第16章
Introduction to Java Programming, 8E

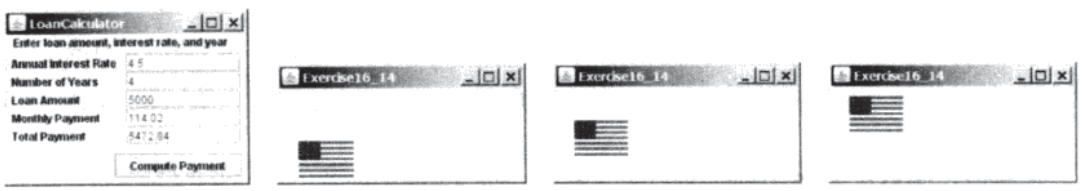
# 事件驱动程序设计

**学习目标**

- 描述事件、事件源和事件类（16.2节）。
- 定义监听器类、向源对象注册监听器对象，然后编写代码来处理事件（16.3节）。
- 使用内部类定义监听器类（16.4节）。
- 使用匿名内部类定义监听器类（16.5节）。
- 探究创建和注册监听器的各种编码风格（16.6节）。
- 点击按钮从文本域获取输入（16.7节）。
- 编写程序处理`WindowEvent`（16.8节）。
- 使用监听器接口适配器简化监听器类的代码（16.9节）。
- 编写程序处理鼠标事件`MouseEvent`（16.10节）。
- 编写程序处理键盘事件`KeyEvent`（16.11节）。
- 使用`javax.swing.Timer`类控制动画（16.12节）。

## 16.1 引言

假如希望编写一个GUI程序，提示用户输入贷款总额、年利率和年数，然后点击Compute Loan按钮获取月偿还额和总偿还额，如图16-1a所示。如何完成这个任务呢？必须使用事件驱动程序设计来编写代码以响应点击按钮事件。




a) b) c) d)

图16-1 a）程序计算贷款偿还额；b）～d）一面旗向上升起

假设希望编写程序用动画实现一面旗上升，如图16-1b～图16-1d所示。该如何完成这个任务呢？解决这个问题的方式有好几种。一种有效的方式就是在事件驱动程序设计中使用一个定时器，这也是本章的主题。

在事件驱动程序设计中，当事件发生时（例如，点击按钮、移动鼠标或定时器）执行代码。14.6节中尝试了一下事件驱动程序设计。你可能会有很多问题，例如，为什么一个定义监听器类来实现`ActionListener`接口。本章将给你所有问题的答案。

## 16.2 事件和事件源

运行Java图形用户界面程序时，程序与用户进行交互，事件驱动程序的执行。事件（event）可以定义为程序发生了某些事情的信号。外部用户动作和内部程序动作都可以触发事件，外部用户动作的例子有移动鼠标、点击按钮和敲击键盘等，而内部程序动作的例子有定时器。程序可以选择响应事件或忽略

事件。

能创建一个事件并触发该事件的组件称为源对象（source object）或源组件（source component）。例如，按钮是按钮点击动作事件的源对象。一个事件是事件类的实例。事件类的根类是java.util.EventObject。一些事件类的层次关系如图16-2所示。

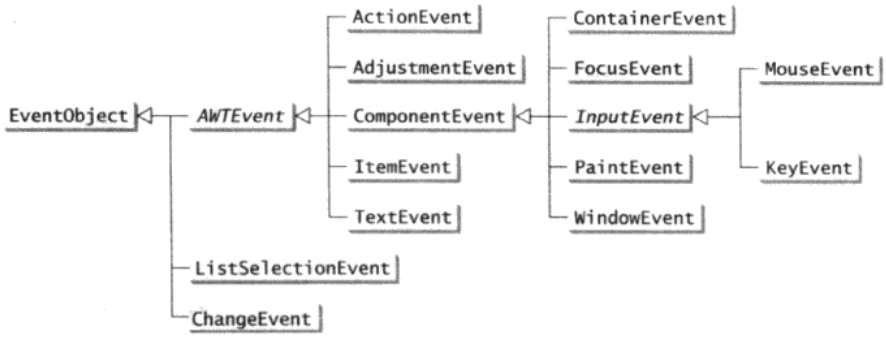



图16-2 事件是EventObject类的对象

事件对象包含与事件相关的一切属性。可以使用EventObject类中的实例方法getSource()获得事件的源对象。EventObject类的子类处理特定类型的事件，例如，动作事件、窗口事件、组件事件、鼠标事件以及按键事件。表16-1列出了外部用户动作、源对象和触发的事件类型。

**表16-1 用户动作、源对象和事件类型**

| 用户动作 | 源对象 | 触发的事件类型 |
|---|---|---|
| 点击按钮 | JButton | ActionEvent |
| 在文本域按回车键 | JTextField | ActionEvent |
| 选定一个新项 | JComboBox | ItemEvent、ActionEvent |
| 选定（多）项 | JList | ListSelectionEvent |
| 点击复选框 | JCheckBox | ItemEvent、ActionEvent |
| 点击单选按钮 | JRadioButton | ItemEvent、ActionEvent |
| 选定菜单项 | JMenuItem | ActionEvent |
| 移动滚动条 | JScrollBar | AdjustmentEvent |
| 移动滚动条 | JSlider | ChangeEvent |
| 窗口打开、关闭、最小化、还原或关闭中 | Window | WindowEvent |
| 按住、释放、点击、回车或退出鼠标 | Component | MouseEvent |
| 移动或拖动鼠标 | Component | MouseEvent |
| 释放或按下键盘上的键回车或退出 | Component | KeyEvent |
| 从容器中添加或删除组件 | Container | ContainerEvent |
| 组件移动、改变大小、隐藏或显示 | Component | ComponentEvent |
| 组件获取或失去焦点 | Component | FocusEvent |

**注意** 如果一个组件可以触发某个事件，那么这个组件的任意子类都可以触发同类型的事件。例如，每个GUI组件都可以触发MouseEvent、KeyEvent、FocusEvent和ComponentEvent，因为Component是所有GUI组件的父类。

**注意** 图16-2中除了ListSelectionEvent和ChangeEvent之外的所有事件类都包括在java.awt.event包中，ListSelectionEvent和ChangeEvent在javax.swing.event包中。AWT事件本来是为AWT组件设计的，但是许多Swing组件都会触发它们。

## 16.3 监听器、注册以及处理事件

Java使用一种基于委托的模型来处理事件：源对象触发一个事件，对此事件感兴趣的对象会处理它。将对此事件感兴趣的对象称为监听器（listener）。一个对象要成为源对象上的事件监听器，需要具备两个条件，如图16-3所示。

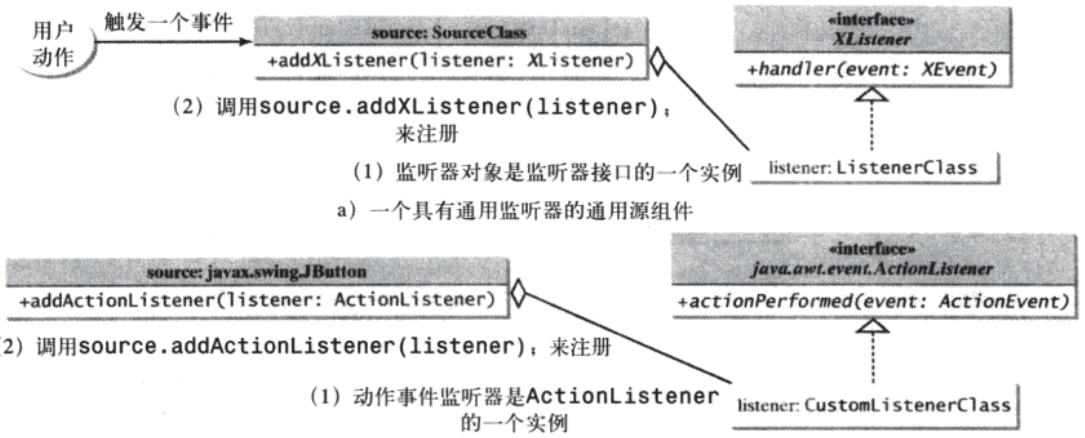



图16-3 监听器必须是一个监听器接口的实例，并且必须注册到源组件

1）监听器对象的类必须是相应的事件监听器接口的实例，以确保监听器有处理这个事件的正确方法。Java为每一种类型的事件都提供了监听器接口。通常，事件XEvent的监听器接口命名为XListener，监听器MouseMotionListener例外。例如，事件ActionEvent对应的监听器接口是ActionListener，ActionEvent的每个监听器都应该实现ActionListener接口。表16-2列出事件类型、对应的监听器接口以及定义在监听器接口中的方法。包含处理事件的监听器接口称为处理器（handler）的方法。

2）监听器对象必须由源对象注册。注册方法依赖于事件的类型。如果事件类型是ActionEvent，那么对应的注册方法是addActionListener。一般来说，XEvent的注册方法命名为addXListener。一个源对象可以触发几种类型的事件。一个源对象针对每个事件都维护着一个注册的监听器列表，通过调用监听器对象上的处理器来通知所有注册的监听器响应这个事件，如图16-4所示（图16-4显示源类的内部实现。为了使用它，不需要知道像JButton这样的源类是如何实现的。然而，这些知识将有助于理解Java事件驱动程序设计的框架结构）。

我们重新看看程序清单14-8。因为JButton对象触发ActionEvent，ActionEvent的监听器对象必须是ActionListener的实例，所以监听器类在第34行实现ActionListener。源对象调用addActionListener(listener)来注册一个监听器，如下所示：

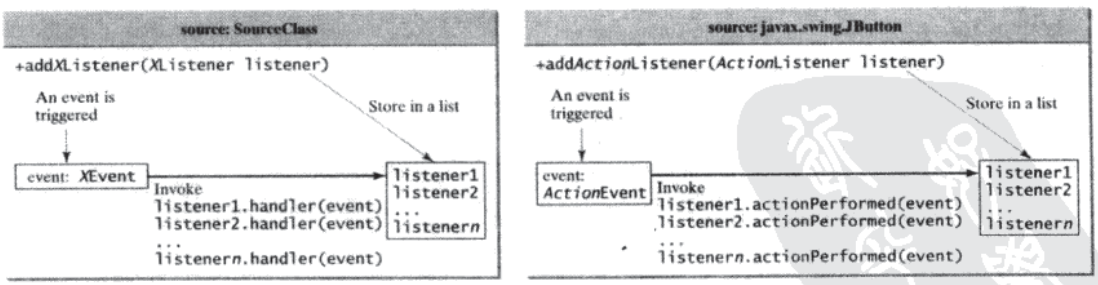
```
JButton jbtOK = new JButton("OK"); // Line 7 in Listing 14.8
ActionListener listener1
   = new OKListenerClass(); // Line 18 in Listing 14.8
jbtOK.addActionListener(listener1); // Line 20 in Listing 14.8
```

点击该按钮时，JButton对象触发一个ActionEvent，然后将它作为参数传递以调用监听器的actionPerformed方法来处理这个事件。

事件对象包含与事件相关的信息，这些可以使用如图16-5所示的方法获得。例如，为判断一个对象是按钮、复选框还是单选按钮，都可以使用e.getSource()方法获得源对象。对于一个动作事件，可以通过使用e.getWhen()方法获得该事件的发生时间。

表16-2 事件、事件监听器和监听器方法

| 事件类（处理器） | 监听器接口 | 监听器方法 |
|---|---|---|
| ActionEvent | ActionListener | actionPerformed(ActionEvent) |
| ItemEvent | ItemListener | itemStateChanged(ItemEvent) |
| MouseEvent | MouseListener | mousePressed(MouseEvent) |
| | | mouseReleased(MouseEvent) |
| | | mouseEntered(MouseEvent) |
| | | mouseExited(MouseEvent) |
| | | mouseClicked(MouseEvent) |
| | MouseMotionListener | mouseDragged(MouseEvent) |
| | | mouseMoved(MouseEvent) |
| KeyEvent | KeyListener | keyPressed(KeyEvent) |
| | | keyReleased(KeyEvent) |
| | | keyTyped(KeyEvent) |
| WindowEvent | WindowListener | windowClosing(WindowEvent) |
| | | windowOpened (WindowEvent) |
| | | windowIconified(WindowEvent) |
| | | windowDeiconified(WindowEvent) |
| | | windowClosed(WindowEvent) |
| | | windowActivated(WindowEvent) |
| | | windowDeactivated(WindowEvent) |
| ContainerEvent | ContainerListener | componentAdded(ContainerEvent) |
| | | componentRemoved(ContainerEvent) |
| ComponentEvent | ComponentListener | componentMoved(ComponentEvent) |
| | | componentHidden(ComponentEvent) |
| | | componentResized(ComponentEvent) |
| | | componentShown(ComponentEvent) |
| FocusEvent | FocusListener | focusGained(FocusEvent) |
| | | focusLost(FocusEvent) |
| AdjustmentEvent | AdjustmentListener | adjustmentValueChanged(AdjustmentEvent) |
| ChangeEvent | ChangeListener | stateChanged(ChangeEvent) |
| ListSelectionEvent | ListSelectionListener | valueChanged(ListSelectionEvent) |




a）通用源对象的内部函数　　b）JButton对象的内部函数

图16-4 源对象通过调用监听器对象的处理器通知事件监听器

现在，我们可以编写一个程序，使用两个按钮控制一个圆的大小，如图16-6所示。

我们将逐步开发这个程序。首先，编写程序清单16-1中的程序，显示一个圆在中央（第14行）以及两个按钮在底部（第15行）的用户接口。

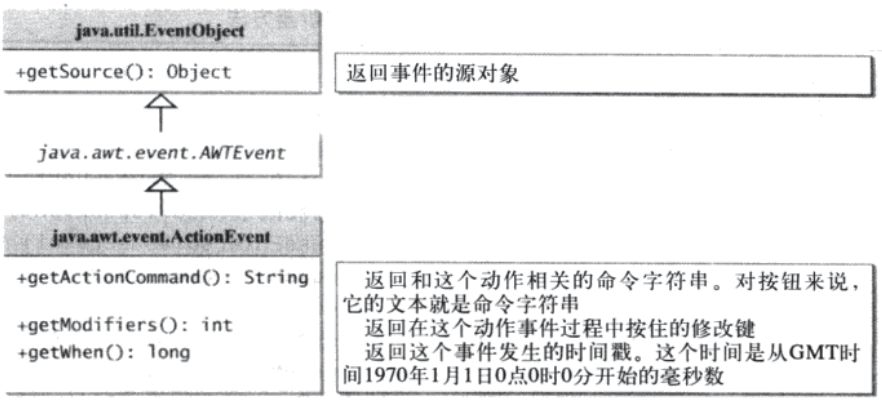



图16-5 可以从一个事件对象获取有用的信息

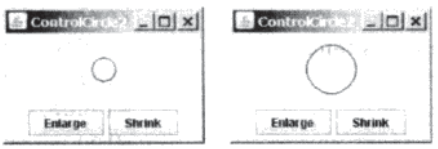



图16-6 用户点击Enlarge和Shrink按钮来放大和缩小圆的尺寸

**程序清单16-1 ControlCircle1.java**

```
1 import javax.swing.*;
2 import java.awt.*;
3
4 public class ControlCircle1 extends JFrame {
5   private JButton jbtEnlarge = new JButton("Enlarge");
6   private JButton jbtShrink = new JButton("Shrink");
7   private CirclePanel canvas = new CirclePanel();
8
9   public ControlCircle1() {
10    JPanel panel = new JPanel(); // Use the panel to group buttons
11    panel.add(jbtEnlarge);
12    panel.add(jbtShrink);
13
14    this.add(canvas, BorderLayout.CENTER); // Add canvas to center
15    this.add(panel, BorderLayout.SOUTH); // Add buttons to the frame
16  }
17
18  /** Main method */
19  public static void main(String[] args) {
20    JFrame frame = new ControlCircle1();
21    frame.setTitle("ControlCircle1");
22    frame.setLocationRelativeTo(null); // Center the frame
23    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
24    frame.setSize(200, 200);
25    frame.setVisible(true);
26  }
27 }
28
29 class CirclePanel extends JPanel {
30  private int radius = 5; // Default circle radius
31
32  /** Repaint the circle */
33  protected void paintComponent(Graphics g) {
34    super.paintComponent(g);
35    g.drawOval(getWidth() / 2 - radius, getHeight() / 2 - radius,
36      2 * radius, 2 * radius);
37  }
38 }
```

如何使用按钮放大和缩小这个圆呢？当点击Enlarge按钮时，希望能用一个比较大的半径来重新绘制这个圆。该如何完成它呢？可以用下面的特征来将程序清单16-1中的程序扩展到程序清单16-2中：

1）定义一个名为EnlargeListener的监听器类，实现ActionListener（第31～35行）。

2）创建一个监听器，并且将它注册到jbtEnlarge（第18行）。

3）在CirclePanel中添加一个名为enlarge()的方法来增加半径，然后重新绘制面板（第41～44行）。

4）实现EnlargeListener中的actionPerformed方法来调用canvas.enlarge()（第33行）。

5）为了让actionPerformed方法可以访问引用变量canvas，将EnlargeListener定义为ControlCircle2类的内部类（第31～35行）。内部类定义在另一个类中。我们将在16.4节介绍内部类。

6）为了避免编译错误，CirclePanel类（第37～52行）现在也定义为ControlCircle2的一个内部类，因为旧的CirclePanel类已经在程序清单16-1中定义。

**程序清单16-2 ControlCircle2.java**

```
1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4
5  public class ControlCircle2 extends JFrame {
6    private JButton jbtEnlarge = new JButton("Enlarge");
7    private JButton jbtShrink = new JButton("Shrink");
8    private CirclePanel canvas = new CirclePanel();
9
10   public ControlCircle2() {
11     JPanel panel = new JPanel(); // Use the panel to group buttons
12     panel.add(jbtEnlarge);
13     panel.add(jbtShrink);
14
15     this.add(canvas, BorderLayout.CENTER); // Add canvas to center
16     this.add(panel, BorderLayout.SOUTH); // Add buttons to the frame
17
18     jbtEnlarge.addActionListener(new EnlargeListener());
19   }
20
21   /** Main method */
22   public static void main(String[] args) {
23     JFrame frame = new ControlCircle2();
24     frame.setTitle("ControlCircle2");
25     frame.setLocationRelativeTo(null); // Center the frame
26     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
27     frame.setSize(200, 200);
28     frame.setVisible(true);
29   }
30
31   class EnlargeListener implements ActionListener {  // Inner class
32     public void actionPerformed(ActionEvent e) {
33       canvas.enlarge();
34     }
35   }
36
37   class CirclePanel extends JPanel { // Inner class
38     private int radius = 5; // Default circle radius
39
40     /** Enlarge the circle */
41     public void enlarge() {
42       radius++;
43       repaint();
44     }
45
46     /** Repaint the circle */
47     protected void paintComponent(Graphics g) {
48       super.paintComponent(g);
```

```
49        g.drawOval(getWidth() / 2 - radius, getHeight() / 2 - radius,
50          2 * radius, 2 * radius);
51      }
52    }
53 }
```

同样地，可以添加Shrink按钮的代码，当点击Shrink按钮时，显示比较小的圆。

## 16.4 内部类

内部类（inner class）或者嵌套类（nested class）是定义在另一个类的范围内的类。图16-7a中的代码定义了两个独立的类：Test和A。图16-7b中的代码定义A为Test的一个内部类。

```
public class Test {
  ...
}

public class A {
  ...
}
```

a)

```
public class Test {
  ...
  // Inner class
  public class A {
  ...
  }
}
```

b)

```
// OuterClass.java: inner class demo
public class OuterClass {
  private int data;

  /** A method in the outer class */
  public void m() {
    // Do something
  }

  // An inner class
  class InnerClass {
    /** A method in the inner class */
    public void mi() {
      // Directly reference data and method
      // defined in its outer class
      data++;
      m();
    }
  }
}
```

c)

图16-7 内部类结合相关类成为主类

图16-7c中定义在OuterClass中的InnerClass类是内部类的另一个例子。内部类可以像常规类一样使用。通常，如果内部类只是被外部类使用，那就将该类定义为内部类。一个内部类有如下特征：

- 一个内部类被编译成一个名为OuterClassName$InnerClassName.class的类。例如，在图16-7b中Test中的内部类A被编译为Test$A.class。
- 内部类可以引用定义在它嵌套的外部类中的数据和方法，所以，不需要将外部类对象的引用传递给内部类的构造方法。因为这个原因，内部类可以使程序更加简单和简洁。
- 使用可见性修饰符定义内部类时，遵从和应用与在类成员上一样的可见性规则。
- 可以将内部类定义为static。一个static内部类可以使用外部类的名字访问。一个static类是不能访问外部类的非静态成员的。
- 内部类的对象经常在外部类中创建。但是，也可以从另一个类中创建一个内部类的对象。如果该内部类是非静态的，就必须先创建一个外部类的实例，然后使用下面的语法创建一个内部类的对象：

```
OuterClass.InnerClass innerObject = outerObject.new InnerClass();
```

- 如果内部类是静态的，那么使用下面的语法为它创建一个对象：

```
OuterClass.InnerClass innerObject = new OuterClass.InnerClass();
```

内部类的一个简单应用就是将从属类合并到主类中。这可以减少源文件的数量。它也可以使类文件更易于组织，因为所有的类文件都是以主类作为前缀命名的。例如，不是创建图16-7a的两个源文件Test.java和A.java，而是将类A合并到类Test中，只创建一个源文件Test.java，如图16-7b所示。最终的类文件是Test.class和Test$A.class。

内部类的另一个实际应用就是为了避免类命名冲突。在程序清单16-1和程序清单16-2中定义了两个版本的CirclePanel。可以将它们定义为内部类，以避免冲突。

## 16.5 匿名类监听器

监听器类是特意为创建一个GUI组件（例如，一个按钮）而设计的监听器对象。监听器类不被其他应用程序所共享，因此，正确的做法是将它作为一个内部类定义在框架类中。

可以使用匿名内部类简化内部类监听器。匿名内部类（anonymous inner class）是没有名字的内部类。它一步完成定义内部类和创建一个该类的实例。程序清单16-2的内部类可以用匿名内部类替换，如下所示：

```
public ControlCircle2() {
  // Omitted

  jbtEnlarge.addActionListener(
    new EnlargeListener());
}

class EnlargeListener
    implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    canvas.enlarge();
  }
}
```

a）内部类EnlargeListener

```
public ControlCircle2() {
  // Omitted

  jbtEnlarge.addActionListener(
    new class EnlargeListener
        implements ActionListener() {
      public void
          actionPerformed(ActionEvent e) {
        canvas.enlarge();
      }
    });
}
```

b）匿名内部类

匿名内部类的语法如下：

```
new SuperClassName/InterfaceName() {
  // Implement or override methods in superclass or interface

  // Other methods if necessary
}
```

由于匿名内部类是一种特殊的内部类，所以，可以将它看做有以下特征的内部类：

- 匿名内部类必须总是扩展父类或实现接口，但是它不能有显式的`extends`或者`implements`子句。
- 匿名内部类必须实现父类或接口中所有的抽象方法。
- 匿名内部类总是使用它的父类的无参数构造方法来创建实例。如果匿名内部类实现了接口，构造方法就是`Object()`。
- 匿名内部类被编译为一个名为`OuterClassName$n.class`的类。例如，如果外部类`Test`有两个匿名内部类，那么它们就被编译为`Test$1.class`和`Test$2.class`。

程序清单16-3给出处理来自四个按钮的事件的例子，如图16-8所示。

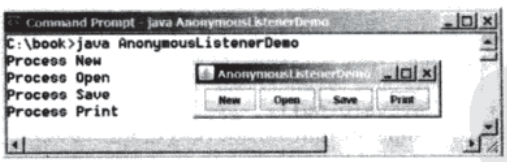



图16-8 程序处理来自四个按钮的事件

**程序清单16-3 AnonymousListenerDemo.java**

```
 1 import javax.swing.*;
 2 import java.awt.event.*;
 3
 4 public class AnonymousListenerDemo extends JFrame {
 5   public AnonymousListenerDemo() {
 6     // Create four buttons
 7     JButton jbtNew = new JButton("New");
 8     JButton jbtOpen = new JButton("Open");
 9     JButton jbtSave = new JButton("Save");
10     JButton jbtPrint = new JButton("Print");
11
```

```
12      // Create a panel to hold buttons
13      JPanel panel = new JPanel();
14      panel.add(jbtNew);
15      panel.add(jbtOpen);
16      panel.add(jbtSave);
17      panel.add(jbtPrint);
18
19      add(panel);
20
21      // Create and register anonymous inner-class listener
22      jbtNew.addActionListener(
23        new ActionListener() {
24          public void actionPerformed(ActionEvent e) {
25            System.out.println("Process New");
26          }
27        }
28      );
29
30      jbtOpen.addActionListener(
31        new ActionListener() {
32          public void actionPerformed(ActionEvent e) {
33            System.out.println("Process Open");
34          }
35        }
36      );
37
38      jbtSave.addActionListener(
39        new ActionListener() {
40          public void actionPerformed(ActionEvent e) {
41            System.out.println("Process Save");
42          }
43        }
44      );
45
46      jbtPrint.addActionListener(
47        new ActionListener() {
48          public void actionPerformed(ActionEvent e) {
49            System.out.println("Process Print");
50          }
51        }
52      );
53    }
54
55    /** Main method */
56    public static void main(String[] args) {
57      JFrame frame = new AnonymousListenerDemo();
58      frame.setTitle("AnonymousListenerDemo");
59      frame.setLocationRelativeTo(null); // Center the frame
60      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
61      frame.pack();
62      frame.setVisible(true);
63    }
64  }
```

程序使用匿名内部类创建四个监听器（第22～52行）。不使用匿名内部类，就必须创建四个独立的类。匿名监听器和内部类监听器的工作方式是一样的。使用匿名内部类压缩这个程序。

匿名内部类被编译为`OuterClassName$#.class`，这里的#是从1开始的，每次编译器遇到匿名类它就会自增1。在这个例子中，匿名内部类就被编译为`AnonymousListenerDemo$1.class`、`AnonymousListenerDemo$2.class`、`AnonymousListenerDemo$3.class`和`AnonymousListenerDemo$4.class`。

程序没有使用`setSize`方法来设置框架的大小，而是使用`pack()`方法（第61行），该方法会依据放在框架中组件的大小来自动调整框架的大小。

## 16.6 定义监听器类的另一种方式

定义监听器类有很多其他方式。例如，可以改写程序清单16-3，新程序只创建一个监听器，将这个监听器注册给按钮，然后让监听器检测出事件源，即哪个按钮触发了这个事件，如程序清单16-4所示。

**程序清单16-4 DetectSourceDemo.java**

```
1 import javax.swing.*;
2 import java.awt.event.*;
3
4 public class DetectSourceDemo extends JFrame {
5   // Create four buttons
6   private JButton jbtNew = new JButton("New");
7   private JButton jbtOpen = new JButton("Open");
8   private JButton jbtSave = new JButton("Save");
9   private JButton jbtPrint = new JButton("Print");
10
11  public DetectSourceDemo() {
12    // Create a panel to hold buttons
13    JPanel panel = new JPanel();
14    panel.add(jbtNew);
15    panel.add(jbtOpen);
16    panel.add(jbtSave);
17    panel.add(jbtPrint);
18
19    add(panel);
20
21    // Create a listener
22    ButtonListener listener = new ButtonListener();
23
24    // Register listener with buttons
25    jbtNew.addActionListener(listener);
26    jbtOpen.addActionListener(listener);
27    jbtSave.addActionListener(listener);
28    jbtPrint.addActionListener(listener);
29  }
30
31  class ButtonListener implements ActionListener {
32    public void actionPerformed(ActionEvent e) {
33      if (e.getSource() == jbtNew)
34        System.out.println("Process New");
35      else if (e.getSource() == jbtOpen)
36        System.out.println("Process Open");
37      else if (e.getSource() == jbtSave)
38        System.out.println("Process Save");
39      else if (e.getSource() == jbtPrint)
40        System.out.println("Process Print");
41    }
42  }
43
44  /** Main method */
45  public static void main(String[] args) {
46    JFrame frame = new DetectSourceDemo();
47    frame.setTitle("DetectSourceDemo");
48    frame.setLocationRelativeTo(null); // Center the frame
49    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
50    frame.pack();
51    frame.setVisible(true);
52  }
53 }
```

这个程序只定义了一个内部监听器类（第31～42行），从该类中创建一个监听器（第22行），然后将它注册给四个按钮（第25～28行）。当点击一个按钮时，该按钮就触发一个ActionEvent，然后调用监听器的actionPerformed方法。actionPerformed方法使用事件的getSource()方法来检查事件源

(第33、35、37、39行)，并且确定哪个按钮触发了这个事件。

也可以定义一个自定制框架类实现ActionListener，以改写程序清单16-3，如程序清单16-5所示。

**程序清单16-5 FrameAsListenerDemo.java**

```
1 import javax.swing.*;
2 import java.awt.event.*;
3
4 public class FrameAsListenerDemo extends JFrame
5     implements ActionListener {
6   // Create four buttons
7   private JButton jbtNew = new JButton("New");
8   private JButton jbtOpen = new JButton("Open");
9   private JButton jbtSave = new JButton("Save");
10  private JButton jbtPrint = new JButton("Print");
11
12  public FrameAsListenerDemo() {
13    // Create a panel to hold buttons
14    JPanel panel = new JPanel();
15    panel.add(jbtNew);
16    panel.add(jbtOpen);
17    panel.add(jbtSave);
18    panel.add(jbtPrint);
19
20    add(panel);
21
22    // Register listener with buttons
23    jbtNew.addActionListener(this);
24    jbtOpen.addActionListener(this);
25    jbtSave.addActionListener(this);
26    jbtPrint.addActionListener(this);
27  }
28
29  /** Implement actionPerformed */
30  public void actionPerformed(ActionEvent e) {
31    if (e.getSource() == jbtNew)
32      System.out.println("Process New");
33    else if (e.getSource() == jbtOpen)
34      System.out.println("Process Open");
35    else if (e.getSource() == jbtSave)
36      System.out.println("Process Save");
37    else if (e.getSource() == jbtPrint)
38      System.out.println("Process Print");
39  }
40
41  /** Main method */
42  public static void main(String[] args) {
43    JFrame frame = new FrameAsListenerDemo();
44    frame.setTitle("FrameAsListenerDemo");
45    frame.setLocationRelativeTo(null); // Center the frame
46    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
47    frame.pack();
48    frame.setVisible(true);
49  }
50 }
```

框架类扩展JFrame并实现ActionListener（第5行）。因此，该类是一个动作事件的监听器类。该监听器注册给四个按钮（第23～26行）。点击按钮时，该按钮触发一个ActionEvent，然后调用监听器的actionPerformed方法。actionPerformed方法使用事件的getSource()方法来检查事件的源（第31、33、35、37行），并确定哪个按钮触发了这个事件。

这种设计不是很好，因为它将太多的责任放在了一个类中。最好能设计一个监听器类，它只负责处理事件。这种设计会让代码易于阅读和维护。

可以用多种方式定义监听器类。哪种方式比较好呢？使用内部类或者匿名内部类定义监听器类已经成为事件处理程序设计的标准，因为它通常都能提供清晰、干净和简洁的代码。因此，我们将在这本书里持续使用它。

## 16.7 问题：贷款计算器

现在，我们可以编写解决本章的前言部分给出的贷款计算器问题的代码。下面是这个程序的主要步骤：

（1）创建一个如图16-9所示的用户接口

1）创建一个5行2列的GridLayout面板。将标签和文本域添加到面板中。将面板标题设置为“Enter loan amount，interest rate，and years”（请输入贷款总额、利率和年份）。

2）创建另一个带FlowLayout(FlowLayout.RIGHT)的面板，并将一个按钮添加到这个面板。

3）将第一个面板添加到框架的中央，将第二个面板添加到框架的南边。

（2）处理事件

创建和注册用以处理点击按钮动作事件的监听器。处理器获取关于贷款总额、利率和年数的输入，计算月偿还额和总偿还额，然后在文本域显示这些值。

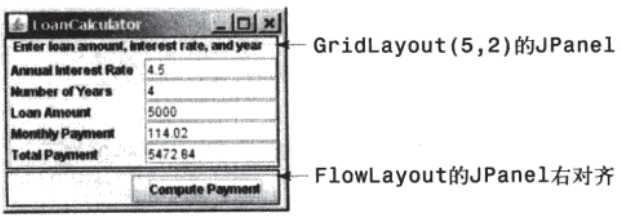



图16-9 程序计算贷款偿还额

程序清单16-6给出完整的程序。

**程序清单16-6 LoanCalculator.java**

```
1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4 import javax.swing.border.TitledBorder;
5
6 public class LoanCalculator extends JFrame {
7   // Create text fields for interest rate,
8   // year, loan amount, monthly payment, and total payment
9   private JTextField jtfAnnualInterestRate = new JTextField();
10  private JTextField jtfNumberOfYears = new JTextField();
11  private JTextField jtfLoanAmount = new JTextField();
12  private JTextField jtfMonthlyPayment = new JTextField();
13  private JTextField jtfTotalPayment = new JTextField();
14
15  // Create a Compute Payment button
16  private JButton jbtComputeLoan = new JButton("Compute Payment");
17
18  public LoanCalculator() {
19    // Panel p1 to hold labels and text fields
20    JPanel p1 = new JPanel(new GridLayout(5, 2));
21    p1.add(new JLabel("Annual Interest Rate"));
22    p1.add(jtfAnnualInterestRate);
23    p1.add(new JLabel("Number of Years"));
24    p1.add(jtfNumberOfYears);
25    p1.add(new JLabel("Loan Amount"));
26    p1.add(jtfLoanAmount);
27    p1.add(new JLabel("Monthly Payment"));
28    p1.add(jtfMonthlyPayment);
```

```
29      p1.add(new JLabel("Total Payment"));
30      p1.add(jtfTotalPayment);
31      p1.setBorder(new
32        TitledBorder("Enter loan amount, interest rate, and year"));
33
34      // Panel p2 to hold the button
35      JPanel p2 = new JPanel(new FlowLayout(FlowLayout.RIGHT));
36      p2.add(jbtComputeLoan);
37
38      // Add the panels to the frame
39      add(p1, BorderLayout.CENTER);
40      add(p2, BorderLayout.SOUTH);
41
42      // Register listener
43      jbtComputeLoan.addActionListener(new ButtonListener());
44    }
45
46    /** Handle the Compute Payment button */
47    private class ButtonListener implements ActionListener {
48      public void actionPerformed(ActionEvent e) {
49        // Get values from text fields
50        double interest =
51          Double.parseDouble(jtfAnnualInterestRate.getText());
52        int year =
53          Integer.parseInt(jtfNumberOfYears.getText());
54        double loanAmount =
55          Double.parseDouble(jtfLoanAmount.getText());
56
57        // Create a loan object
58        Loan loan = new Loan(interest, year, loanAmount);
59
60        // Display monthly payment and total payment
61        jtfMonthlyPayment.setText(String.format("%.2f",
62          loan.getMonthlyPayment()));
63        jtfTotalPayment.setText(String.format("%.2f",
64          loan.getTotalPayment()));
65      }
66    }
67
68    public static void main(String[] args) {
69      LoanCalculator frame = new LoanCalculator();
70      frame.pack();
71      frame.setTitle("LoanCalculator");
72      frame.setLocationRelativeTo(null); // Center the frame
73      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
74      frame.setVisible(true);
75    }
76  }
```

构造方法创建用户界面（第18～44行）。按钮是事件的源。创建一个监视器并注册到按钮（第43行）。

监听器类（第47～66行）实现actionPerformed方法。点击按钮时，调用actionPerformed方法获取利率（第51行）、年数（第53行）以及贷款总额（第55行）。调用jtfAnnualInterestRate.getText()返回jtfAnnualInterestRate文本域中的字符串文本。贷款用来计算贷款偿还额。这个类已经在程序清单10-2中介绍过。调用loan.getMonthlyPayment()会返回这笔贷款的月偿还额。String.format方法使用类似printf语法将一个数字格式化为希望的格式。在文本域上调用setText方法可以在文本域中设置一个字符串值（第61行）。

## 16.8 窗口事件

前面几节使用的都是动作事件。也可以用相同的方式处理其他事件。本节给出一个处理

WindowEvent的例子。Window类的任何一个子类都可能触发下面的窗口事件：打开窗口、正在关闭窗口、关闭窗口、激活窗口、变成非活动窗口、最小化窗口和还原窗口。程序清单16-7中的程序创建一个框架，监听窗口事件，然后显示一条信息表明当前发生的事件。图16-10显示这个程序的一个运行示例。

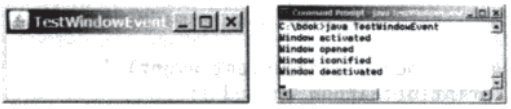



图16-10 从命令行提示符运行程序时，窗口事件就显示在控制台上

**程序清单16-7 TestWindowEvent.java**

```
1  import java.awt.event.*;
2  import javax.swing.JFrame;
3
4  public class TestWindowEvent extends JFrame {
5    public static void main(String[] args) {
6      TestWindowEvent frame = new TestWindowEvent();
7      frame.setSize(220, 80);
8      frame.setLocationRelativeTo(null); // Center the frame
9      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10     frame.setTitle("TestWindowEvent");
11     frame.setVisible(true);
12   }
13
14   public TestWindowEvent() {
15     addWindowListener(new WindowListener() {
16       /**
17        * Handler for window-deiconified event
18        * Invoked when a window is changed from a minimized
19        * to a normal state.
20        */
21       public void windowDeiconified(WindowEvent event) {
22         System.out.println("Window deiconified");
23       }
24
25       /**
26        * Handler for window-iconified event
27        * Invoked when a window is changed from a normal to a
28        * minimized state. For many platforms, a minimized window
29        * is displayed as the icon specified in the window's
30        * iconImage property.
31        */
32       public void windowIconified(WindowEvent event) {
33         System.out.println("Window iconified");
34       }
35
36       /**
37        * Handler for window-activated event
38        * Invoked when the window is set to be the user's
39        * active window, which means the window (or one of its
40        * subcomponents) will receive keyboard events.
41        */
42       public void windowActivated(WindowEvent event) {
43         System.out.println("Window activated");
44       }
45
46       /**
47        * Handler for window-deactivated event
48        * Invoked when a window is no longer the user's active
49        * window, which means that keyboard events will no longer
50        * be delivered to the window or its subcomponents.
51        */
52       public void windowDeactivated(WindowEvent event) {
```

```
53          System.out.println("Window deactivated");
54        }
55
56        /**
57         * Handler for window-opened event
58         * Invoked the first time a window is made visible.
59         */
60        public void windowOpened(WindowEvent event) {
61          System.out.println("Window opened");
62        }
63
64        /**
65         * Handler for window-closing event
66         * Invoked when the user attempts to close the window
67         * from the window's system menu.  If the program does not
68         * explicitly hide or dispose the window while processing
69         * this event, the window-closing operation will be cancelled.
70         */
71        public void windowClosing(WindowEvent event) {
72          System.out.println("Window closing");
73        }
74
75        /**
76         * Handler for window-closed event
77         * Invoked when a window has been closed as the result
78         * of calling dispose on the window.
79         */
80        public void windowClosed(WindowEvent event) {
81          System.out.println("Window closed");
82        }
83      });
84    }
85  }
```

Window类或者Window的任何子类都可能会触发WindowEvent。因为JFrame是Window的子类，所以它可以触发WindowEvent。

TestWindowEvent扩展JFrame，并且实现WindowListener。WindowListener接口定义了几个抽象方法（windowActivated、windowClosed、windowClosing、windowDeactivated、windowDeiconified、windowIconified、WindowOpened）来处理窗口事件：激活窗口、关闭窗口、正在关闭窗口、变成非活动的窗口、最小化窗口、还原窗口或者打开窗口。

当像激活窗口这样的窗口事件发生时，windowActivated方法就被触发。如果希望处理事件，就应该用一个具体响应实现windowActivated方法。

## 16.9 监听器接口适配器

因为WindowListener接口中的方法都是抽象的，所以即使程序并不关注某些事件，还是必须实现所有的方法。为了方便起见，Java提供称作方便适配器（convenience adapter）的支持类，它提供监听器接口中所有方法的默认实现。默认的实现只是一个空的程序体。Java为每个AWT监听器接口提供有多个处理器的方便监听器适配器。每个XListener的方便监听器适配器命名为XAdapter。例如，WindowAdapter是WindowListener的方便监听器适配器。表16-3列出方便适配器。

表16-3 方便适配器

| 适配器 | 接口 |
|---|---|
| WindowAdapter | WindowListener |
| MouseAdapter | MouseListener |
| MouseMotionAdpter | MouseMotionListener |
| KdyAdpter | KeyListener |
| ContainerAdpter | ContainerListener |
| ComponentAdpter | ComponentListener |
| FocusAdpter | FocusListener |

如果你只对激活窗口的事件感兴趣，那么使用WindowAdapter就可以将前面的例子简化为如程序清单16-8所示。WindowAdapter类用来创建一个匿名监听器而不是WindowListener类（第15行）。windowActivated处理器在第16行实现。

**程序清单16-8 AdapterDemo.java**

```
1 import java.awt.event.*;
2 import javax.swing.JFrame;
3
4 public class AdapterDemo extends JFrame {
5   public static void main(String[] args) {
6     AdapterDemo frame = new AdapterDemo();
7     frame.setSize(220, 80);
8     frame.setLocationRelativeTo(null); // Center the frame
9     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10    frame.setTitle("AdapterDemo");
11    frame.setVisible(true);
12  }
13
14  public AdapterDemo() {
15    addWindowListener(new WindowAdapter() {
16      public void windowActivated(WindowEvent event) {
17        System.out.println("Window activated");
18      }
19    });
20  }
21 }
```

## 16.10 鼠标事件

当在一个组件上按下、释放、点击、移动或拖动鼠标时就会产生鼠标事件。MouseEvent对象捕获这个事件，例如，获取相应的点击次数或鼠标的位置（x和y坐标）等，如图16-11所示。

因为MouseEvent类继承InputEvent类，所以可以在MouseEvent对象上使用InputEvent类中定义的方法。

java.awt.Point类表示组件上的一个点。该类包含两个用来表示坐标的公共变量x和y。为了创建一个Point，使用下面的构造方法：

```
Point(int x, int y)
```

它利用指定的x坐标和y坐标构造一个Point对象。通常，类中的数据域应该是私有的，但是，这个类具有两个公共数据域。

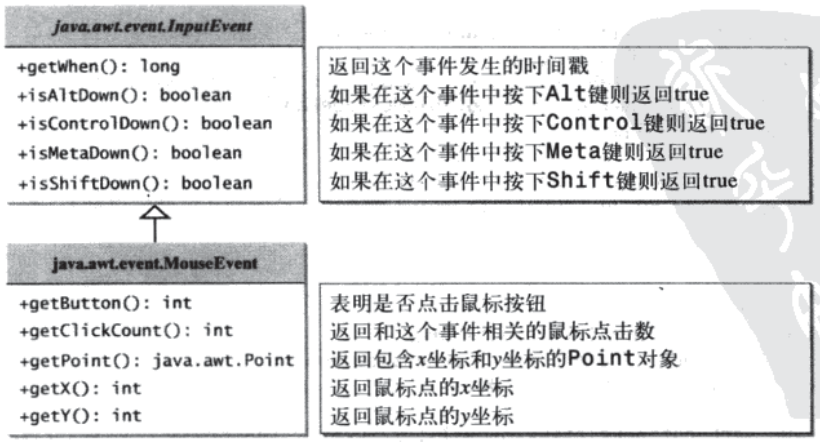



图16-11 MouseEvent类封装了鼠标事件的信息

Java提供了两个处理鼠标事件的监听器接口MouseListener和MouseMotionListener，如图16-12所示。实现MouseListener接口可以监听如鼠标的按下、释放、输入、退出或点击的动作，然后实现MouseMotionListener接口以监听如拖动鼠标或移动鼠标的动作。

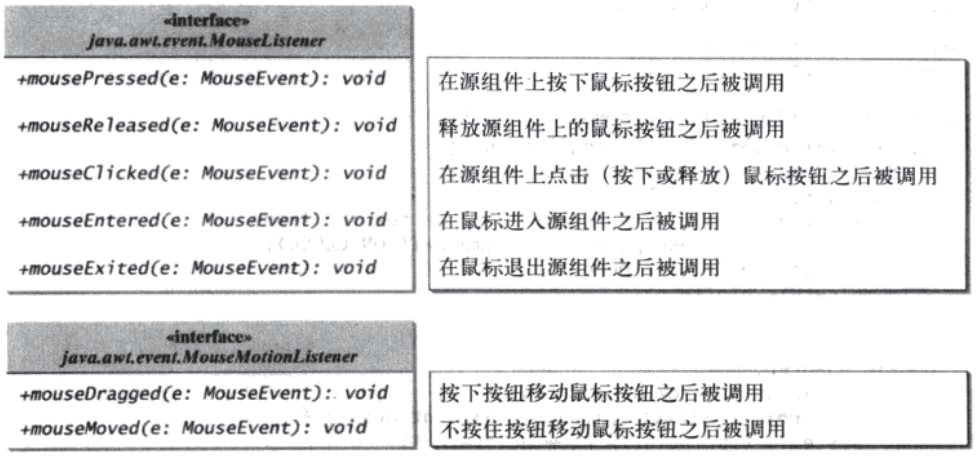



图16-12 MouseListener接口处理鼠标键按下、释放、点击、输入和退出事件。MouseMotionListener接口处理拖动和移动鼠标的事件

## 举例：使用鼠标在面板上移动消息

本例编写一个程序，在面板中显示一条信息，如清序清单16-9所示。可以用鼠标移动这条消息。信息会随着鼠标的拖动而移动，信息总是显示在鼠标指针处。图16-13显示了程序的一个运行示例。

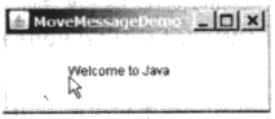



图16-13 可以拖动鼠标移动消息

**程序清单16-9 MoveMessageDemo.java**

```
1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4
5 public class MoveMessageDemo extends JFrame {
6   public MoveMessageDemo() {
7     // Create a MovableMessagePanel instance for moving a message
8     MovableMessagePanel p = new MovableMessagePanel
9       ("Welcome to Java");
10
11    // Place the message panel in the frame
12    setLayout(new BorderLayout());
13    add(p);
14  }
15
16  /** Main method */
17  public static void main(String[] args) {
18    MoveMessageDemo frame = new MoveMessageDemo();
19    frame.setTitle("MoveMessageDemo");
20    frame.setSize(200, 100);
21    frame.setLocationRelativeTo(null); // Center the frame
22    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
23      frame.setVisible(true);
24    }
25
26    // Inner class: MovableMessagePanel draws a message
27    static class MovableMessagePanel extends JPanel {
28      private String message = "Welcome to Java";
29      private int x = 20;
30      private int y = 20;
31
32      /** Construct a panel to draw string s */
33      public MovableMessagePanel(String s) {
34        message = s;
35        addMouseMotionListener(new MouseMotionAdapter() {
36          /** Handle mouse-dragged event */
37          public void mouseDragged(MouseEvent e) {
38            // Get the new location and repaint the screen
39            x = e.getX();
40            y = e.getY();
41            repaint();
42          }
43        });
44      }
45
46      /** Paint the component */
47      protected void paintComponent(Graphics g) {
48        super.paintComponent(g);
49        g.drawString(message, x, y);
50      }
51    }
52  }
```

MovableMessagePanel类扩展JPanel类来绘制消息（第27行）。除此之外，拖动鼠标时它还处理重新显示消息。这个类被定义为主类中的内部类，因为它只在这个类中使用。因此，这个内部类被定义为静态的，因为它不引用主类的任何实例成员。

MouseMotionListener接口包含两个处理器：mouseMoved和mouseDragged，它们用于处理鼠标动作事件。移动鼠标按下按钮时，调用mouseDragged方法，用于重画视图区域并在鼠标点处显示消息。不按按钮而移动鼠标时，就会调用mouseMoved方法。

因为监听器只关心鼠标的拖动事件，所以匿名内部类监听器扩展MouseMotionAdapter覆盖mouseDragged方法。如果内部类实现了MouseMotionListener接口，即使监听器并不关心某些事件，也必须实现所有的处理器。

当按下按钮移动鼠标时，就会调用mouseDragged方法。该方法利用MouseEvent类中的getX方法和getY方法（第39～40行）来获取鼠标的位置。它就成为消息的新位置。调用repaint()方法（第41行）会导致paintComponent被调用（第47行），这样，就将在新位置显示这个消息。

## 16.11 按键事件

按键事件可以利用键盘来控制和执行一些动作，或者从键盘上获取输入。只要按下、释放一个键或者在一个组件上敲击，就会触发按键事件。KeyEvent对象描述事件的特性（即按下、放开或者敲击一个键）和对应的键值，如图16-14所示。Java提供KeyListener接口来处理按键事件，如图16-15所示。

当按下一个键时，就会调用keyPressed处理器，当松开一个键时，就会调用keyReleased处理器，当输入一个统一码字符时，就会调用keyTyped处理器。如果这个键没有统一码（例如，功能键、修改键、动作键和控制键），则不会调用keyTyped处理器。

每个按键事件都有一个相关的按键字符或按键代码，它们分别由KeyEvent中的getKeyChar()和getKeyCode()方法返回。按键编码是定义在表16-4中的常量。对于统一码字符的按键，按键代码和统一码的值相同。

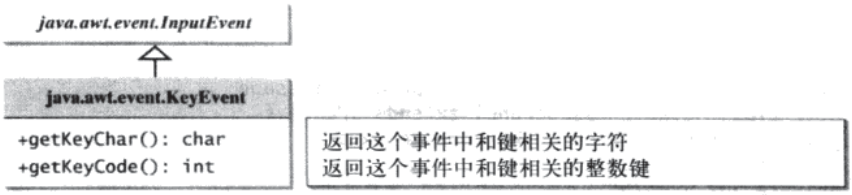



图16-14 KeyEvent类封装了关于按键事件的信息

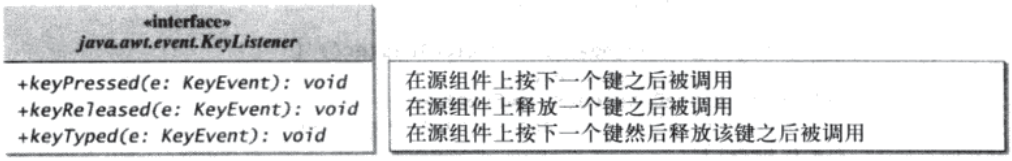



图16-15 按下、释放和敲击KeyListener接口处理键的事件

**表16-4 按键常量**

| 常 量 | 描 述 | 常 量 | 描 述 |
|---|---|---|---|
| VK_HOME | Home键 | VK_CONTROL | 控制键 |
| VK_End | End键 | VK_SHIFT | Shift键 |
| VK_PGUP | Page Up键 | VK_BACK_SPACE | 退格键 |
| VK_PGDN | Page Down键 | VK_CAPS_LOCK | 大写字母锁定键 |
| VK_UP | 上箭头 | VK_NUM_LOCK | 数字键盘锁定键 |
| VK_DOWN | 下箭头 | VK_ENTER | 回车键 |
| VK_LEFT | 左箭头 | VK_UNDEFINED | 按键代码未知 |
| VK_RIGHT | 右箭头 | VK_F1到VK_F12 | 从F1到F12的功能键 |
| VK_ESCAPE | Esc键 | VK_0到VK_9 | 从0到9的数字键 |
| VK_TAB | Tab键 | VK_A到VK_Z | 从A到Z的字母键 |

对于按下按键和释放按键事件，getKeyCode()返回定义在表中的值。对于按键敲击事件，getKeyCode()返回VK_UNDEFINED，而getKeyChar()返回输入的字符。

程序清单16-10中的程序显示用户输入的字符。用户可以使用箭头键VK_UP、VK_DOWN、VK_LEFT和VK_RIGHT向上、向下、向左、向右移动字符，程序的运行示例如图16-16所示。

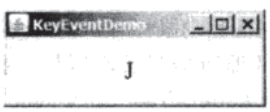



图16-16 程序通过显示一个字符以及向上、向下、向左或向右响应按键事件

**程序清单16-10 KeyEventDemo.java**

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class KeyEventDemo extends JFrame {
6    private KeyboardPanel keyboardPanel = new KeyboardPanel();
7
8    /** Initialize UI */
9    public KeyEventDemo() {
10     // Add the keyboard panel to accept and display user input
11     add(keyboardPanel);
12
13     // Set focus
14     keyboardPanel.setFocusable(true);
15   }
```

```
16
17   /** Main method */
18   public static void main(String[] args) {
19     KeyEventDemo frame = new KeyEventDemo();
20     frame.setTitle("KeyEventDemo");
21     frame.setSize(300, 300);
22     frame.setLocationRelativeTo(null); // Center the frame
23     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
24     frame.setVisible(true);
25   }
26
27   // Inner class: KeyboardPanel for receiving key input
28   static class KeyboardPanel extends JPanel {
29     private int x = 100;
30     private int y = 100;
31     private char keyChar = 'A'; // Default key
32
33     public KeyboardPanel() {
34       addKeyListener(new KeyAdapter() {
35         public void keyPressed(KeyEvent e) {
36           switch (e.getKeyCode()) {
37             case KeyEvent.VK_DOWN: y += 10; break;
38             case KeyEvent.VK_UP: y -= 10; break;
39             case KeyEvent.VK_LEFT: x -= 10; break;
40             case KeyEvent.VK_RIGHT: x += 10; break;
41             default: keyChar = e.getKeyChar();
42           }
43
44           repaint();
45         }
46       });
47     }
48
49     /** Draw the character */
50     protected void paintComponent(Graphics g) {
51       super.paintComponent(g);
52
53       g.setFont(new Font("TimesRoman", Font.PLAIN, 24));
54       g.drawString(String.valueOf(keyChar), x, y);
55     }
56   }
57 }
```

KeyboardPanel类扩展JPanel显示一个字符（第28行）。这个类定义为主类中的内部类，因为它只用在这个类中。而且，内部类要定义为静态的，因为它不引用任何主类中的实例成员。

由于程序从键盘上获取输入，因此，它监听键盘事件KeyEvent，并且扩展KeyAdapter来处理按键输入（第34行）。

当按下一个键时，就会调用keyPressed处理器。程序使用e.getKeyCode()方法获取按键代码，使用e.getKeyChar()方法获取按键的对应字符。当按下一个非箭头键时，显示它的字符（第41行）。当按下箭头键时，字符就按照箭头键指示的方向移动（第37～40行）。

仅有一个焦点的组件能够接收KeyEvent。要使一个组件成为聚焦的，需要将属性isFocusable设置为true（第14行）。

每次重新绘制组件时，都会在第53行为Graphics对象创建一种新字体。这是不够的。最好一次就将字体创建为一个数据域。

## 16.12 使用Timer类的动画

并非所有的源对象都是GUI组件。类javax.swing.Timer就是一个按照预定频率触发

ActionEvent事件的源组件。图16-17列出了这个类中的一些方法。

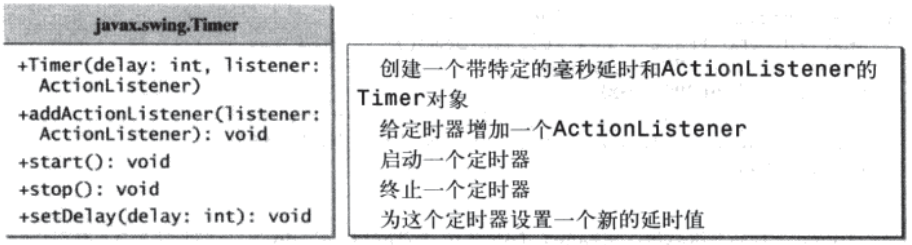



图16-17 Timer对象以固定频率触发ActionEvent事件

一个Timer对象可以作为ActionEvent事件的源。监听器必须是ActionListener的实例并且要注册到Timer对象。利用延时和监听器，可以使用Timer类的唯一一个构造方法创建一个Timer对象，其中延时是指两个动作事件之间间隔的毫秒数。可以使用addActionListener方法添加其他的监听器，用setDelay方法调整延时。要启动定时器，调用start()方法。要终止定时器时，调用stop()方法。

Timer类可以用于控制动画。例如，可以利用它显示一条移动的消息，如图16-18所示，其代码在程序清单16-11中。

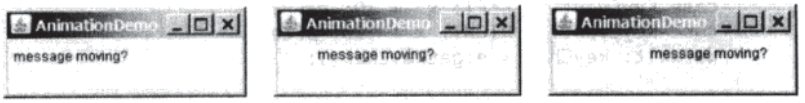



图16-18 消息在面板上移动

**程序清单16-11 AnimationDemo.java**

```
1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4
5 public class AnimationDemo extends JFrame {
6   public AnimationDemo() {
7     // Create a MovingMessagePanel for displaying a moving message
8     add(new MovingMessagePanel("message moving?"));
9   }
10
11  /** Main method */
12  public static void main(String[] args) {
13    AnimationDemo frame = new AnimationDemo();
14    frame.setTitle("AnimationDemo");
15    frame.setSize(280, 100);
16    frame.setLocationRelativeTo(null); // Center the frame
17    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
18    frame.setVisible(true);
19  }
20
21  // Inner class: Displaying a moving message
22  static class MovingMessagePanel extends JPanel {
23    private String message = "Welcome to Java";
24    private int xCoordinate = 0;
25    private int yCoordinate = 20;
26
27    public MovingMessagePanel(String message) {
28      this.message = message;
29
30      // Create a timer
31      Timer timer = new Timer(1000, new TimerListener());
32      timer.start();
33    }
34
```

```
35     /** Paint message */
36     protected void paintComponent(Graphics g) {
37       super.paintComponent(g);
38
39       if (xCoordinate > getWidth()) {
40         xCoordinate = -20;
41       }
42       xCoordinate += 5;
43       g.drawString(message, xCoordinate, yCoordinate);
44     }
45
46     class TimerListener implements ActionListener {
47       /** Handle ActionEvent */
48       public void actionPerformed(ActionEvent e) {
49         repaint();
50       }
51     }
52   }
53 }
```

MovingMessagePanel类扩展JPanel以显示一条消息（第22行）。这个类定义为主类中的一个内部类，因为只在这个类中使用它。而且，因为这个内部类不能引用主类的任意实例成员，所以，要把这个内部类定义为静态的。

一个内部类监听器在第46行定义为监听ActionEvent。第31行为监听器创建一个Timer类。这个定时器从第32行开始。每秒钟定时器都会触发一个ActionEvent，而监听器会在第49行响应来重新绘制面板。当绘制好了一个面板，它的x坐标就会增加（第42行），因此，消息会向右显示。当x坐标超过了面板的范围，它会被重置为-20（第40行），所以，消息继续从左向右移动。

在15.12节中，绘制一个显示当前时间的StillClock。但是，时钟显示过后就不再走针了。如何才能使时钟每秒钟都能显示新的当前时间呢？使时钟走针的关键是每秒都用新的当前时间来重新绘制这个时钟。可以用程序清单16-12中的代码使用定时器控制时钟的重新绘制。

**程序清单16-12 ClockAnimation.java**

```
1 import java.awt.event.*;
2 import javax.swing.*;
3
4 public class ClockAnimation extends JFrame {
5   private StillClock clock = new StillClock();
6
7   public ClockAnimation() {
8     add(clock);
9
10    // Create a timer with delay 1000 ms
11    Timer timer = new Timer(1000, new TimerListener());
12    timer.start();
13  }
14
15  private class TimerListener implements ActionListener {
16    /** Handle the action event */
17    public void actionPerformed(ActionEvent e) {
18      // Set new time and repaint the clock to display current time
19      clock.setCurrentTime();
20      clock.repaint();
21    }
22  }
23
24  /** Main method */
25  public static void main(String[] args) {
26    JFrame frame = new ClockAnimation();
27    frame.setTitle("ClockAnimation");
28    frame.setSize(200, 200);
29    frame.setLocationRelativeTo(null); // Center the frame
```

```
30      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
31      frame.setVisible(true);
32    }
33 }
```

程序显示一个正在运转的时钟，如图16-19所示。ClockAnimation创建一个StillClock（第5行）。第11行为ClockAnimation创建一个Timer。这个定时器从第12行开始。定时器每秒钟触发一个ActionEvent，而监听器响应设置新时间（第19行）并重新绘制时钟（第20行）。定义在StillClock中的setCurrentTime()方法设置时钟的当前时间。

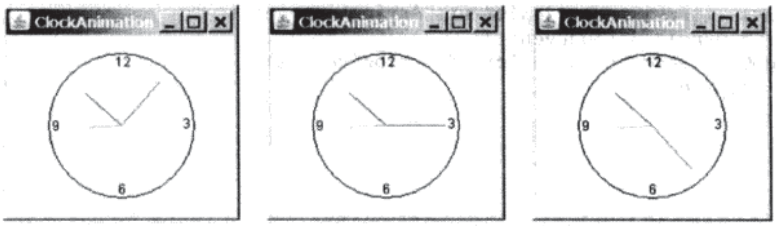



图16-19 在面板上显示一个运转的时钟

## 关键术语

anonymous inner class（匿名内部类）
convenience listener adapter（方便监听器适配器）
event（事件）
event delegation（事件委托）
event handler（事件处理器）
event listener（事件监听器）
event-listener interface（事件监听器接口）
event object（事件对象）
event registration（事件注册）
event source（source object）（事件源（源对象））
event-driven programming（事件驱动的程序设计）
inner class（内部类）

## 本章小结

- 事件类的根类是java.util.EventObject。EventObject的子类处理各种特殊类型的事件，例如，动作事件、窗口事件、组件事件、鼠标事件和按键事件。可以使用EventObject类中的getSource()实例方法判断事件的源对象。如果一个组件能够触发某个事件，那么它的所有子类都能触发同类型的事件。
- 监听器对象的类必须实现相应的事件监听器接口。Java语言为每种事件类提供监听器接口。XEvent的监听器接口通常命名为XListener，但是，MouseMotionListener除外。例如，ActionEvent对应的监听器接口是ActionListener；每个ActionEvent的监听器都应该实现ActionListener接口。监听器接口包含称为处理器的处理事件的方法。
- 监听器对象必须由源对象注册。注册方法依赖于事件的类型。对ActionEvent来讲，注册方法是addActionListener。一般来说，XEvent的方法命名为addXListener。
- 内部类或者嵌套类是定义在另一个类中的类。内部类可以引用定义在它嵌套的外部类中的数据和方法，所以，不需要将外部类的引用传递给内部类的构造方法。
- 方便适配器是能够提供监听器接口中所有方法默认实现的支持类。Java为每个AWT监听器接口提供带多个处理器的方便监听器适配器。XListener的方便监听器适配器命名为XAdapter。
- 一个源对象可以触发几种类型的事件。对每种事件，源对象维护一个注册的监听器列表，通过调用监听器对象的处理器，通知所有已经注册的监听器去处理事件。
- 在一个组件上点击、释放、移动或拖动鼠标时就会触发鼠标事件。鼠标事件对象捕获事件，例如，

和事件相关的点击次数或鼠标点的位置（x和y坐标）。

- Java提供两个处理鼠标事件的监听器接口：MouseListener和MouseMotionListener来处理事件，实现MouseListener接口来监听诸如按下、释放、点击、输入或退出鼠标等动作，实现MouseMotionListener接口来监听诸如移动或拖动鼠标的动作。
- KeyEvent对象描述事件的性质（即按下、释放或敲击一个键）以及相对应的键值。
- 当按下按键时就会调用keyPressed处理器，当释放按键时就会调用keyReleased处理器，而当敲入一个统一码字符键时，就会调用keyTyped处理器。如果某个按键没有统一码（例如，功能键、修改键、动作键和控制键），则不会调用keyTyped处理器。
- 可以使用Timer类控制Java的动画。定时器以固定频率触发ActionEvent。监听器通过更新画面来模拟动画。

## 复习题

### 16.2～16.3节

16.1 一个按钮能够触发WindowEvent吗？一个按钮能够触发MouseEvent吗？一个按钮能够触发ActionEvent吗？

16.2 为什么监听器必须是一个合适的监听器接口的实例？解释如何注册一个监听器对象以及如何实现一个监听器接口。

16.3 一个源可以有多个监听器吗？一个监听器可以监听多个源吗？一个源可以是自身的监听器吗？

16.4 如何实现定义在监听器接口中的方法？要实现所有定义在监听器接口中的方法吗？

### 16.4～16.9节

16.5 内部类可以用在不是嵌套它的其他类中吗？

16.6 修饰符public、private和static可以用在内部类吗？

16.7 如果类A是类B的内部类，那么A的.class文件是什么？如果类B包括两个匿名内部类，那么这两个类的.class文件名是什么？

16.8 下面代码有何错误？

```
import java.swing.*;
import java.awt.*;

public class Test extends JFrame {
  public Test() {
    JButton jbtOK = new JButton("OK");
    add(jbtOK);
  }

  private class Listener
      implements ActionListener {
    public void actionPerform
        (ActionEvent e) {
      System.out.println
        (jbtOK.getActionCommand());
    }
  }

  /** Main method omitted */
}
```

a)

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

public class Test extends JFrame {
  public Test() {
    JButton jbtOK = new JButton("OK");
    add(jbtOK);
    jbtOK.addActionListener(
      new ActionListener() {
        public void actionPerformed
            (ActionEvent e) {
          System.out.println
            (jbtOK.getActionCommand());
        }
    }  // Something missing here
  }

  /** Main method omitted */
}
```

b)

16.9 JFrame中的方法setSize(width,height)和方法pack()有什么不同？

### 16.10～16.11节

16.10 使用什么方法可以获取事件的源？使用什么方法获取动作事件、鼠标事件或者按键事件的时间戳？使用什么方法获取鼠标事件的鼠标点位置？使用什么方法获取按键事件的按键字符？

16.11 鼠标的按下、释放、点击、进入和退出的监听器接口是什么？鼠标移动和拖动的监听器接口是什么？

16.12 键盘上的每个键是否都有统一码？KeyEvent类中的按键编码是否等价于统一码？

16.13 按下一个键之后是否调用keyPressed处理器？释放一个键之后是否调用keyReleased处理器？在敲击任意一个键之后是否调用keyTyped处理器？

**16.12节**

16.14 如何创建一个定时器？如何启动定时器？如何停止定时器？

16.15 Timer类是否有无参构造方法？能将多个监听器添加到定时器吗？

## 编程练习题

**16.2～16.9节**

16.1 （在控制台上找出点击了哪个按钮）通过添加代码给练习题12.1，在控制台上显示消息，指出点击了哪个按钮。

16.2 （使用ComponentEvent）任何GUI组件都可以触发一个ComponentEvent。ComponentListener定义componentMoved、componentResized、componentShown和componentHidden方法来处理组件事件。编写测试程序来演示ComponentEvent。

*16.3 （移动小球）编写一个程序，在面板上移动小球。应该定义一个面板类来显示小球，并提供向左、向右、向上和向下移动小球的方法，如图16-20a所示。

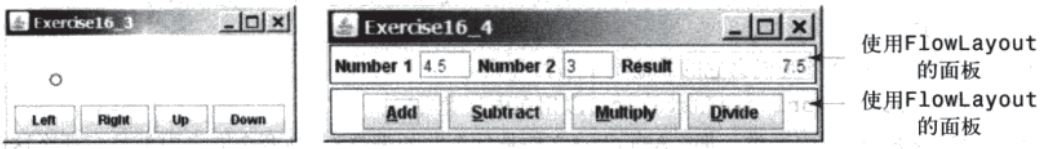


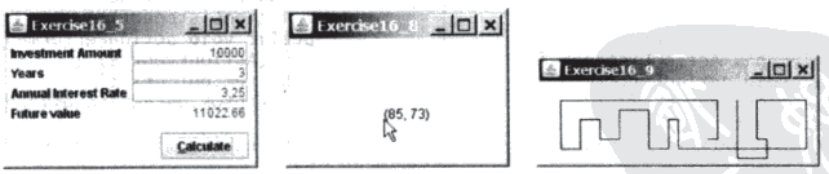

图16-20 a）练习题16.3使用按钮来移动小球；b）程序完成double数字上加法、减法、乘法和除法

*16.4 （创建一个简单的计算器）编写一个程序完成加法、减法、乘法和除法操作（参见图16-20b）。

*16.5 （创建一个投资值计算器）编写一个程序，计算投资值在给定利率以及给定年数下的未来值。计算的公式如下所示：

$$futureValue = investmentAmount * (1 + monthlyInterestRate)^{years*12}$$ （未来值=投资值×(1+月利率)^(年数×12)）

使用文本域显示利率、投资总额和年数。当用户点击Calculate按钮时在文本域显示未来的总额，如图16-21a所示。




a） b） c）

图16-21 a）用户输入投资总额、年数和利率来计算未来值；b）练习题16.8显示鼠标的位置；c）练习题16.9使用箭头键绘制直线

**16.10节**

**16.6 （两个消息交替出现）编写一个程序，当点击鼠标时，面板上交替显示两个消息“Java is fun”和“Java is powerful”。

*16.7 （使用鼠标设置背景色）编写一个程序，显示一个面板的背景色，当按下鼠标键时背景色为黑色，释放鼠标时背景色为白色。

*16.8 （显示鼠标的位置）编写两个程序，一个显示点击鼠标时鼠标的位置（参见图16-21a），而另一个显示当按下鼠标时显示鼠标的位置，当释放鼠标时停止显示。

**16.11节**

*16.9 （使用箭头键画线）编写一个程序，使用箭头键绘制线段。所画的线从框架的中心开始，当敲击向右、向上、向左或向下的箭头键时，相应地向东、向北、向西或向南方向画线，如图16-21c所示。

**16.10 （输入并显示字符串）编写一个程序，从键盘接收一个字符串并把它显示在面板上。回车键表明字符的结束。当输入一个新字符串时，会将它显示在面板上。

*16.11 （显示一个字符）编写一个程序，从键盘获得一个字符输入，然后将这个字符显示在鼠标点所在的位置。

**16.12节**

**16.12 （显示一个转动的风扇）程序清单15-4显示了一个静止的风扇。编写程序显示一个转动的风扇。

**16.13 （播放幻灯）25张幻灯片都以图像文件（slide0.jpg，slide1.jpg，...，slide24.jpg）的形式存储在图像目录中，在本书的源代码中可以下载。每个图像的大小都是800 × 600像素。编写一个Java应用程序，自动重复显示这些幻灯片。每秒显示一张幻灯片。幻灯片按顺序显示。当显示完最后一张幻灯片时，第一张幻灯片重复显示，依此类推。

**提示** 在框架中放置一个标签，并且在标签中将幻灯片设置为图像图标。

**16.14 （升旗）编写一个Java程序，用动画实现升旗，如图16-1所示（参见15.11节中关于如何显示图像的内容）。

**16.15 （赛车）编写一个Java程序，模拟汽车比赛，如图16-22a所示。汽车从左向右移动。当它到达右终点，就从左边重新开始，然后继续同样的过程。可以使用定时器控制动画。使用新的基坐标（x，y）重新绘制汽车，如图16-22b所示。

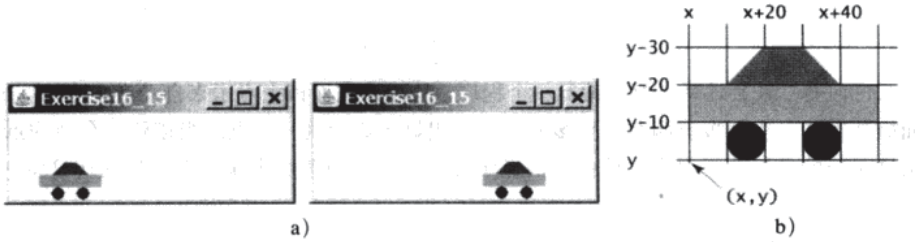



图16-22 a）练习题16.15显示一个移动的汽车；b）可以使用一个新的基点重画汽车

*16.16 （显示一个闪烁的标签）编写一个显示闪烁标签的程序。

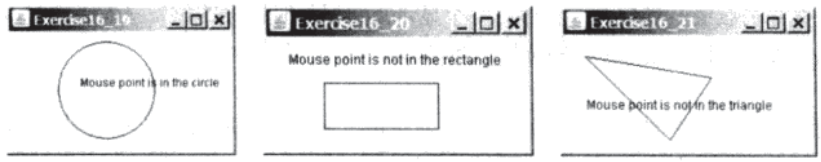
**提示** 为了使标签闪烁，需要以固定频率交替地重画带标签的面板和不带标签的面板（黑屏）。使用一个boolean变量来控制交替显示。

*16.17 （控制一个移动的标签）修改程序清单16-11，使用鼠标控制一个移动的标签。按住鼠标键时，标签停止移动，释放按钮则继续移动。

## 综合题

*16.18 （使用键移动圆）编写程序使用箭头键向上、向下、向左、向右移动一个圆。

**16.19 （几何方面：是否在圆内？）编写一个程序，绘制一个圆心在（100，60）而半径为50的圆。当鼠标移动时，显示一个消息表示鼠标点是否在圆内，如图16-23a所示。




a) b) c)

图16-23 检查一个点是否在圆内，是否在矩形内，是否在三角形内

**16.20 （几何问题：是否在矩形内？）编写一个程序，绘制一个中心在（100，60）宽为100而高为40的矩形。当鼠标移动时，显示一个消息表示鼠标点是否在矩形内，如图16-23b所示。为了检查一个点是否在矩形内，需要使用定义在练习题10.12中的MyRectangle2D类。

**16.21 （几何问题：是否在三角形内？）编写一个程序，绘制三个端点分别是（20，20）、（100，100）和（140，40）的三角形。当鼠标移动时，显示一个消息表示鼠标点是否在三角形内，如图16-23c所示。为了检查一个点是否在三角形内，需要使用定义在练习题10.13中的Triangle2D类。

***16.22 （游戏：豆机动画）编写程序用动画实现练习题15.24介绍的豆机。在10个球掉下来之后动画结束，如图16-24所示。

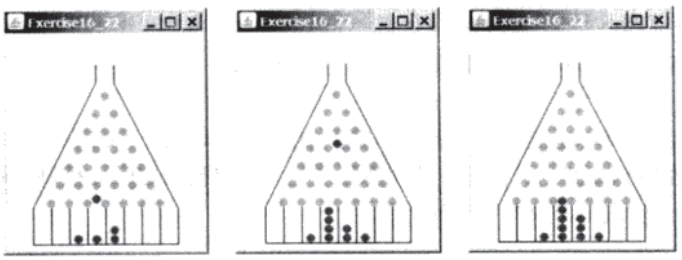
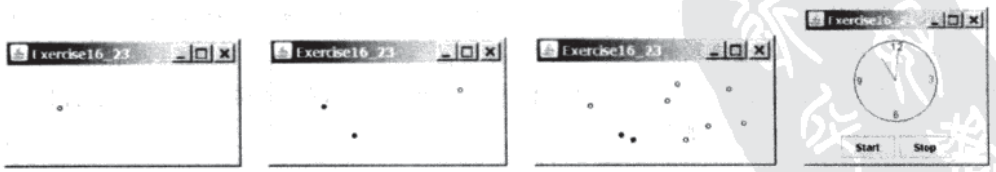



图16-24 球落入豆机

***16.23 （几何问题：最接近的点对）编写一个程序，让用户在面板上点击来自动产生点。初始状态时，面板是空的。当面板有两个或更多的点，突出显示最近的一对点。当创建一个新的点时，就突出显示新的最近的一对点。使用小圆显示点，使用实心圆突出显示点，如图16-25a～图16-25c所示。

**提示** 将点存储在ArrayList中。

*16.24 （控制时钟）修改程序清单16-12来添加start()和stop()方法以启动和停止时钟。编写一个程序，让用户控制带Start和Stop按钮的时钟，如图16-25d所示。




a) b) c) d)

图16-25 练习题16.23允许用户随着鼠标的点击而创建新的点，并凸显最近的点对。

练习题16.24允许用户启动和停止一个时钟

***16.25 （游戏：打气球）编写一个程序，在面板内的任意位置显示一个气球（如图16-26a所示）。使用向左和向右箭头键指定枪向左或向右瞄准气球（如图16-26b）。点击向上箭头键可以从枪中发出一个小球（如图16-26c所示）。一旦球击中该气球，就显示气球碎片（如图16-26e所示），然后

在任意位置显示一个新的气球（如图16-26f所示）。如果球没有击中气球，一旦它碰到面板的边界，该球就消失了。然后点击向上的箭头键就可以发出另一个球。当点击向左或向右箭头键，枪就会向左或向右转5度。（教师可以如下修改这个游戏：（1）显示被打破的气球的个数；（2）显示一个倒计时的计时器（例如，60秒），一旦时间到就终止这个游戏；（3）允许气球动态地上升。）

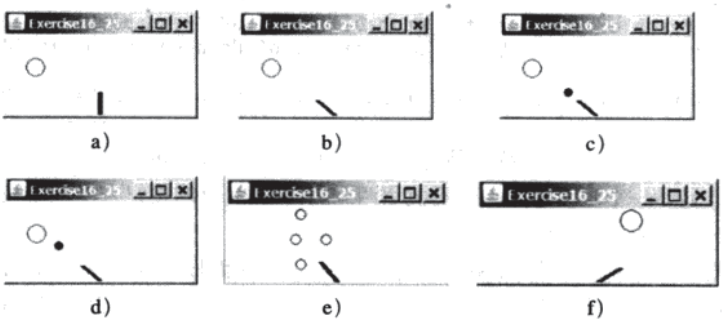



图16-26 a）在任意一个位置显示气球；b）点击向左/向右箭头键来瞄准气球；c）点击向上键发出一个球；d）球直接向气球移动；e）球击中气球；f）在任意位置显示一个新的气球

**16.26 （使用鼠标移动一个圆）编写一个程序，显示半径为10像素的圆。可以将鼠标在圆内点一下，然后随着鼠标的移动来拖动（即按住鼠标移动）圆，如图16-27a～图16-27b所示。

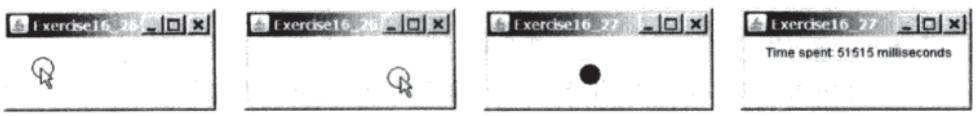



图16-27 a）～b）可以点击、拖动和移动圆；c）点击一个圆时，一个新的圆会出现在任意位置；d）点击20个圆后，面板中会显示所用的时间

***16.27 （游戏：手眼协调）编写一个程序，显示一个半径为10像素的实心圆，该圆放置在面板上的任意位置，并填充任意颜色，如图16-27c所示。点击这个圆时，它会消失，然后在另一个任意的位置显示新的任意颜色的圆。在点击了20个圆之后，在面板上显示所用的时间，如图16-27d所示。

***16.28 （模拟：自避免随机漫步）栅栏中的自避免漫步是从一个点到另一点的过程中，不重复两次访问一个点的路线。自避免漫步已经应用在物理、化学和数学学科中。它们可以用来模拟像溶剂和聚合物这样的链状物。编写一个程序，显示一个从中心点出发到边界点结束的随机路径，如图16-28a所示，或者在一个死端点结束（即该点被四个已经访问过的点包围），如图16-28b所示。假设栅栏的大小是16×16。

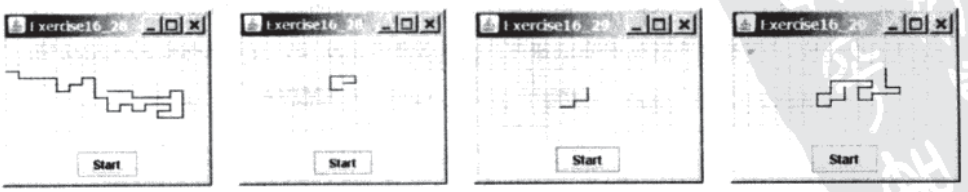



图16-28 a）一个在边界点结束的路径；b）一个在死端点结束的路径；c）～d）动画显示逐步构造路径的过程

***16.29 （动画：自避免随机漫步）修改上一个练习题，在动画中一步一步地显示路线，如图16-28c图16-28d所示。

**16.30 （模拟：自避免随机漫步）编写一个模拟程序，显示出现死端点路径的可能性随着格子数量的增加而变大。程序模拟大小从10到80的栅栏。对每种大小的栅栏，模拟自避免随机漫步10 000次，然后显示出现死端点路径的概率，如下面的示例输出所示：

```
For a lattice of size 10, the probability of dead-end paths is 10.6%
For a lattice of size 11, the probability of dead-end paths is 14.0%
...
For a lattice of size 80, the probability of dead-end paths is 99.5%
```

*16.31 （几何问题：显示正$n$边形）练习题15.25创建RegularPolygonPanel显示一个正$n$边形。编写一个程序，显示一个正多边形，然后使用两个名为+1和-1的按钮来增加或减少多边形的边数，如图16-29a～图16-29b所示。

**16.32 （几何问题：添加或删除点）编写一个程序，让用户在面板上点击以自动创建或移去点。当用户右击鼠标时，就创建一个点，并且显示在鼠标的位置，用户还可以将鼠标移到一个点上，然后左击鼠标以移去这个点，如图16-29c所示。

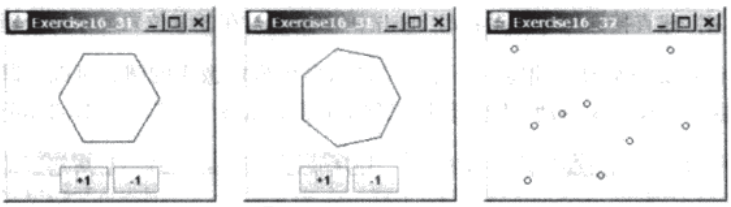



图16-29 点击+1或-1按钮增加或减少练习题16.31中正多边形的边数。

练习题16.32允许用户动态地创建/移去点

**16.33 （几何问题：回文）编写一个程序，用动画完成回文摆动，如图16-30所示。点击向上箭头键增加速度，点击向下箭头键降低速度。点击S键停止动画，点击R键重新开始。

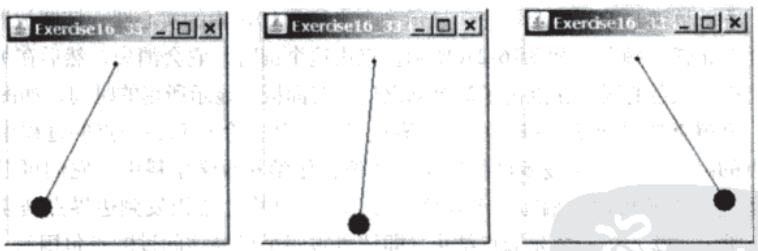



图16-30 练习题16.33用动画实现一个回文摆动

**16.34 （游戏：剑子手）编写一个程序，用动画实现剑子手游戏的摆动，如图16-31所示。点击向上箭头键增加速度，点击向下箭头键降低速度。点击S键停止动画，点击R键重新开始动画。

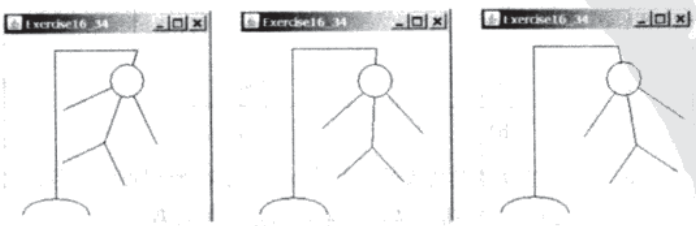
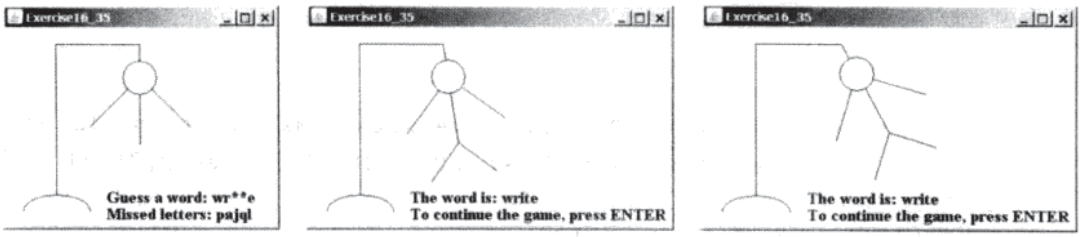



图16-31 练习题16.34用动画实现一个剑子手游戏

***16.35 （游戏：刽子手）练习题9.31给出了流行的刽子手游戏的控制台版本。编写一个GUI程序让用户来玩这个游戏。用户通过一次输入一个字母来猜单词，如图16-32a所示。如果用户七次都没猜对，被吊的人就摆动起来，如图16-32b～图16-32c所示。一旦完成一个单词，用户就可以点击Enter键继续猜另一个单词。




a) b) c)

图16-32 练习题16.35开发一个完整的刽子手游戏

*16.36 （翻硬币）编写一个程序，显示九个硬币的正面（H）或反面（T），如图16-33所示。当点击一个格子时，硬币就被翻面。一个格子就是一个JLabel。编写一个自定制的格子类，该类扩展JLabel且使用鼠标监听器处理这些点击。当程序开始时，所有的格子都被初始化为H。

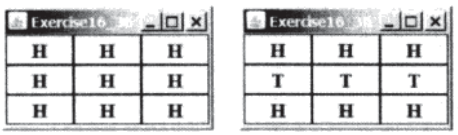



图16-33 练习题16.36可以让用户点击格子来翻硬币