



# 面向对象的软件构造导论

## 第九章 Swing图形用户界面



# 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



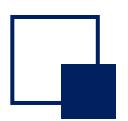
## 课程内容

- 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



## 课程内容

- 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，名为抽象窗口工具包(Abstract Window Toolkit, AWT).
- 基本AWT库将处理用户界面元素的任务委托给各个目标平台上的原生GUI工具包，由原生GUI工具包负责用户界面元素的创建和行为。
- 问题：
  - 编写依赖于原生用户界面元素的高质量、可移植的图形库，非常困难
  - 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



# 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。



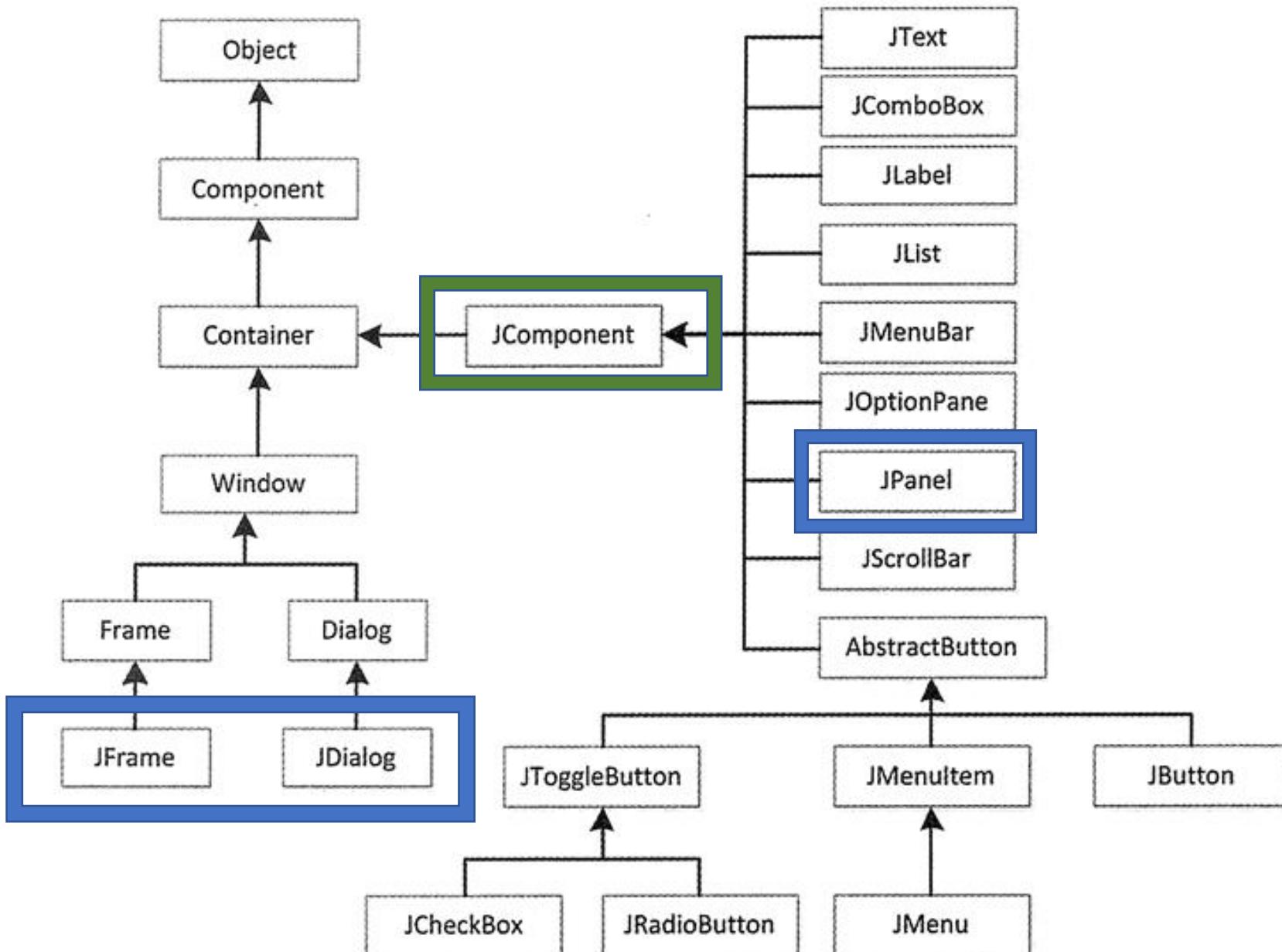
# Swing框架

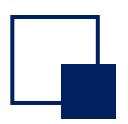
---

- Swing GUI包含了两种元素：组件和容器。
- 组件是单独的控制元素，例如按键或者文本编辑框。组件要放到容器中才能显示出来。
- 容器也是组件，因此容器也可放到别的容器中。
- 组件和容器构成了包含层级关系。



# Swing框架





# 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AWT的类Component及其子类Container。常见的组件有标签JLabel、按键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

---

- 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。



# Swing框架

---

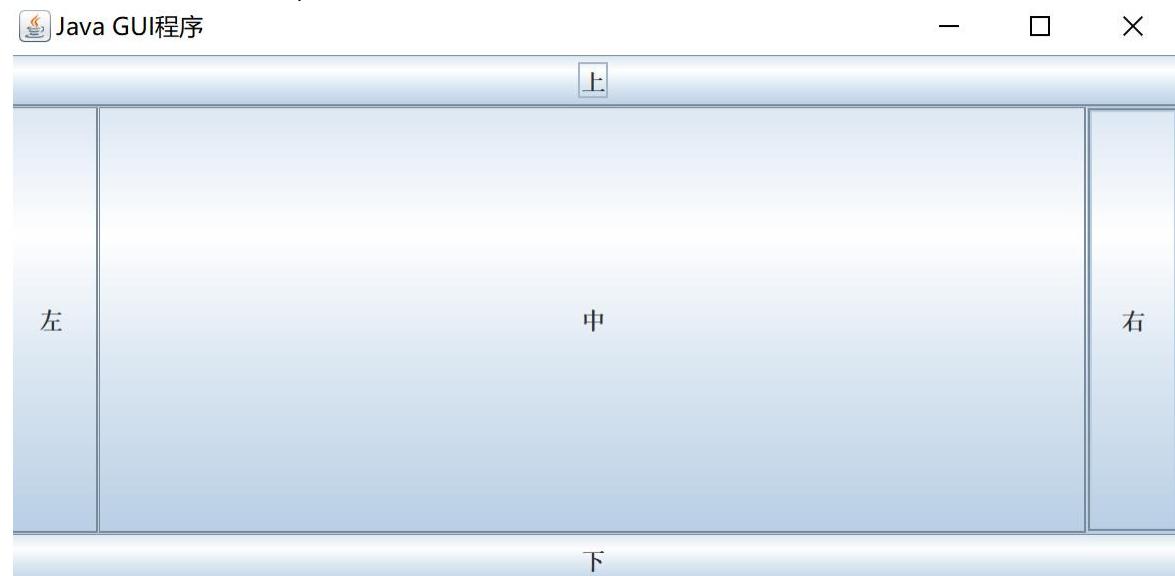
- Java提供了若干种布局管理器。

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

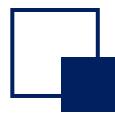
```
JFrame frame=new JFrame("Java GUI程序"); //创建Frame窗口  
frame.setSize(400,200);  
frame.setLayout(new BorderLayout()); //为Frame窗口设置布局  
为BorderLayout  
JButton button1=new JButton ("上");  
JButton button2=new JButton("左");  
JButton button3=new JButton("中");  
JButton button4=new JButton("右");  
JButton 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);  
frame.setVisible(true);  
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```





## 课程内容

- 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。



## 显示窗体

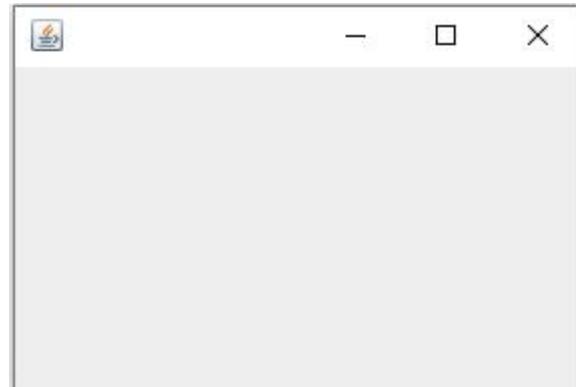
```
1. class SimpleFrame extends JFrame{  
2.     private static final int DEFAULT_WIDTH = 300;  
3.     private static final int DEFAULT_HEIGHT = 200;  
4.  
    public SimpleFrame(){  
        setSize(DEFAULT_WIDTH,DEFAULT_HEIGHT);  
    }  
}
```

- 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.}
```





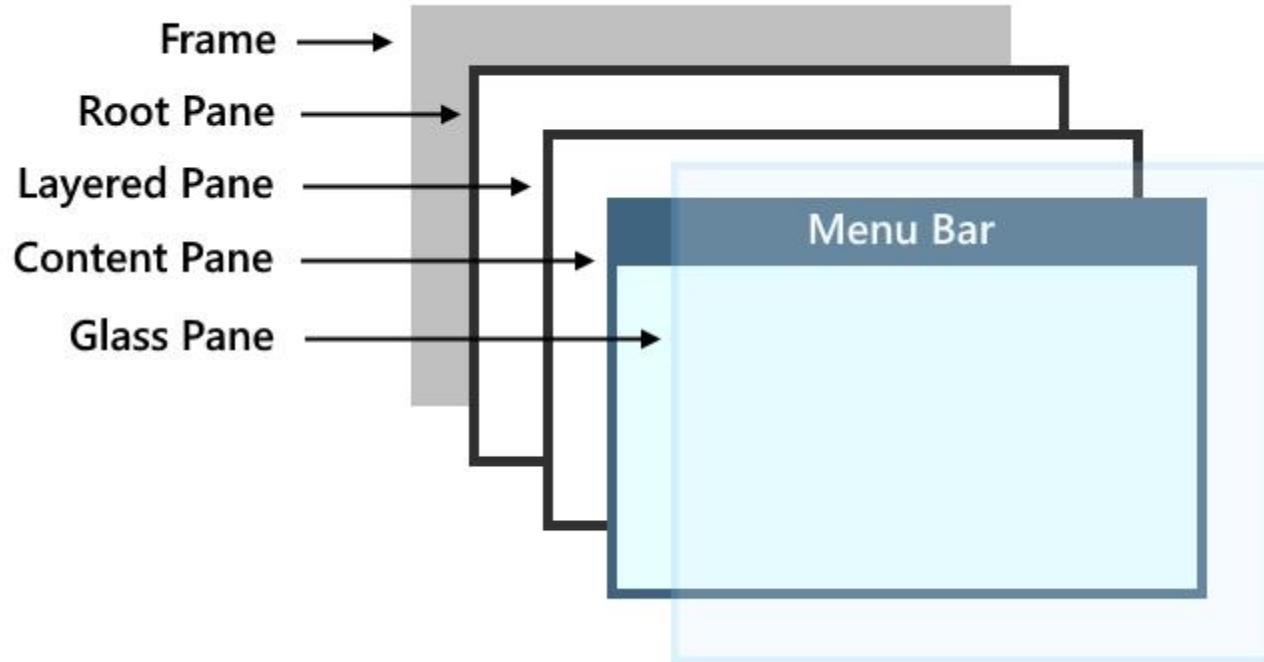
# 显示窗体

---

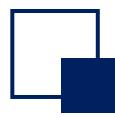
- `EventQueue.invokeLater(()->{statements});`
  - 所有Swing组件必须由事件分派线程 (event dispatch thread) 配置，这是控制线程，它将鼠标点击和按键等事件传递给用户接口组件。
- `frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);`
  - 定义用户关闭窗体时的相应动作——退出。
- `frame.setVisible(true);`
  - 窗体起初是不可见的。程序员可以在向内添加了一系列组件后，通过setVisible方法让它显示。



# 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

---

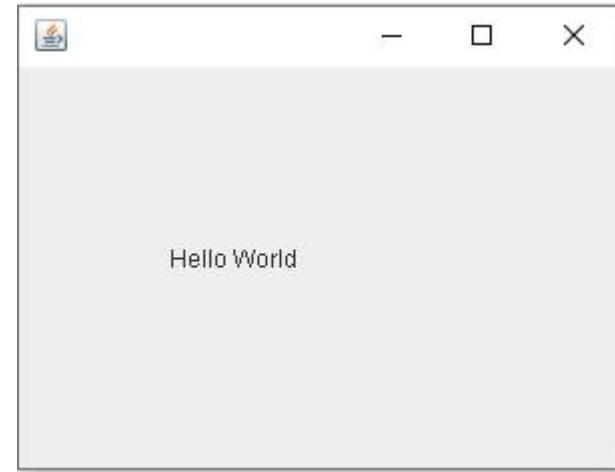
- 我们想在窗体中显示信息（如字符串“Hello World”）。
- 我们把一个组件添加到窗体中，消息将绘制在这个组件上。
- 在组件上进行绘制：
  - 定义一个扩展JComponent的类
  - 覆盖其中的paintComponent方法。
  - 在paintComponent方法中，设置Graphics对象。Graphics对象包含了绘制图案、图像和文本的方法。Java中，所有的绘制都必须通过Graphics对象完成。

## 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.  
    public helloworldFrame()  
15.    {  
16.        add(new helloworldComponent());  
17.        pack();  
18.    }  
19.}
```



- 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：
  - 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。



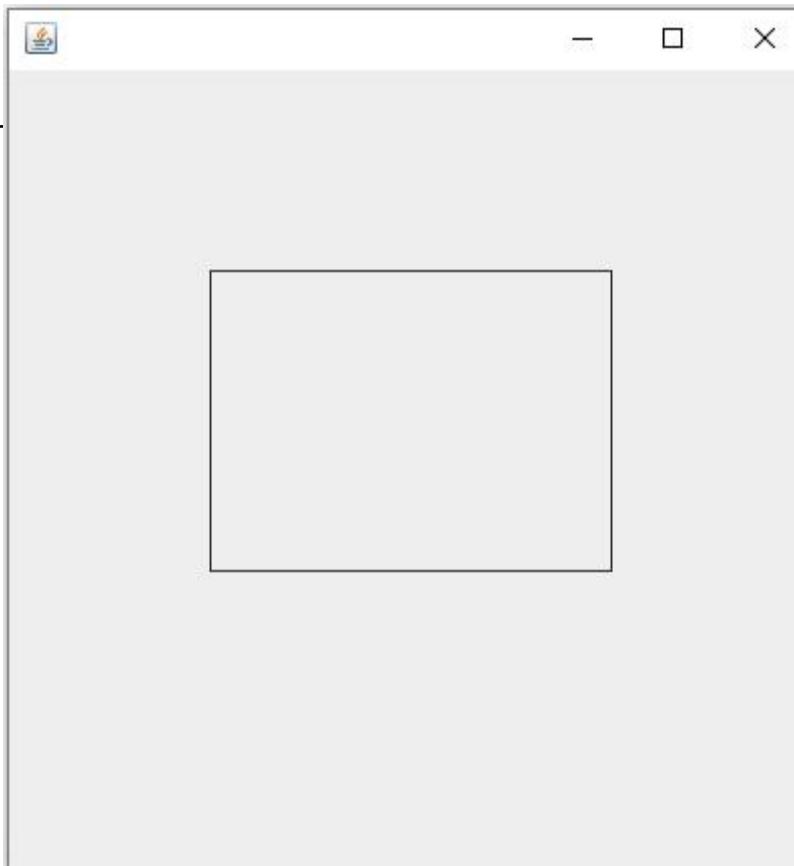
## 绘制2D图形

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



## 绘制2D图形

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

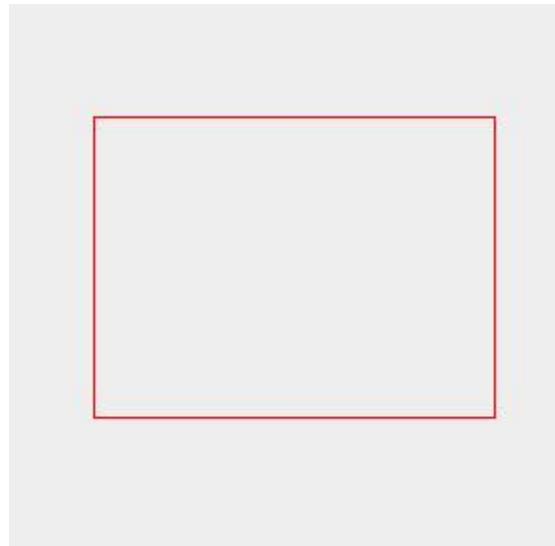


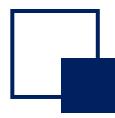


## 使用颜色

- 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。

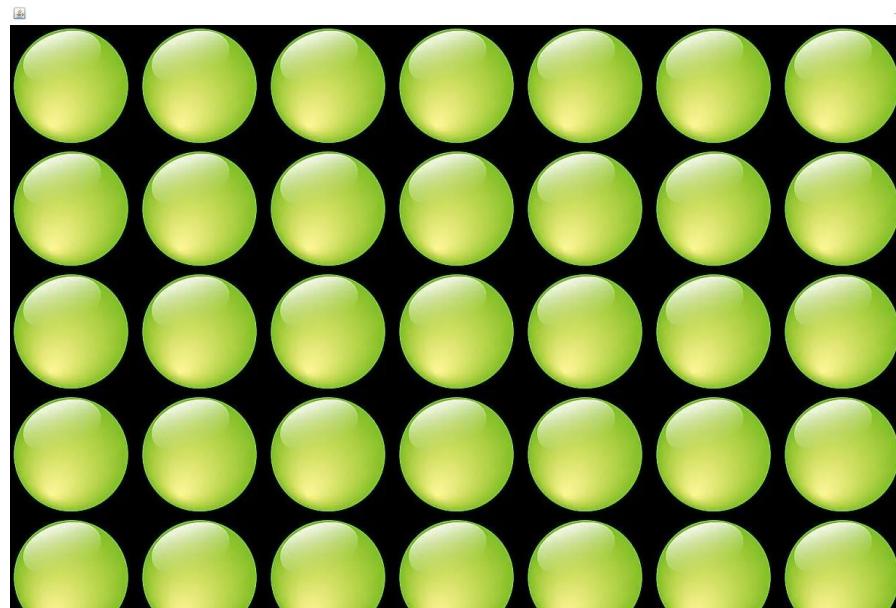
```
1. public void paintComponent(Graphics g)
2. {
3.     int X = 100;
4.     int Y = 100;
5.     Image image = new ImageIcon(path).getImage();
6.
7.     g.drawImage(image, X, Y, null);
}
```



## 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。





# 显示图像



## 课程内容

- 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，从列表中选择一个项目



# 事件机制

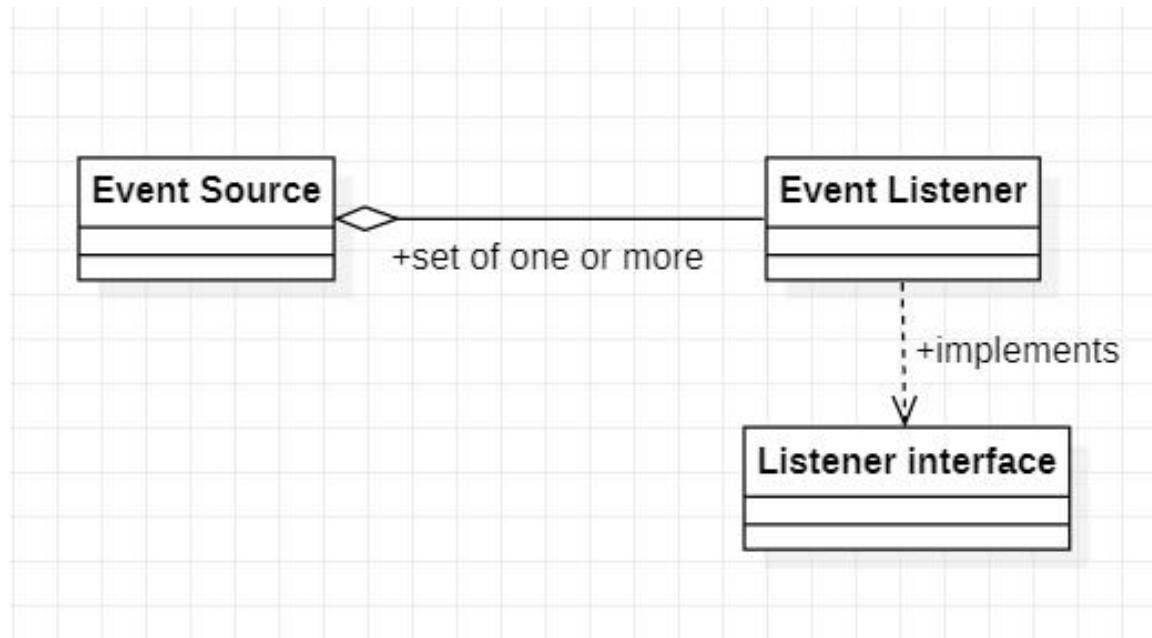
---

- 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。





## 事件机制

---

```
1.ActionListener listener = , , , ;  
2.var button = new JButton("OK");  
3.button.addActionListener(listener)
```

- 只要按钮被点击（产生了“动作事件”），listener对象就会得到通知。



# 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.  
    public ButtonFrame()  
7. {  
8.     setSize(DEFAULT_WIDTH,DEFAULT_HEIGHT);  
9.  
    var YellowButton = new JButton("Yellow");  
10.   var BlueButton = new JButton("Blue");  
11.   var RedButton = new JButton("Red");  
  
12.   buttonPanel = new JPanel();
```



## 实例：按钮点击

---

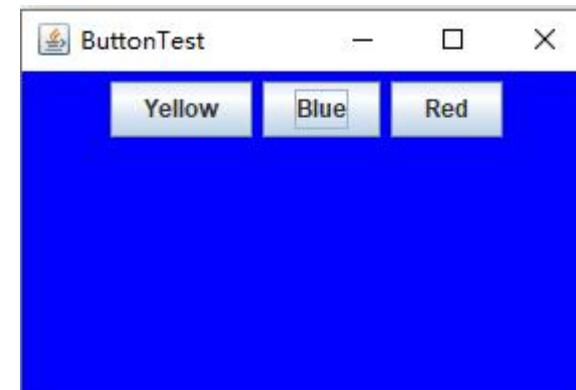
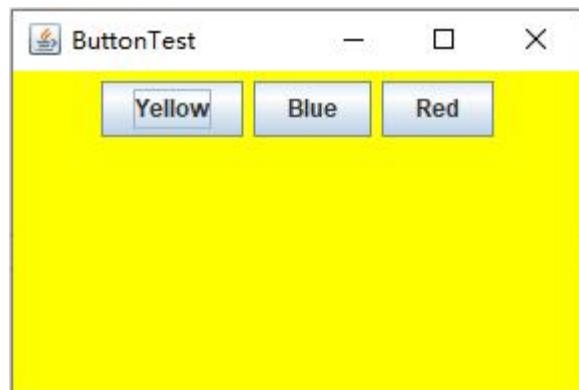
```
13.         buttonPanel = new JPanel();
14.
15.         buttonPanel.add(YellowButton);
16.         buttonPanel.add(BlueButton);
17.         buttonPanel.add(RedButton);
18.
19.         add(buttonPanel);
20.
21.         var YellowAction = new ColorAction(Color.YELLOW);
22.         var BlueAction = new ColorAction(Color.BLUE);
23.         var RedAction = new ColorAction(Color.RED);
24.     }
```



## 实例：按钮点击

---

```
25.private class ColorAction implements ActionListener{  
26.    private Color backgroundColor;  
27.  
28.    public ColorAction(Color C){  
29.        backgroundColor = C;  
30.    }  
31.    public void actionPerformed(ActionEvent event){  
32.        buttonPanel.setBackground(backgroundColor);  
33.    }  
34.}
```





# ActionEvent

```
java.lang.Object
  └── java.util.EventObject
      └── java.awt.AWTEvent
          └── java.awt.event.ActionEvent
```

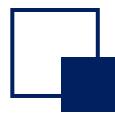
```
button.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        // 1. 获取事件源
        JButton sourceButton = (JButton) e.getSource();

        // 2. 获取动作命令
        String command = e.getActionCommand();
        System.out.println("命令: " + command);

        // 3. 检查是否按Shift键
        if (((e.getModifiers() & ActionEvent.SHIFT_MASK) != 0) {
            System.out.println("Shift键被按下");
        }

        // 4. 获取时间戳
        SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");
        System.out.println("点击时间: " + sdf.format(new Date(e.getWhen())));
    }
});
```

事件类型	触发场景
ActionEvent	按钮点击/菜单选择/文本框回车
ItemEvent	复选框/单选按钮状态改变
MouseEvent	鼠标移动/点击/拖拽
KeyEvent	键盘按键



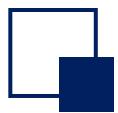
## 简洁地指定监听器

- 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));
}
```



# Lambda

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

```
button.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent event) {
        buttonPanel.setBackground(backgroundColor);
    }
});
```

```
public interface ActionListener {
    void actionPerformed(ActionEvent e); // 只有一个抽象方法
}
```



## 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 ButtonFrame2()
{
    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);
}
```





## 课程内容

- 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



# 文本输入

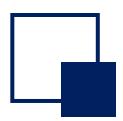
---

## 三个继承自 JTextField 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



# 文本输入

---

## □ 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。

```
var group = new ButtonGroup();

var smallButton = new JRadioButton("small",false);
//标签small,初始不选中
group.add(smallButton);

.....
```



## 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
    }
}
```

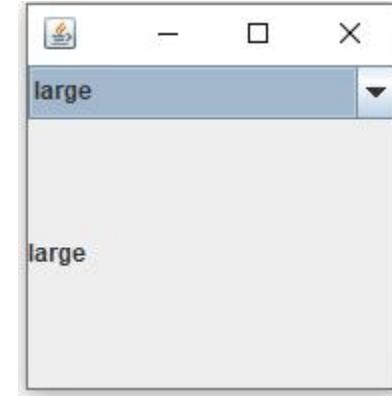


## 选择组件-组合框

- 提供一个下拉选择列表，可以从中选择一项。

- 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：

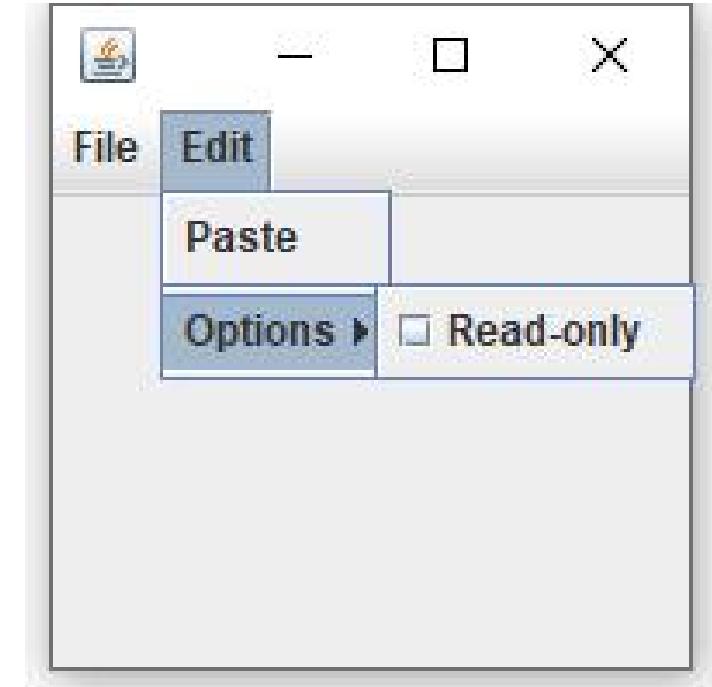
```
faceCombo.removeItem("small");  
faceCombo.removeItemAt(0);
```



# 菜单

---

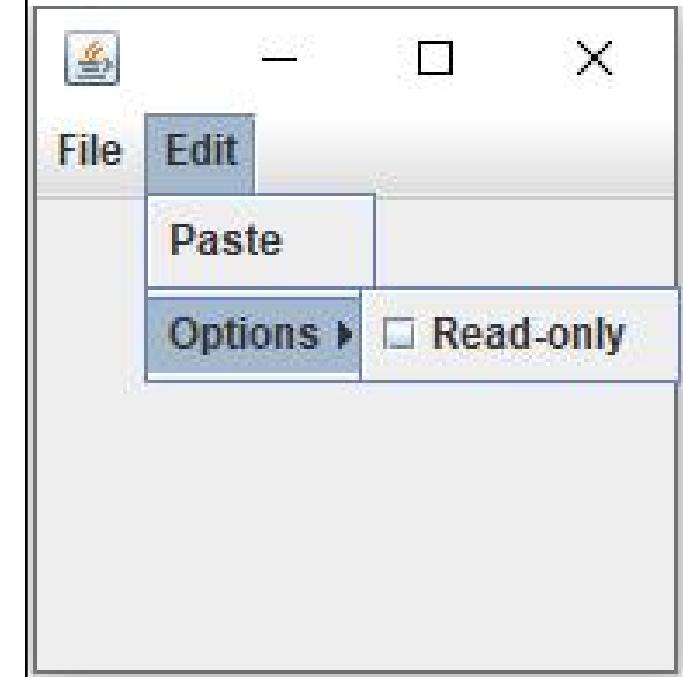
- 位于窗口顶部的菜单栏 (menu bar) 包括了下拉菜单的名字。点击一个名字就可以打开包含菜单项 (menu item) 和子菜单 (submenu) 的菜单。
- 当用户点击菜单项时，所有的菜单都会被关闭并且将一条消息发送给程序。





# 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.
4. var fileMenu = new JMenu("File");
5. menuBar.add(fileMenu);
6. var editMenu = new JMenu("Edit");
7. menuBar.add(editMenu);
8. var pasteItem = new JMenuItem("Paste");
9. editMenu.add(pasteItem);
10. editMenu.addSeparator();
11. var readonlyItem = new JCheckBoxMenuItem("Read-only");
12. var optionMenu = new JMenu("Options");
13. optionMenu.add(readonlyItem);
14. editMenu.add(optionMenu);
```





## 课程内容

- 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

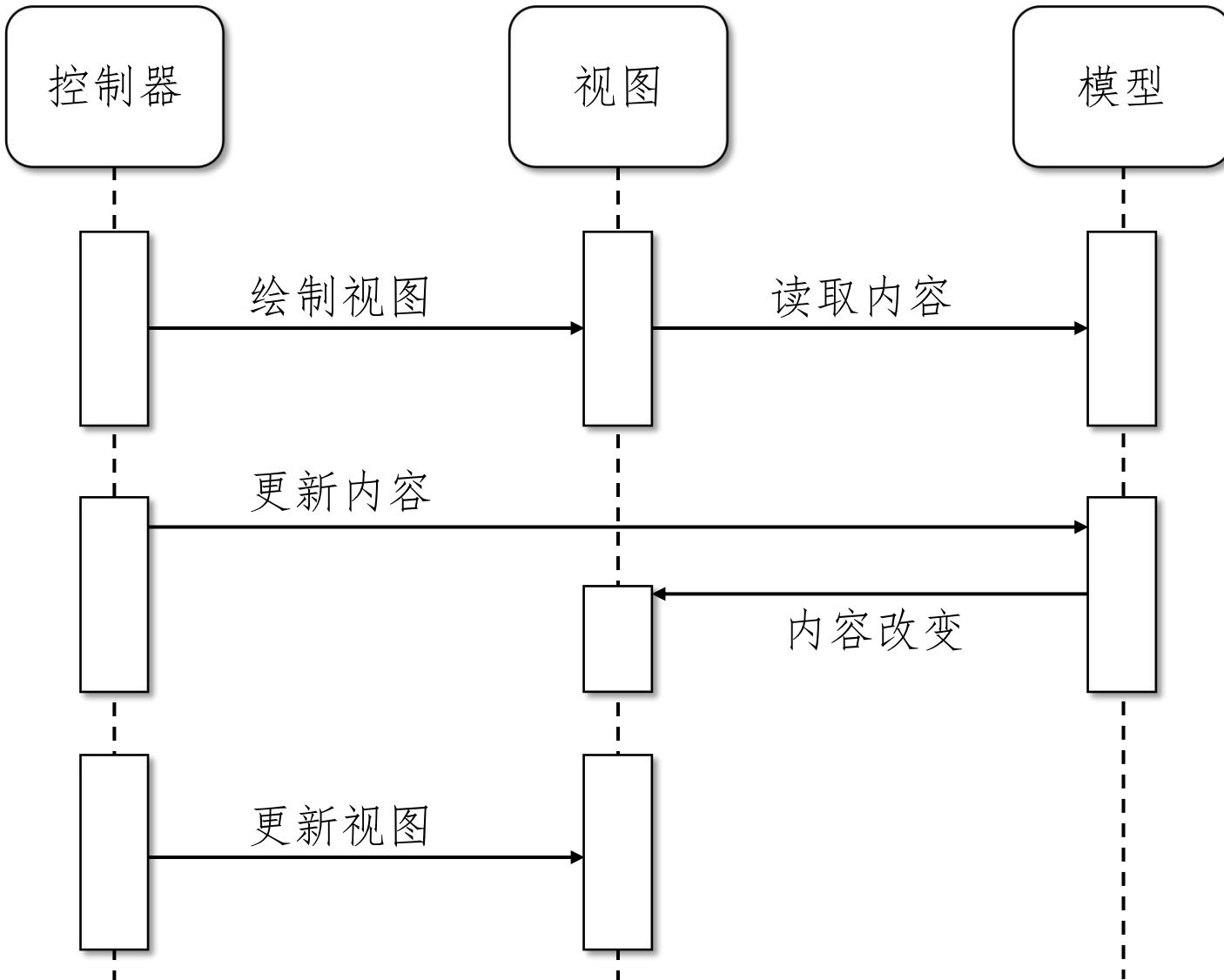
---

## □ 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





## 课程内容

- 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