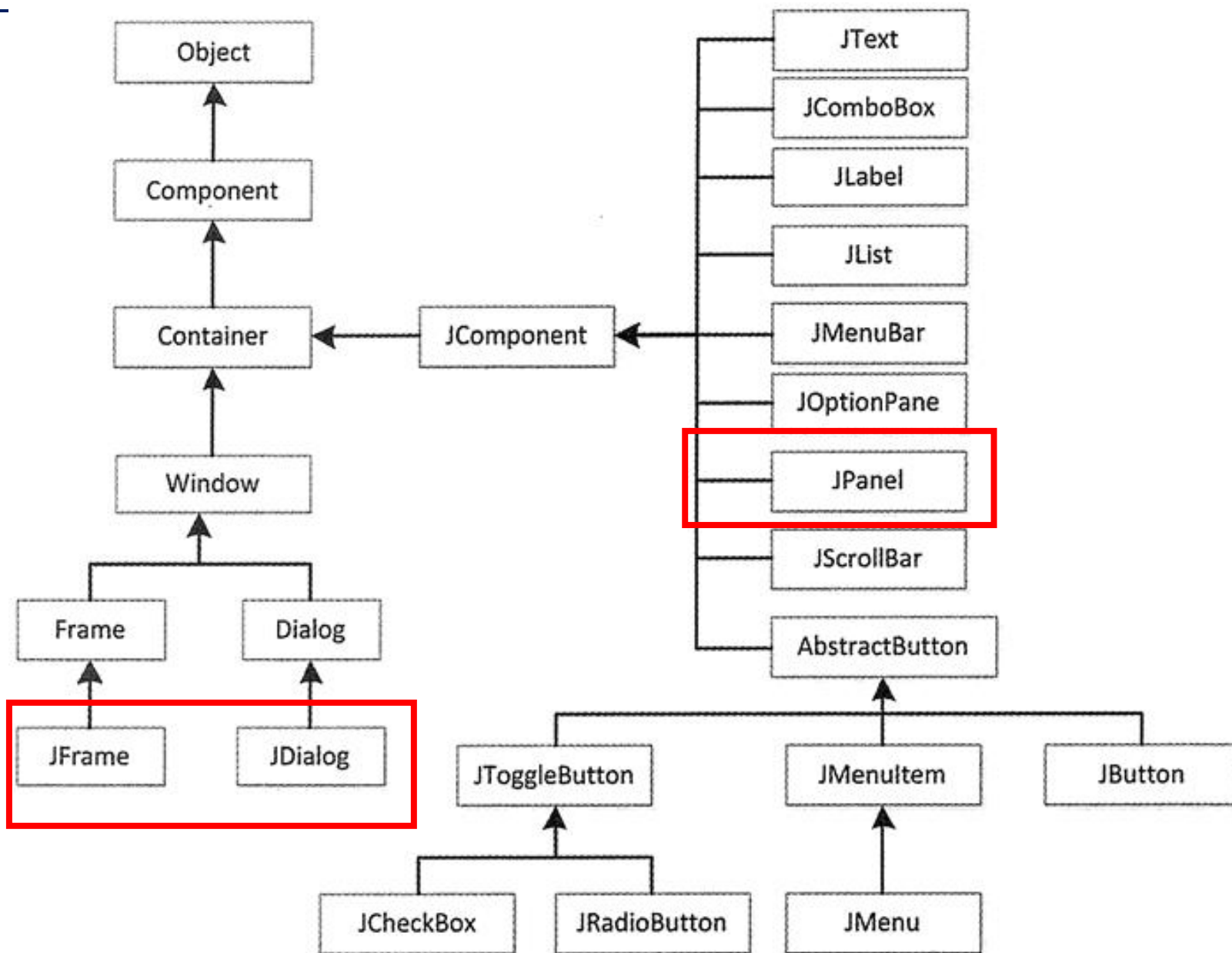


Swing框架

- 容器是一种可以包含组件的特殊组件。Swing中有两大类容器。
 - 一类是**重量级容器**，或者称为顶层容器，它们不继承于JComponent，包括JFrame，JApplet，JDialog. 它们的最大特点是不能被别的容器包含，只能作为界面程序的最顶层容器来包含其它组件。
 - 第二类容器是**轻量级容器**，或者称为中间层容器，它们继承于JComponent，包括JPanel，JScrollPane等。中间层容器用来将若干个相关联的组件放在一起。由于中间层容器继承于JComponent，因此它们本身也是组件，它们可以（也必须）包含在其它的容器中。



Swing框架





Swing框架

- **布局管理器**控制着容器中组件的位置。当向容器中增加组件时，需要给容器设置一种布局管理器，让它来管理容器中各个组件的位置，即排列布局方式。
- 如果不使用布局管理器，则需要先画好各个组件的位置并计算组件间的距离，再向容器中添加。

```
JPanel panel = new JPanel();
panel.setLayout(new FlowLayout());

panel.add(new JButton("按钮1"));
panel.add(new JButton("按钮2"));
panel.add(new JButton("按钮3"));
```

```
JPanel panel = new JPanel();
panel.setLayout(null); // 禁用布局管理器

JButton b1 = new JButton("按钮1");
b1.setBounds(50, 30, 80, 30); // 手动设置位置与大小

panel.add(b1);
```



Swing框架

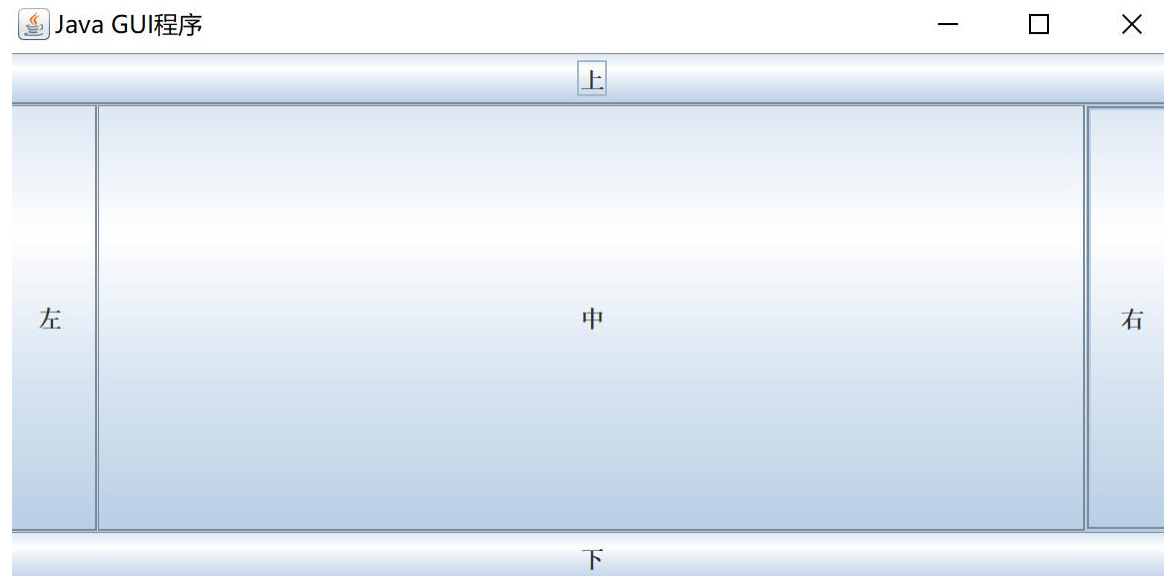
- Java提供了若干种布局管理器。

| 布局管理器 | 特性 |
|---------------|----------------------------------|
| FlowLayout | 流式布局管理器，是从左到右，中间放置，一行放不下就换到另外一行。 |
| BorderLayout | 边框布局管理器分为东、南、西、北、中心五个方位。 |
| GridLayout | 网格式布局。 |
| GridBagLayout | 网格式布局，可以放置不同大小的组件。 |
| BoxLayout | 盒布局管理器，把组件水平或者竖直排在一起。 |
| SpringLayout | 按照一定的约束条件来组织组件。 |

Swing框架

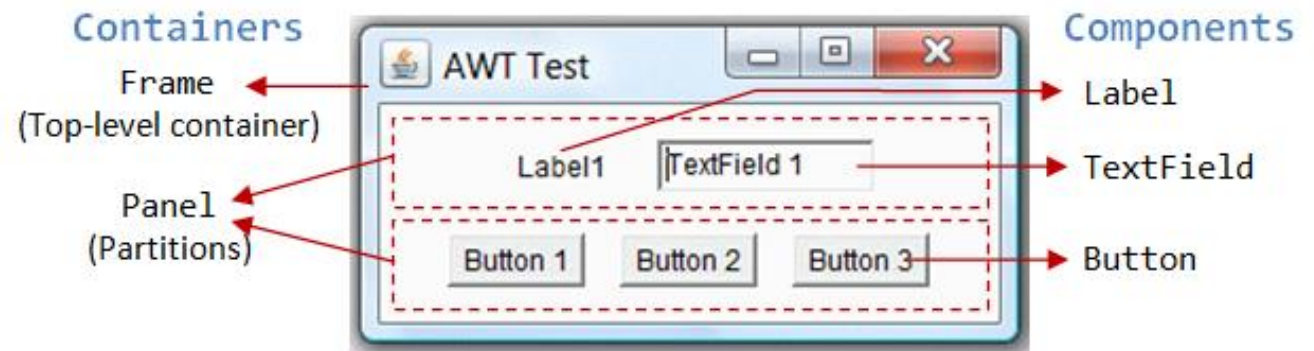
- 例如BorderLayout

```
JFrame frame=new JFrame("Java GUI程序"); //创建Frame窗口
frame.setSize(400,200);
frame.setLayout(new BorderLayout()); //为Frame窗口设置布局为
BorderLayout
JButton button1=new JButton ("上");
JButton button2=new JButton("左");
JButton button3=new JButton("中");
JButton button4=new JButton("右");
JButton button5=new JButton("下");
frame.add(button1,BorderLayout.NORTH);
frame.add(button2,BorderLayout.WEST);
frame.add(button3,BorderLayout.CENTER);
frame.add(button4,BorderLayout.EAST);
frame.add(button5,BorderLayout.SOUTH);
frame.setBounds(300,200,600,300);//覆盖了setSize
frame.setVisible(true);//只有调用后, GUI 才会真正出现在屏幕上。
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);//表示点击右上
角“x”后, 程序会真正退出。
```



是否所有的容器都是组件？

- ☐ A 否
- ☒ B 是
- ☐ C 不确定



提交



課程內容

- Swing框架
- Swing图形处理、绘制颜色的原理
- 事件机制
- Swing基本用户组件
- MVC模式

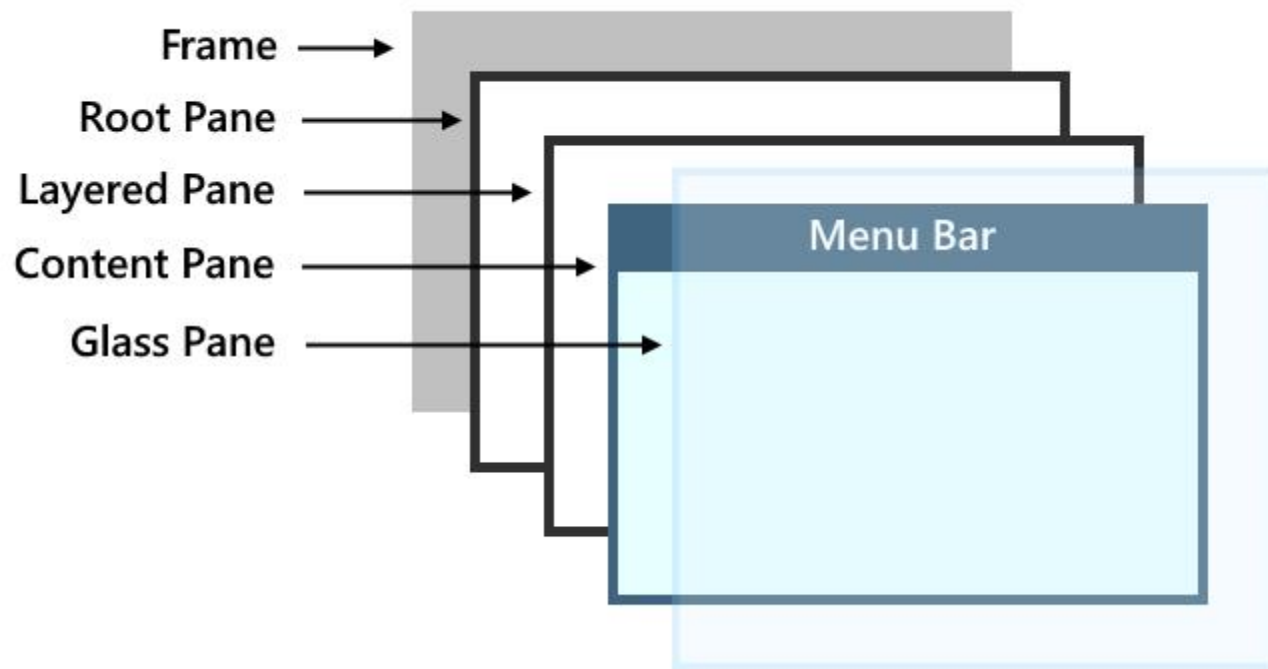


显示窗体

- 顶层窗口（没有包含在其它窗口中的窗口）称为窗体(Frame)。
- Swing中用于描述顶层窗口的类名为JFrame，扩展了AWT中的Frame库
- JFrame是极少数不绘制在画布上的Swing组件之一。它的修饰部件（按钮、标题栏、图标等）由用户的窗口系统绘制。
- 我们只需要关注 **内容窗格**(content pane)。添加到窗体的所有组件都会自动添加到内容窗格中。



显示窗体



- 我们只需要关注内容窗格(content pane)。
- 添加到窗体的所有组件都会自动添加到内容窗格中。



显示窗体

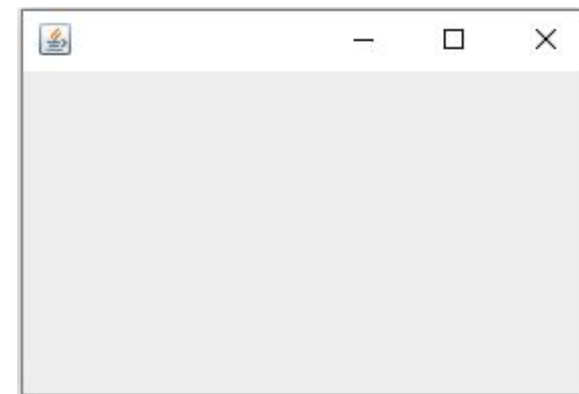
```
1. class SimpleFrame extends JFrame{  
2.     private static final int DEFAULT_WIDTH = 300;  
3.     private static final int DEFAULT_HEIGHT = 200;  
4.  
5.     public SimpleFrame(){  
6.         setSize(DEFAULT_WIDTH,DEFAULT_HEIGHT);  
7.     }  
8. }
```

- 默认情况下，窗体大小为 0×0 像素。



显示窗体

```
1. public class SimpleFrameTest {  
2.     public static void main(String[] args){  
3.         EventQueue.invokeLater(()->  
4.         {  
5.             var frame = new SimpleFrame();  
6.             frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
7.             frame.setVisible(true);  
8.         });  
9.     }  
10. }
```





显示窗体

❑ `EventQueue.invokeLater(()->{statements});`

- 所有Swing组件必须由事件分派线程（event dispatch thread）配置，这是控制线程，它将鼠标点击和按键等事件传递给用户接口组件。

❑ `frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);`

- 定义用户关闭窗体时的相应动作——退出。

❑ `frame.setVisible(true);`

- 窗体起初是不可见的。程序员可以在向内添加了一系列组件后，通过setVisible方法让它显示。



Lambda表达式

- Lambda 表达式是一种匿名函数，它可以使用简洁的语法定义一个函数，而无需显式命名。它常用于替换需要**匿名**内部类或简单函数对象的场景，并且可以捕获其所在作用域的变量。
- 在Java、C# 和C++ 等多种语言中都有实现，通常使用 \rightarrow / \Rightarrow 箭头符号来分隔参数和函数体。
- Java 8 引入Lambda 表达式，主要用来**简化**匿名内部类的写法，例如在集合的遍历和操作中。

```
// 使用 Lambda 表达式计算两个数的和
```

```
MathOperation addition = (a, b) -> a + b;
```

```
// 调用 Lambda 表达式
```

```
int result = addition.operation(5, 3);
```

```
System.out.println("5 + 3 = " + result);
```

```
List<String> names = Arrays.asList("Alice", "Bob", "Charlie");
```

```
// 使用 Lambda 表达式遍历列表
```

```
names.forEach(name -> System.out.println(name));
```



显示窗体中的信息

- 我们想在窗体中显示信息（如字符串“Hello World”）。
- 我们把一个组件添加到窗体中，消息将绘制在这个组件上。
- 在组件上进行绘制：
 - 定义一个扩展JComponent的类
 - 覆盖其中的paintComponent方法。
 - 在paintComponent方法中，设置Graphics对象。Graphics对象包含了绘制图案、图像和文本的方法。Java中，所有的绘制都必须通过Graphics对象完成。



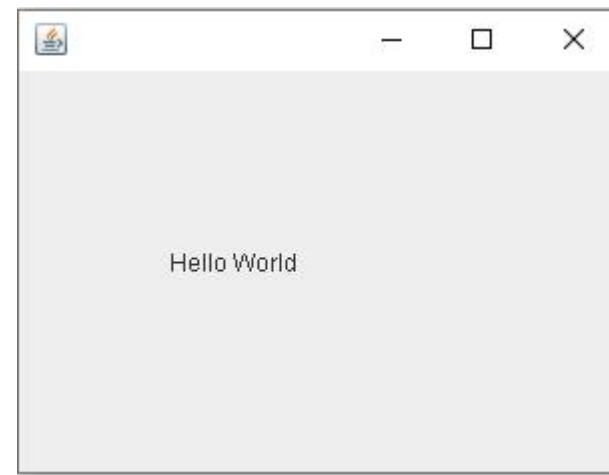
显示窗体中的信息

```
1. class HelloWorldComponent extends JComponent{
2.     private static final int MSG_X = 75;
3.     private static final int MSG_Y = 100;
4.
5.     private static final int DEFAULT_WIDTH = 300;
6.     private static final int DEFAULT_HEIGHT = 200;
7.
8.     public void paintComponent(Graphics g){
9.         g.drawString("Hello World", MSG_X, MSG_Y);
10.    }
11.
12.    public Dimension getPreferredSize(){
13.        return new Dimension(DEFAULT_WIDTH, DEFAULT_HEIGHT);
14.    }
15. }
```



显示窗体中的信息

```
13. class helloworldFrame extends JFrame{  
14.  
15.     public helloworldFrame()  
16.     {  
17.         add(new helloworldComponent());  
18.         pack();  
19.     }  
20. }
```



- ❑ 在Line 10.中，我们设定了组件的大小，返回一个有首选宽度和高度的Dimension类对象。
- ❑ Line 17.中，在窗体中填入组件时，用pack()方法来使用它们的首选大小。



绘制2D图形

- ▣ 获得Graphics2D类的一个对象，使用Java 2D库绘制图形。
- ▣ Java 2D库采用面向对象的方式组织几何图形（直线、矩形、椭圆）：
 - Line2D
 - Rectangle2D
 - Ellipse2D
- ▣ Java 2D库针对像素采用的是浮点坐标。内部计算都采用单精度float，为避免强制类型转换的处理，可以使用Double图形类。



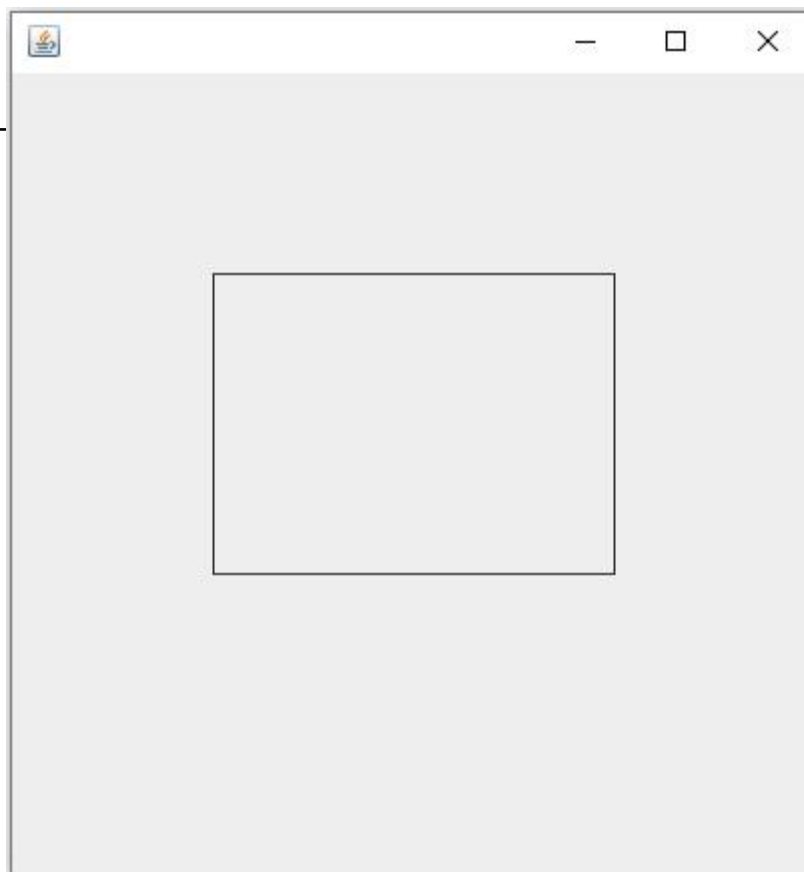
绘制2D图形

```
1. class DrawComponent extends JComponent{
2.     public static final int DEFAULT_WIDTH = 400;
3.     public static final int DEFAULT_HEIGHT = 400;
4.
5.     public void paintComponent(Graphics g){
6.         var g2 = (Graphics2D) g;
7.
8.         double leftX = 100;
9.         double topY = 100;
10.        double width = 200;
11.        double height = 150;
12.        var rect = new Rectangle2D.Double(leftX,topY,width,height);
13.        g2.draw(rect);
14.    }
```



绘制2D图形

```
14. public Dimension getPreferredSize(){  
15.     return new Dimension(DEFAULT_WIDTH,DEFAULT_HEIGHT);  
16. }  
17.  
18. }
```

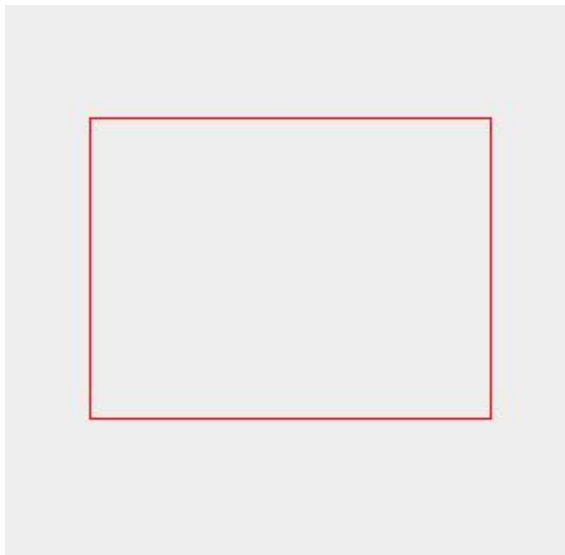




使用颜色

- 使用Graphics2D类的setPaint方法可以为图形上下文的所有后续的绘制操作选择颜色。

```
var rect = new Rectangle2D.Double(leftX,topY,width,height);  
g2.setPaint(Color.RED);  
g2.draw(rect);
```





使用颜色

- 可以用一种颜色填充一个封闭图形的内部。

```
var rect = new Rectangle2D.Double(leftX,topY,width,height);  
g2.setPaint(Color.RED);  
g2.fill(rect);
```



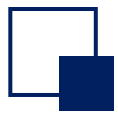


使用颜色

- 想要用多种颜色，就需要选择一个颜色、绘制图形，再选择另外一种颜色、再绘制图形。
- Color类用于定义颜色。在java.awt.Color类中提供了13个预定义的常量，分别表示13种标准颜色。
- 可以提供三色分量来创建Color对象。取值为0~255间的整数。

```
g2.setPaint(new Color(147,112,219));
```





显示图像

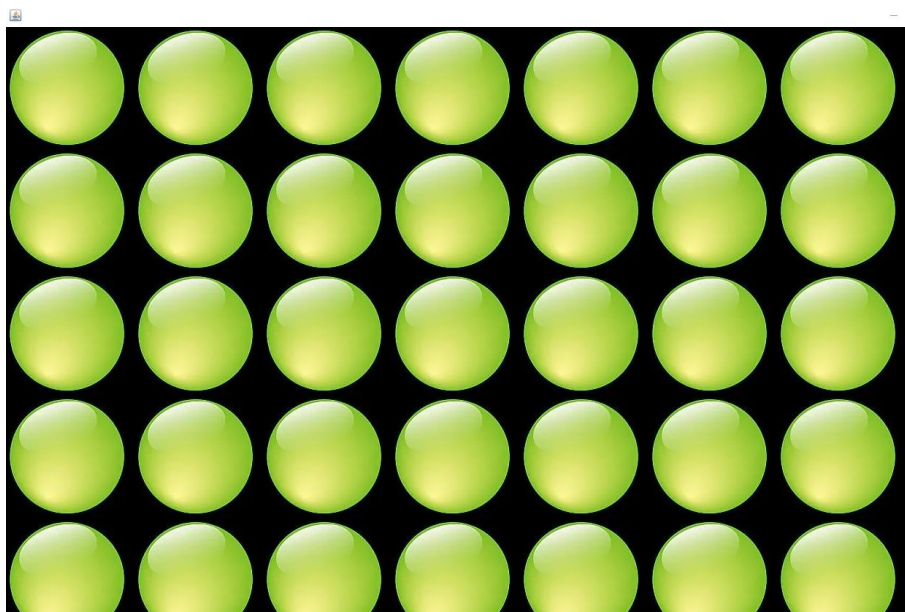
- ❑ 可以使用ImageIcon类从文件读取图像。
- ❑ 变量image包含一个封装了图像数据的对象的引用。可以使用Graphics类的drawImage方法显示这个图像。

```
1.    public void paintComponent(Graphics g)
2.    {
3.        int X = 100;
4.        int Y = 100;
5.        Image image = new ImageIcon(path).getImage();
6.
7.        g.drawImage(image, X, Y, null);
8.    }
```



显示图像

- 再进一步，在一个窗口中平铺显示图像。采用paintComponent的方法来实现。
- 首先在左上角显示图像的一个副本，然后使用copyArea调用将其复制到整个窗口。



[illegible]

上述为什么用 `copyArea()` 而不是重新画？

提交



課程內容

- Swing框架
- Swing图形处理、绘制颜色的原理
- 事件机制
- Swing基本用户组件
- MVC模式



事件

□ 什么是事件？

- 改变对象的状态被称为事件，点击一个按钮，移动鼠标，通过键盘输入一个字符，从列表中选择个项目

| 事件类别 | 触发方式 | 对应事件类 | 举例 |
|--------|----------------|-------------|------------|
| 动作事件 | 点击按钮、菜单项等 | ActionEvent | 点击“确定”按钮 |
| 鼠标事件 | 按下、释放、移动、进入、离开 | MouseEvent | 鼠标移到某个区域 |
| 键盘事件 | 按下、释放某个键 | KeyEvent | 按下 Enter 键 |
| 窗口事件 | 打开、关闭、最小化等 | WindowEvent | 点击关闭窗口按钮 |
| 焦点事件 | 获取或失去输入焦点 | FocusEvent | 光标进入文本框 |
| 列表选择事件 | 选择或取消选择列表项 | ItemEvent | 选择复选框或下拉选项 |



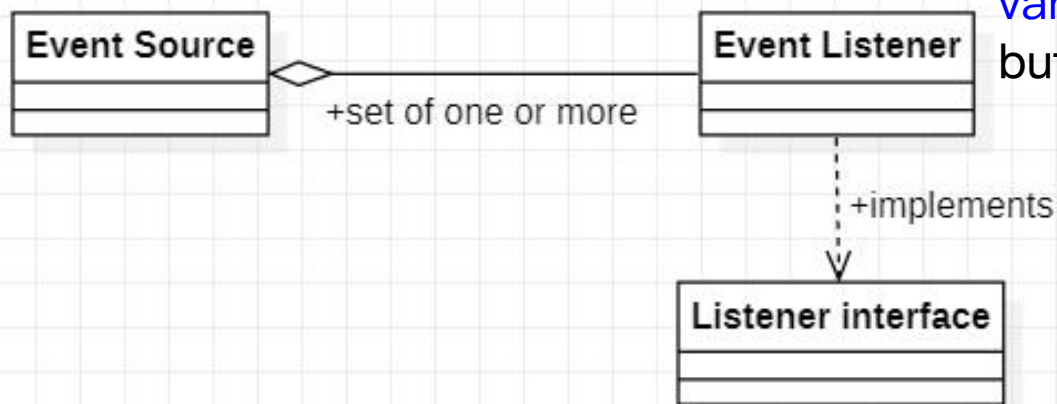
事件机制

- 任何支持GUI的操作环境都要不断地监视按键或点击鼠标这样的事件。这些事件再报告给正在运行的程序。每个程序将决定如何对这些事件做出响应。
- 事件处理机制（三类对象）
 - 事件（Event）：用户对组件的一次操作称为一个事件
 - 事件源（Event Source）：事件发生的场所，通常就是各个组件如按钮或滚动条。
 - 事件监听器：实现了监听器接口(listener interface)的类实例。



事件机制

- 事件源对象能够注册监听器对象并向其发送事件对象。
- 当事件发生时，事件源将事件对象发送给所有注册的监听器。
- 监听器对象再使用事件对象中的信息决定如何对事件做出响应。



```
var button = new JButton("OK");
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // 使用事件对象的信息
        System.out.println("事件源: " + e.getSource());
        System.out.println("命令: " + e.getActionCommand());
        System.out.println("按钮被点击了! ");
    }
})
```



事件机制

```
public class ButtonTest {  
    public static void main(String[] args) {  
        // 创建按钮  
        JButton button = new JButton("OK");  
        // 创建监听器对象 (实现 ActionListener 接口)  
        ActionListener listener = new ActionListener() {  
            public void actionPerformed(ActionEvent e) {  
                System.out.println("按钮被点击了! ");  
            }  
        };  
        // 注册监听器  
        button.addActionListener(listener);  
        // 创建窗口并添加按钮  
        JFrame frame = new JFrame("事件示例");  
        frame.add(button);  
        frame.setSize(200, 150);  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        frame.setVisible(true);  
    }  
}
```

- 只要按钮被点击（产生了“动作事件”），listener对象就会得到通知
- Swing 框架创建一个 ActionEvent 对象
- 事件源（button）把该事件发送给注册的监听器
- 调用 `actionPerformed(ActionEvent e)` 方法
- 执行方法体内的代码（打印“按钮被点击了！”）



事件机制

```
1. class MyListener implements ActionListener
2. {
3.     public void actionPerformed(ActionEvent event)
4.     {
5.         //reaction to button click goes here
6.     }
7. }
```

- ❑ 用户点击按钮，Jbutton对象就会创建一个ActionEvent对象
- ❑ 然后调用listener.actionPerformed(event),并传入这个事件对象。
- ❑ 一个事件源可以有多个监听器。



实例：按钮点击

```
1. public class ButtonFrame extends JFrame
2. {
3.     private JPanel buttonPanel;
4.     private static final int DEFAULT_WIDTH = 300;
5.     private static final int DEFAULT_HEIGHT = 200;
6.
7.     public ButtonFrame()
8.     {
9.         setSize(DEFAULT_WIDTH,DEFAULT_HEIGHT);
10.
11.         var YellowButton = new JButton("Yellow");
12.         var BlueButton = new JButton("Blue");
13.         var RedButton = new JButton("Red");
14.
15.         buttonPanel = new JPanel();
```



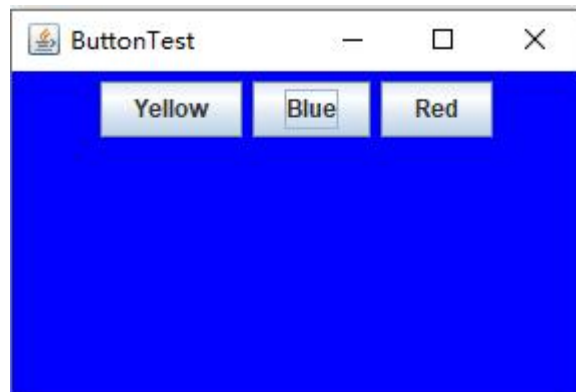
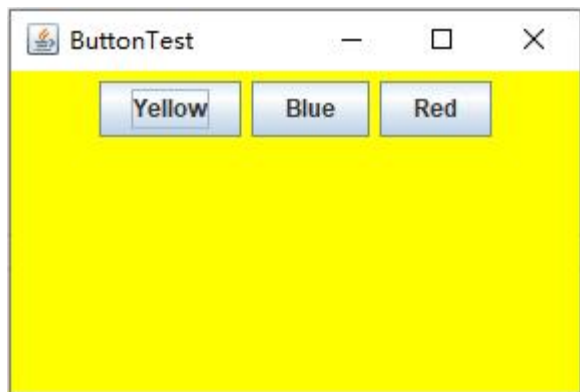
实例：按钮点击

```
13.      buttonPanel.add(YellowButton);
14.      buttonPanel.add(BlueButton);
15.      buttonPanel.add(RedButton);
16.
        add(buttonPanel);
17.
        var YellowAction = new ColorAction(Color.YELLOW);
18.      var BlueAction = new ColorAction(Color.BLUE);
19.      var RedAction = new ColorAction(Color.RED);
20.
        YellowButton.addActionListener(YellowAction);
21.      BlueButton.addActionListener(BlueAction);
22.      RedButton.addActionListener(RedAction);
23.  }
```



实例：按钮点击

```
25. private class ColorAction implements ActionListener{  
26.     private Color backgroundColor;  
27.  
    public ColorAction(Color C){  
28.         backgroundColor = C;  
29.     }  
  
30.     public void actionPerformed(ActionEvent event){  
31.         buttonPanel.setBackground(backgroundColor);  
32.     }  
33. }
```





简洁地指定监听器

- 一般情况下，每个监听器执行一个单独的动作。

```
exitButton.addActionListener(event->System.exit(0));
```

- 有多个相互关联的动作，可以实现一个辅助方法（以颜色按钮为例）。

```
public void makeButton(String name, Color backgroundColor)
{
    var button = new JButton(name);
    buttonPanel.add(button);
    button.addActionListener(event->
        buttonPanel.setBackground(backgroundColor));
}
```



简洁地指定监听器

- 有多个相互关联的动作，可以实现一个辅助方法,从而改进上例。

```
public BottonFrame2()
{
    setSize(DEFAULT_WIDTH,DEFAULT_HEIGHT);

    buttonPanel = new JPanel();
    add(buttonPanel);

    makeButton("yellow", Color.YELLOW);
    makeButton("blue", Color.BLUE);
    makeButton("red", Color.RED);
    makeButton("green", Color.Green);
}
```





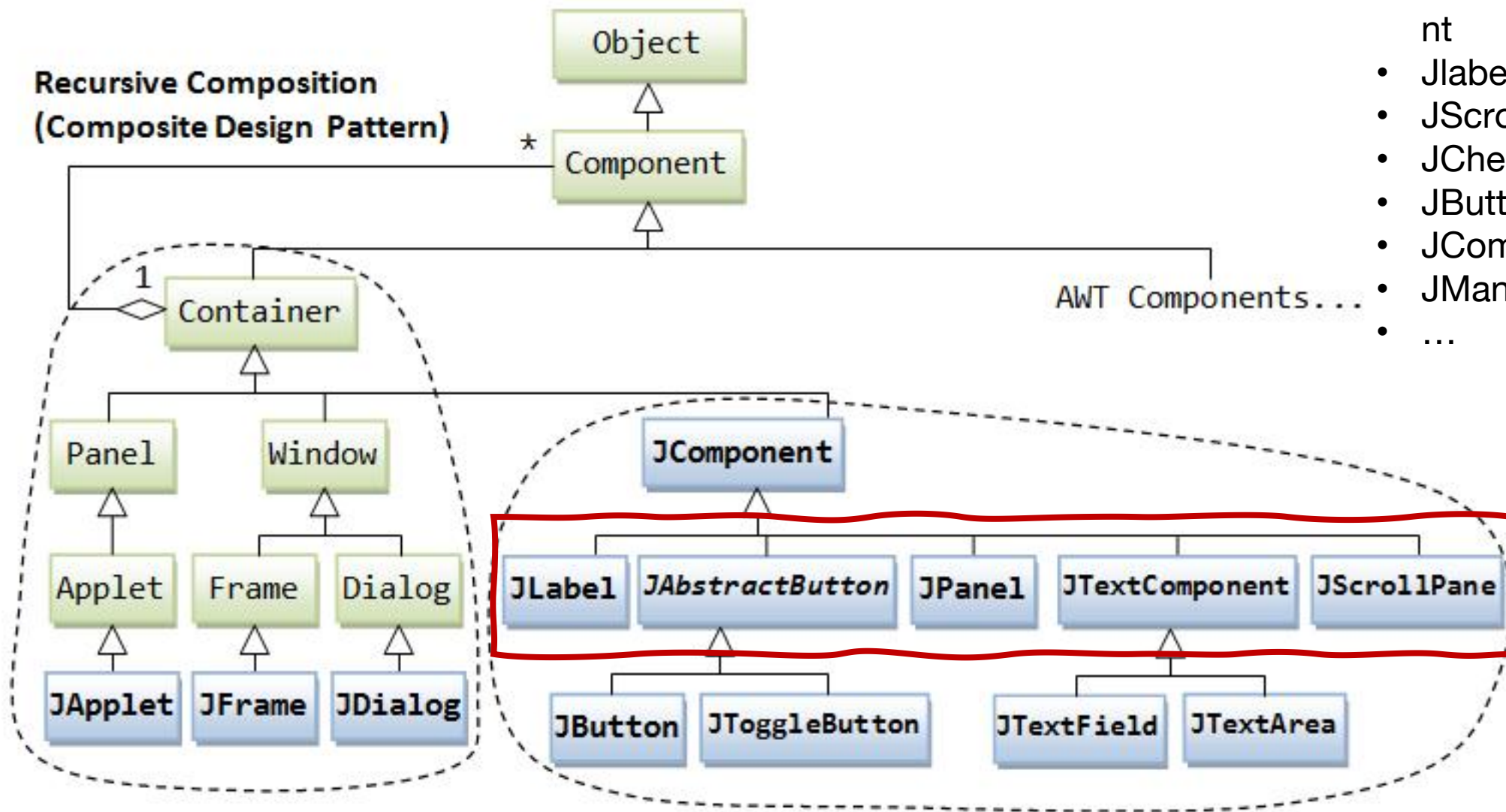
課程內容

- Swing框架
- Swing图形处理、绘制颜色的原理
- 事件机制
- **Swing基本用户组件**
- MVC模式



SWING基本用户组件

Recursive Composition
(Composite Design Pattern)





文本输入

□ 三个继承自JTextComponent类（抽象类）的方法：

- **文本域(JTextField)**: 接收单行文本。
- **文本区(JTextArea)**: 接收多行文本。
- **密码域(JPasswordField)**: 接收单行文本，且不显示文本内容。

□ 文本域

- 改变文本域中的内容；获取用户键入的文本；trim():去掉文本域内容前后的空格

□ 文本区

- 可以输入**多行文本**，用**回车键**换行，每行以\n结尾；构造时，可以指定文本区的行数和列数

□ 密码域

- **特殊类型的文本域**，每个输入的字符由回显字符(echo character)表示；一种典型的回显字符：星号*；char[] getPassword(): **密码并不是以String返回**



文本输入-标签和标签组件

- ❑ 标签是容纳文本的组件。
- ❑ 没有任何的修饰，不能响应用户输入。
- ❑ 可以利用标签标识组件。
 - 文本域本身没有标识。
 - 用正确的文本构造一个JLabel组件。
 - 把它放置在距离需要标识的组件足够近的地方。
- ❑ JLabel构造器允许指定初始文本和图标，及内容的排列方式。

```
new JLabel("User name:",SwingConstants.RIGHT)
```



文本输入-滚动窗格

- 在Swing中，文本区没有滚动条。
- 如果需要滚动条，可以将文本区放在滚动窗格(scroll pane)中。

```
var textArea = new JTextArea(TEXTAREA_ROWS,TEXTAREA_COLUMNS);  
var scrollPane = new JScrollPane(textArea);
```



选择组件-复选框

- 接收的输入只是“是”或“否”
- 自动带有标识标签
- 需要一个紧邻的标签来说明其用途，在构造器中指定标签文本



```
bold = new JCheckBox("Bold");
```

- 选中/取消选中复选框：`bold.setSelected(true);`
- 获取每个复选框的当前状态：`bold.isSelected();`
 - `true`: 选中；`false`: 没有选中



选择组件-单选按钮

- 在多个选择中只能选中一项。为单选按钮组构造ButtonGroup类型的对象，将JRadioButton类型的对象添加到按钮组中。

```
JRadioButton small = new JRadioButton("小杯");  
JRadioButton medium = new JRadioButton("中杯");  
JRadioButton large = new JRadioButton("大杯");  
  
// 创建按钮组，并把三个单选按钮加入进去  
ButtonGroup sizeGroup = new ButtonGroup();  
sizeGroup.add(small);  
sizeGroup.add(medium);  
sizeGroup.add(large);  
// 默认选中一个  
medium.setSelected(true);
```



选择组件-单选按钮

- ButtonGroup类有getSelection方法。
需要用setActionCommand方法明确地为
所有单选按钮设定动作命令。



ACTION Choice Selected: medium

```
var smallButton = new JRadioButton("small",false);  
smallButton.setActionCommand("small");    //设定动作命令
```

```
class checkboxlistener implements ActionListener{  
    public void actionPerformed(ActionEvent ev) {  
        String choice = group.getSelection().getActionCommand();  
        System.out.println("ACTION Choice Selected: " + choice);  
        //执行一些功能  
    }  
}
```

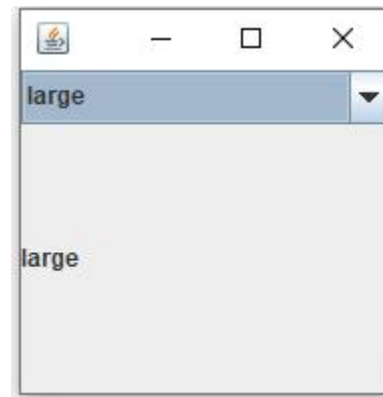


选择组件-组合框

- 提供一个下拉选择列表，可以选择一项。

- 构造组合框并添加选项：

```
JComboBox<String> faceCombo = new JComboBox<>();  
faceCombo.addItem("small");  
faceCombo.addItem("medium");  
faceCombo.addItem("large");
```



- 监听选项：

```
faceCombo.addActionListener(event->  
label.setText(faceCombo.getSelectedItem().toString()));
```

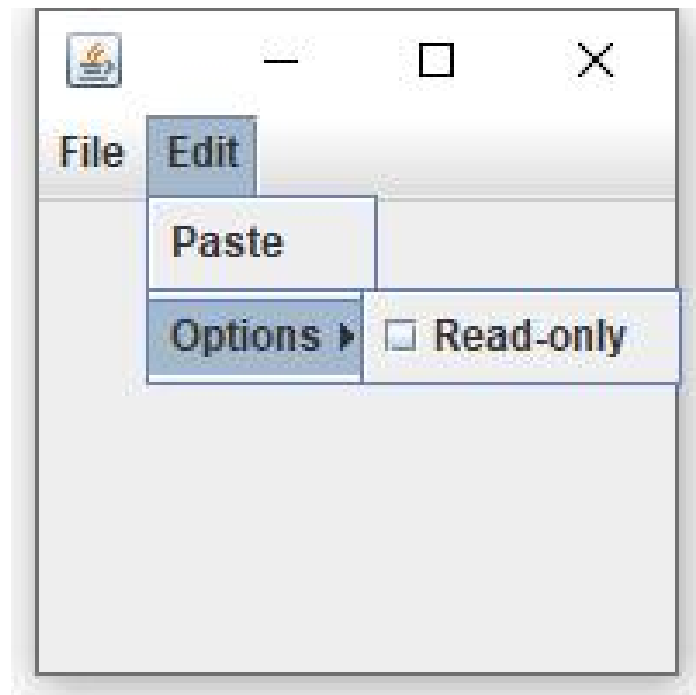
- 删除选项：

```
faceCombo.removeItem("small");  
faceCombo.removeItemAt(0);
```



菜单

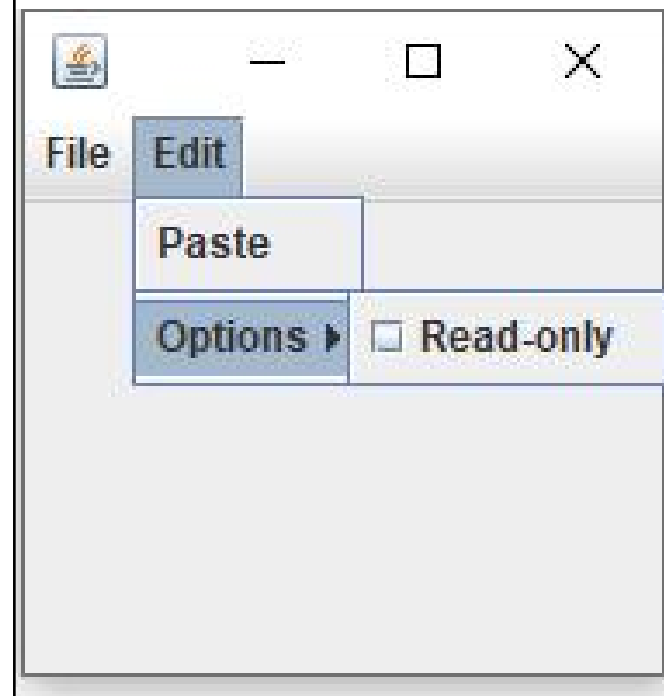
- 位于窗口顶部的菜单栏（menu bar）包括了下拉菜单的名字。点击一个名字就可以打开包含菜单项（menu item）和子菜单（submenu）的菜单。
- 当用户点击菜单项时，所有的菜单都会被关闭并且将一条消息发送给程序。





菜单

```
1. var menuBar = new JMenuBar(); //创建菜单栏
2. frame.setJMenuBar(menuBar); //将菜单栏放在窗体顶部
3. var fileMenu = new JMenu("File");
4. menuBar.add(fileMenu);
5. var editMenu = new JMenu("Edit");
6. menuBar.add(editMenu);
7. var pasteItem = new JMenuItem("Paste");
8. editMenu.add(pasteItem);
9. editMenu.addSeparator();
10. var readOnlyItem = new JCheckBoxMenuItem("Read-only");
11. var optionMenu = new JMenu("Options");
12. optionMenu.add(readOnlyItem);
13. editMenu.add(optionMenu);
```





課程內容

- Swing框架
- Swing图形处理、绘制颜色的原理
- 事件机制
- Swing基本用户组件
- MVC模式



MVC模式

□ 每个组件都有三个特征

- 内容：按钮是否被按下；文本域中的文本等
- 外观：颜色、大小等
- 行为：对事件的反应

□ 为了实现这些需求，Swing采用了一种设计模式(design pattern)：模型-视图-控制器模式（model-view-controller, MVC）

- 模型（model）：存储内容
- 视图（view）：显示内容
- 控制器（controller）：处理用户输入



MVC模式

□ 模型：

- 存储完整的内容。
- 实现改变内容和查找内容的方法。
- 模型没有用户界面，是完全不可见的。

□ 视图：

- 一个模型可以有多个视图。
- 每个视图可以显示全部内容的不同部分或不同方面。
- 当模型更新时，需要通知与之关联的所有视图同步更新。



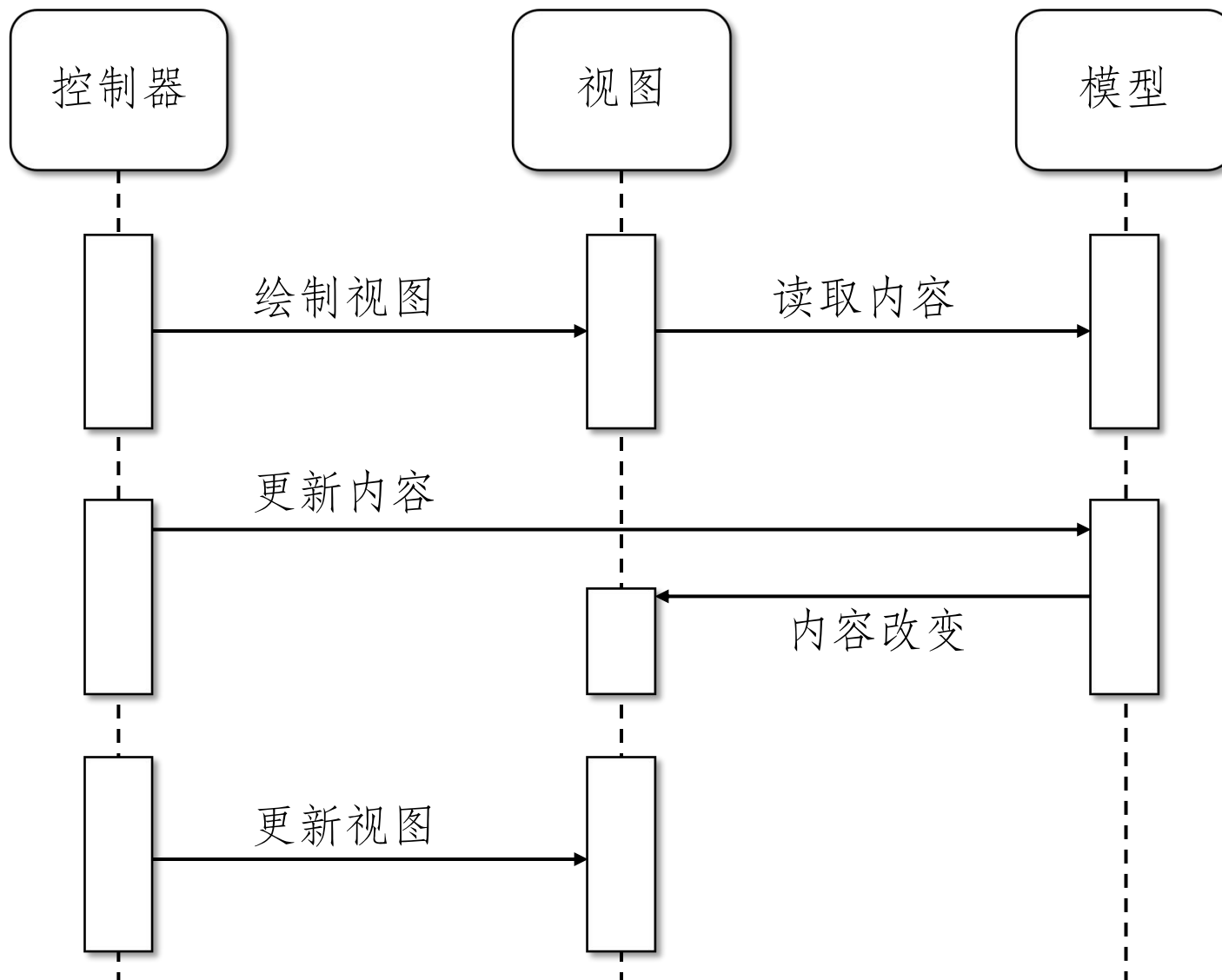
MVC模式

□ 控制器：

- 使视图与模型分离开
- 负责处理用户输入事件，如点击鼠标和按键。
- 决定是否将事件转化成对模型或视图的更改。
- 例：用户在文本框中敲下了一个按键，控制器调用模型的“插入字符”命令，然后模型告诉视图进行更新。
- 例：用户按下一个箭头键，控制器通知视图滚动，对底层文本不会有影响。



MVC模式





課程內容

- Swing框架
- Swing图形处理、绘制颜色的原理
- 事件机制
- Swing基本用户组件
- MVC模式