Java 程序设计

第8章 事件处理

田英鑫 tyx@hit.edu.cn
哈尔滨工业大学软件学院

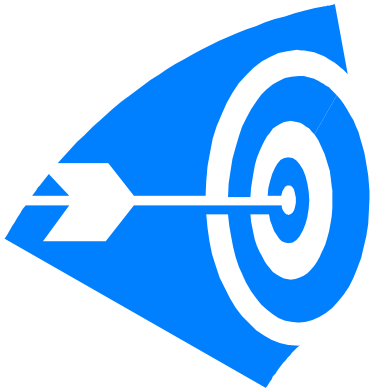
第8章 事件处理

& 本章导读

第8章 事件处理

& 本章重点

- 8.2 Java 的事件处理模型
- 8.3 实例：处理按钮单击事件
- 8.7 窗口事件
- 8.9 键盘事件
- 8.10 鼠标事件

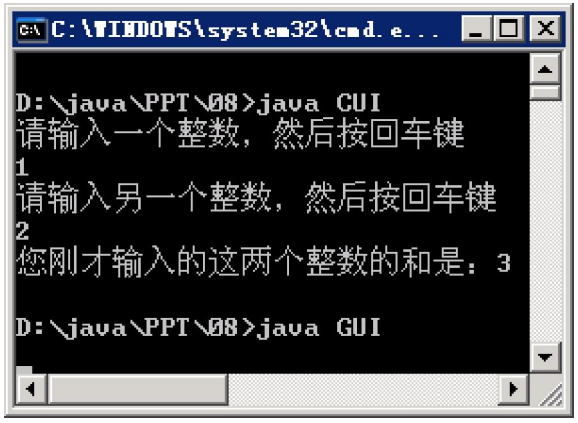


& 本章难点

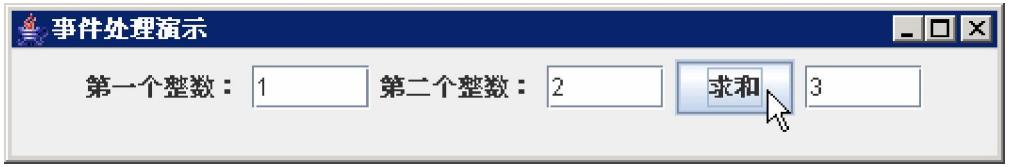
- 8.4 使用内部类处理事件
- 8.5 特殊的监听器

8.1 事件处理基础

- 应用程序的用户界面
  - CUI
    - Characteral User Interface
  - GUI
    - Graphical User Interfaces



CUI程序



GUI程序

8.1 事件处理基础

- CUI 程序的特点
  - 主要采用顺序的、过程驱动的程序设计方法
  - 程序是一系列预先定义好的操作序列的组合
  - 程序按预先设定好的顺序执行
- GUI 程序的特点－事件驱动
  - 程序不是由事先预定好的顺序执行的，而是由事件的发生来控制的
  - 事件的发生是随机的、不确定的，并没有预定的顺序，这样就允许用户用各种合理的顺序来安排程序的执行流程
  - 对于图形用户界面而言，最重要的是实现和用户的交互，接受用户的输入，并执行相应的动作

8.1 事件处理基础

- 什么是事件？
  - 事件本身就是一个抽象的概念，他是表现另一对象状态变化的对象
  - 在面向对象的程序设计中，事件是以消息（即事件对象）进行传递的，事件消息是对象间通信的基本方式
  - 在GUI程序中组件对象根据用户的交互产生各种类型的事件对象，这些事件对象由应用程序的事件处理代码捕获
  - 事件在Java中和其他对象基本是一样的，但有一点不同的是，事件对象是由系统自动生成自动传递到适当的事件处理程序

8.1 事件处理基础

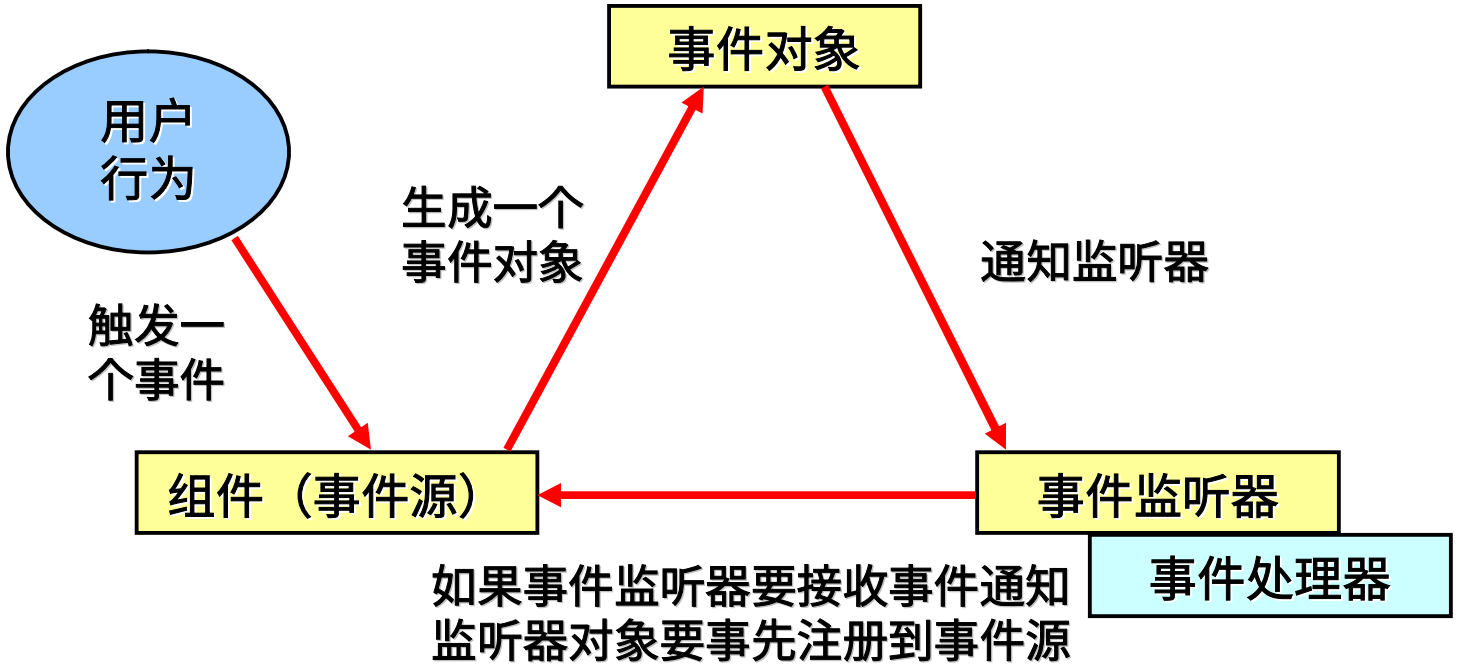
- 事件
  - 事件的触发
    - 用户行为，如：
      - 移动鼠标，点击鼠标按钮和按下键盘键等
    - 操作系统，如：
      - 时钟等，也可以引发事件
    - 程序触发
      - 用户程序触发事件
  - 事件的响应
    - 程序可以响应事件也可以忽略事件
  - 事件的相关信息封装在一个事件对象中
    - Java中的事件是java.util.EventObject派生出来的

8.2 Java 的事件处理模型

- Java 中与事件相关的概念
  - 事件对象
    - 事件发生后产生的对象
    - 不同的事件产生的事件对象是不同的
  - 事件源
    - 发生事件的组件就是事件源
  - 事件监听器
    - 事先注册给事件源的对象，事件发生后在其内部的事件处理器中负责对事件作出响应
- Java 中与事件处理相关的包
  - java.awt.event
  - javax.swing.event

8.2 Java 的事件处理模型

- 事件的注册和通知

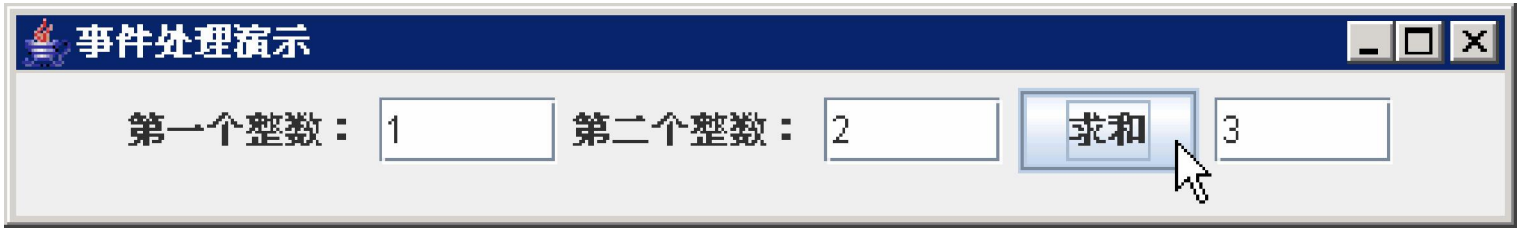


8.2 Java 的事件处理模型

- **AWT 的事件处理机制**
  - 监听器对象是一个实现了特定监听器接口的类的实例，不同的事件对应不同的监听器接口
  - 事件源是一个能够注册监听器对象并向它们发送事件对象的对象
  - 一个事件源可以注册多个事件监听器，当事件发生时，事件源会把事件对象发送给所有注册过的监听器
  - 一个监听器对象也可以注册给多个事件源，监听器对象将利用事件对象中的信息决定如何对事件做出响应

8.3 实例：处理按钮点击事件

- 程序运行界面



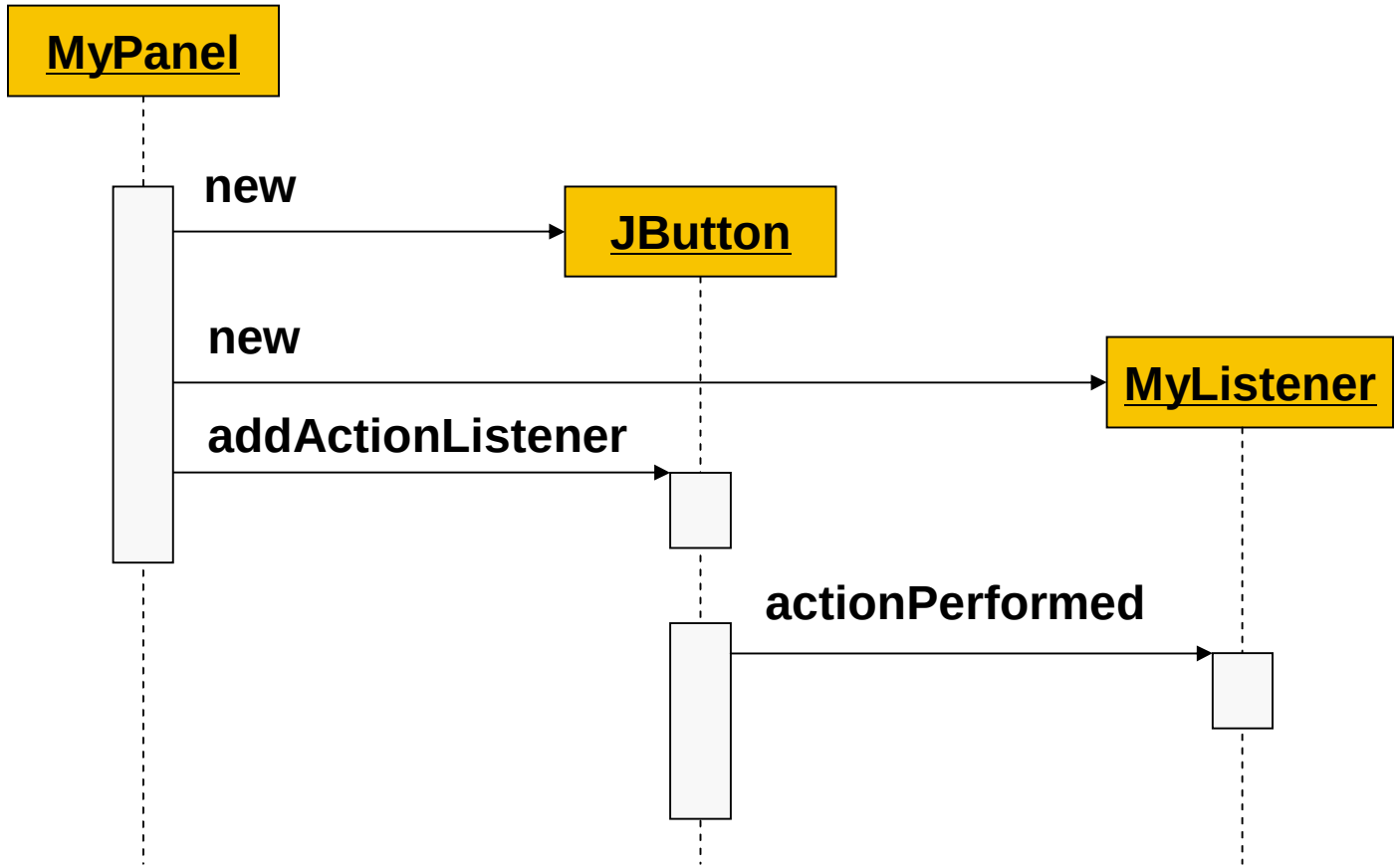
- 程序运行方式
  - 在前两个文本域中分别输入两个整数
  - 点击“求和”按钮后把计算结果显示在第三个文本域中

8.3 实例：处理按钮点击事件

- 程序中涉及的主要对象（类）
  - 存放各种可视控件的容器
    - MyPanel
  - 事件源
    - JButton
  - 事件监听器
    - MyListener
  - 其它组件
    - 两个JLabel用于显示提示信息
    - 三个JTextField用于输入和输出数据
    - 主窗口MyFrame

8.3 实例：处理按钮点击事件

- 事件的通告过程



8.3 实例：处理按钮点击事件

- 创建一个摆放控件的面板

```
class MyPanel extends JPanel {
   JLabel lb1 = new JLabel("第一个整数：");
   JTextField tf1 = new JTextField(5);
   JLabel lb2 = new JLabel("第二个整数：");
   JTextField tf2 = new JTextField(5);
   JButton btn = new JButton("求和"); //该按钮为事件源
   JTextField tf3 = new JTextField(5);
   public MyPanel() {
      add(lb1);add(tf1);add(lb2);add(tf2);
      add(btn);add(tf3);
      ...
   }
}
```

8.3 实例：处理按钮点击事件

- 构造一个处理 ActionEvent 的监听器

```
class MyListener implements ActionListener // 事件监听器
{
   public void actionPerformed(ActionEvent e) // 事件处理器
   {
      String s1 = tf1.getText();
      String s2 = tf2.getText();
      int a = Integer.parseInt(s1);
      int b = Integer.parseInt(s2);
      int c = a + b;
      tf3.setText("" + c);
   }
}
```

8.3 实例：处理按钮点击事件

- 实例化监听器对象并注册给事件源

```
class MyPanel extends JPanel
{
    ...
    // 实例化监听器对象
    ActionListener listener = new MyListener();
    public MyPanel()
    {
        ...
        // 将监听器对象注册给事件源
        btn.addActionListener(listener);
    }
}
```

8.3 实例：处理按钮点击事件

DEMO

8.4 使用内部类处理事件

n 使用普通内部类处理事件

```
class MyPanel extends JPanel
{
    ...
    // 如果希望在监听器中访问事件源对象所属的类的其它成员
    // 则需要将监听器声明成该事件源对象所属的类的内部类
    class MyListener implements ActionListener
    {
        public void actionPerformed(ActionEvent event)
        {
            ...
        }
    }
}
```

8.4 使用内部类处理事件

- 使用匿名内部类处理事件

```
class MyPanel extends JPanel {
    public MyPanel() {
        JButton btn = new JButton("求和");
        btn.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                ...
            }
        });
    }
}
```

8.5 特殊的监听器

- 将事件源所属的容器作为监听器

```
class MyPanel extends JPanel implements ActionListener
{
   public MyPanel()
   {
      JButton btn = new JButton("求和");
      btn.addActionListener(this); // this就是监听器
   }
   public void actionPerformed(ActionEvent event)
   {
      ...
   }
}
```

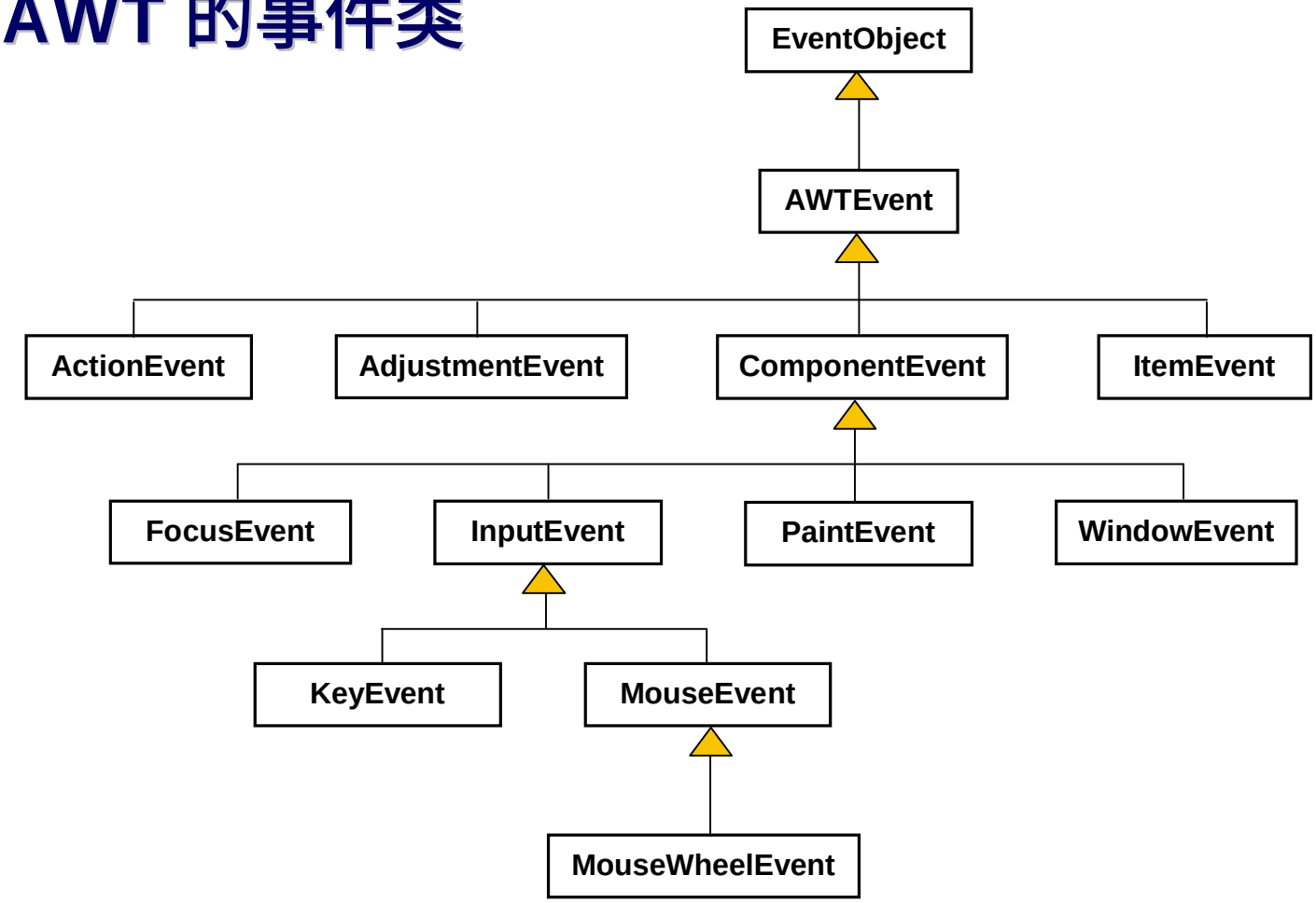
8.5 特殊的监听器

- 将事件源对象本身作为监听器（用处不大）

```
class MyButton extends JButton implements ActionListener
{
    public MyButton(String title)
    {
        super(title);
        // this即是事件源又是监听器
        this.addActionListener(this);
    }
    public void actionPerformed(ActionEvent event)
    {
        ...
    }
}
```

8.6 AWT 事件继承层次

- AWT 的事件类



8.6 AWT 事件继承层次

- 常用的 AWT 事件
  - ActionEvent
  - ItemEvent
  - KeyEvent
  - MouseEvent
  - MouseWheelEvent
  - FocusEvent
  - WindowEvent

常用事件及监听器方法参见教材：表8-1

8.7 窗口事件

- 事件类
  - WindowEvent
- 监听器接口
  - WindowListener
- 监听器方法（处理器）
  - windowClosing(WindowEvent)
  - windowClosed(WindowEvent)
  - windowOpened(WindowEvent)
  - windowIconified(WindowEvent)
  - windowDeiconified(WindowEvent)
  - windowActivated(WindowEvent)
  - windowDeactivated(WindowEvent)

8.7 窗口事件

- 实例：捕获窗口事件

```
class Terminator implements WindowListener
{
   public void windowClosing(WindowEvent e)
   {
      System.exit(0);
   }
   public void windowOpened(WindowEvent e) {}
   public void windowIconified(WindowEvent) {}
   public void windowDeiconified(WindowEvent) {}
   public void windowClosed(WindowEvent) {}
   public void windowActivated(WindowEvent) {}
   public void windowDeactivated(WindowEvent) {}
}
```

8.8 事件适配器

- 什么是事件适配器
  - 为简化编程，JDK针对大多数事件监听器接口定义了相应的实现类，这些类称为事件适配器

```
class Terminator extends WindowAdapter
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
    //不需要重写其他方法
}
```

8.8 事件适配器

- 事件适配器的作用
  - 在适配器类中实现了相应的监听器接口中所有的抽象方法，但不做任何处理，
  - 子类只要继承适配器类就等于实现了相应的监听器接口
  - 子类可以选择性的覆盖监听器接口中的方法，无需覆盖所有的方法
- 事件适配器的缺点
  - 根据Java的单继承行，子类若继承自事件适配器类就不能再继承自其它的类

8.9 键盘事件

- 键盘事件监听器接口
  - 处理键盘事件，可以使用下面KeyListener接口中的处理器
- KeyListener 接口中的监听器方法
  - keyPressed(KeyEvent e)
    - 按下键时触发
  - keyReleased(KeyEvent e)
    - 松开键时触发
  - keyTyped(KeyEvent e)
    - 按下并松开键时触发

8.9 键盘事件

- KeyEvent 类
  - getKeyCode()
    - 返回虚拟键码
  - getKeyChar() 方法
    - 返回用户键入的字符，用于keyTyped方法中
- 常用的键和对应的KeyEvent中的常量
  - A…Z　　VK_A…VK_Z
  - 0…9　　VK_0…VK_9
  - Home　　VK_HOME
  - Page Up　　VK_PGUP

8.9 键盘事件

- KeyEvent 类的常用方法
  - 判断Shift键是否被按下
    - isShiftDown
  - 判断Control键是否被按下
    - isControlDown
  - 判断Alt键是否被按下
    - isAltDown
  - 判断Meta键是否被按下
    - isMetaDown
    - Meta键是Sun和Macintosh键盘专有的键

参见例8-3:Sketch.java、KeyEventTest.java

8.10 鼠标事件

- 处理鼠标事件的两个监听器接口
  - MouseListener
  - MouseMotionListener
- MouseListener 接口
  - 用来监听鼠标的按下、松开、进入、退出和点击等行为
- MouseMotionListener 接口
  - 用来监听鼠标的移动和拖动等行为

8.10 鼠标事件

- MouseListener 接口
  - mouseEntered(MouseEvent e)
  - mouseExited(MouseEvent e)
  - mousePressed(MouseEvent e)
  - mouseReleased(MouseEvent e)
  - mouseClicked(MouseEvent e)
- MouseMotionListener 接口
  - mouseMoved(MouseEvent e)
  - mouseDragged(MouseEvent e)

8.10 鼠标事件

- MouseEvent 类的常用方法
  - getModifiersEx
    - 获取鼠标按钮和键盘的修饰符
  - getPoint
    - 返回事件发生时Point类型对象代表的鼠标坐标
  - getX
    - 返回事件发生时鼠标X坐标
  - getY
    - 返回事件发生时鼠标Y坐标
  - getClickCount
    - 返回击键次数

参见例8-4:MouseTest.java

8.11 多点传送

- 发送给多个监听器
  - 当事件产生时一般将事件发送给一个监听器对象
  - 有时需要把一个事件同时传递给多个监听器对象
- 实例：关闭所有窗口
  - 使用New按钮生成多个窗口
  - 使用Close all按钮关闭所有的窗口

参见例8-6:MulticastTest.java

Any Question?