Глава 12
СОБЫТИЯ

Основные понятия

Обработка любого события (нажатие кнопки, щелчок мышью и др.) состоит в связывании события с методом, его обрабатывающим. Принцип обработки событий, начиная с Java 2, базируется на модели делегирования событий. В этой модели имеется блок прослушивания события (**EventListener**), который ждет поступления события определенного типа от источника, после чего обрабатывает его и возвращает управление. Источник – это объект, который генерирует событие, если изменяется его внутреннее состояние, например, изменился размер, изменилось значение поля, произведен щелчок мыши по форме или выбор значения из списка. После генерации объект-событие пересылается для обработки зарегистрированному в источнике блоку прослушивания как параметр его методов – обработчиков событий.

Блоки прослушивания **Listener** представляют собой объекты классов, реализующих интерфейсы прослушивания событий, определенных в пакете **java.awt.event**. Соответствующие методы, объявленные в используемых интерфейсах, необходимо явно реализовать при создании собственных классов прослушивания. Эти методы и являются обработчиками события. Передаваемый источником блоку прослушивания объект-событие является аргументом обработчика события. Объект класса – блока прослушивания события необходимо зарегистрировать в источнике методом

```
источник.addСобытиеListener(объект_прослушиватель);
```

После этого объект-прослушиватель (**Listener**) будет реагировать именно на данное событие и вызывать метод «обработчик события». Такая логика обработки событий позволяет легко отделить интерфейсную часть приложения от функциональной, что считается необходимым при проектировании современных приложений. Удалить слушателя определенного события можно с помощью метода **remove**Событие**Listener()**.

Источником событий могут являться элементы управления: кнопки (**JButton**, **JCheckbox**, **JRadioButton**), списки, кнопки-меню. События могут генерироваться фреймами и апплетами, как mouse- и key-события. События генерируются окнами при развертке, сворачивании, выходе из окна. Каждый класс-источник определяет один или несколько методов **add**Событие**Listener()** или наследует эти методы

Когда событие происходит, все зарегистрированные блоки прослушивания уведомляются и принимают копию объекта события. Таким образом источник вызывает метод-обработчик события, определенный в классе, являющемся блоком прослушивания, и передает методу объект события в качестве параметра. В качестве блоков прослушивания на практике используются внутренние классы. В

этом случае в методе, регистрирующем блок прослушивания в качестве параметра, используется объект этого внутреннего класса.

Каждый интерфейс, включаемый в блок прослушивания, наследуется от интерфейса **EventListener** и предназначен для обработки определенного типа событий. При этом он содержит один или несколько методов, которые всегда принимают объект события в качестве единственного параметра и вызываются в определенных ситуациях. В таблице приведены некоторые интерфейсы и их методы, которые должны быть реализованы в классе прослушивания событий, реализующем соответствующий интерфейс:

Интерфейсы	Обработчики события
`ActionListener`	`actionPerformed(ActionEvent e)`
`AdjustmentListener`	`adjustmentValueChanged(` ` AdjustmentEvent e)`

`ComponentListener`	`componentResized(`
	` ComponentEvent e)`
	`componentMoved(ComponentEvent e)`
	`componentShown(ComponentEvent e)`
	`componentHidden(ComponentEvent e)`

`ContainerListener`	`componentAdded(ContainerEvent e)`
	`componentRemoved(`
	`ContainerEvent e)`

`FocusListener`	`focusGained(FocusEvent e)` `focusLost(FocusEvent e)`
`ItemListener`	`itemStateChanged(ItemEvent e)`

`KeyListener`	`keyPressed(KeyEvent e)`
	`keyReleased(KeyEvent e)`
	`keyTyped(KeyEvent e)`

`MouseListener`	`mouseClicked(MouseEvent e)`
	`mousePressed(MouseEvent e)`
	`mouseReleased(MouseEvent e)`
	`mouseEntered(MouseEvent e)`
	`mouseExited(MouseEvent e)`

`MouseMotionListener`	`mouseDragged(MouseEvent e)` `mouseMoved(MouseEvent e)`
`TextListener`	`textValueChanged(TextEvent e)`

`WindowListener`	`windowOpened(WindowEvent e)`
	`windowClosing(WindowEvent e)`
	`windowClosed(WindowEvent e)`
	`windowIconified(WindowEvent e)`
	`windowDeiconified(WindowEvent e)`
	`windowActivated(WindowEvent e)`

Событие, которое генерируется в случае возникновения определенной ситуации и затем передается зарегистрированному блоку прослушивания для обработки, – это объект класса событий. В корне иерархии классов событий находится суперкласс **EventObject** из пакета **java.util**. Этот класс содержит два метода: **getSource()**, возвращающий источник событий, и **toString()**, возвращающий строчный эквивалент события. Абстрактный класс **AWTEvent** из пакета **java.awt** является суперклассом всех AWT-событий, связанных с компонентами. Метод **getID()** определяет тип события, возникающего вследствие действий пользователя в визуальном приложении. Ниже приведены некоторые из классов событий, производных от **AWTEvent**, и расположенные в пакете **java.awt.event**:

ActionEvent – генерируется: при нажатии кнопки; двойном щелчке клавишей мыши по элементам списка; при выборе пункта меню;

AdjustmentEvent – генерируется при изменении полосы прокрутки;

ComponentEvent – генерируется, если компонент скрыт, перемещен, изменен в размере или становится видимым;

FocusEvent – генерируется, если компонент получает или теряет фокус ввода;

TextEvent – генерируется при изменении текстового поля;

ItemEvent – генерируется при выборе элемента из списка.

Класс **InputEvent** является абстрактным суперклассом событий ввода (для клавиатуры или мыши). События ввода с клавиатуры обрабатывает класс **KeyEvent**, события мыши – **MouseEvent**.

Чтобы реализовать методы-обработчики событий, связанных с клавиатурой, необходимо определить три метода, объявленные в интерфейсе **KeyListener**. При нажатии клавиши генерируется событие со значением **KEY_PRESSED**. Это приводит к запросу обработчика событий **keyPressed()**. Когда клавиша отпускается, генерируется событие со значением **KEY_RELEASED** и выполняется обработчик **keyReleased()**. Если нажатием клавиши сгенерирован символ, то посылается уведомление о событии со значением **KEY_TYPED** и вызывается обработчик **keyTyped()**.

Для регистрации события приложение-источник из своего объекта должно вызвать метод **addKeyListener(KeyListener el)**, регистрирующий блок прослушивания этого события. Здесь **el** – ссылка на блок прослушивания события.

```
/* пример # 1 : обработка событий клавиатуры: MyKey.java */
package chapt12;
import java.awt.*;
import java.awt.event.*;
import javax.swing.JApplet;

public class MyKey extends JApplet {
      private String msg = " ";
      private int x = 0, y = 20; // координаты вывода

      private class AppletKeyListener
                        implements KeyListener {
            // реализация всех трех методов интерфейса KeyListener
```

```
            public void keyPressed(KeyEvent e) {
                  showStatus("Key Down");
            } // отображение в строке состояния
            public void keyReleased(KeyEvent e) {
                  showStatus("Key Up");
            } // отображение в строке состояния
            public void keyTyped(KeyEvent e) {
                  msg += e.getKeyChar();
                  repaint(); // перерисовать
            }
      }
      public void init() {
            /* регистрация блока прослушивания */
            addKeyListener(new AppletKeyListener());
            requestFocus(); // запрос фокуса ввода
      }
      public void paint(Graphics g) {
            // значение клавиши в позиции вывода
            g.drawString(msg, x, y);
      }
}
```

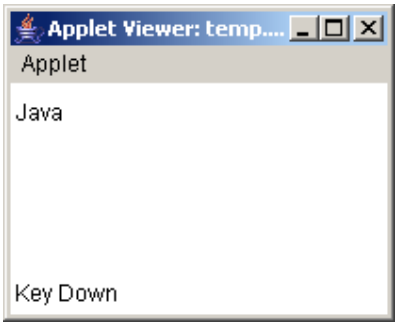

Рис. 12.1. Результат нажатия клавиши отображен в строке состояния

Коды специальных клавиш (перемещение курсора, функциональных клавиш) недоступны через **keyTyped()**, для обработки нажатия этих клавиш используется метод **keyPressed()**.

В качестве блока прослушивания в методе **init()** зарегистрирован внутренний класс **AppletKeyListener**. Затем в блоке прослушивания реализованы все три метода обработки события, объявленные в интерфейсе **KeyListener**.

В следующем апплете проверяется принадлежность прямоугольнику координат нажатия клавиши мыши с помощью реализации интерфейса **MouseListener** и события **MouseEvent**.

```
/* пример # 2 : события нажатия клавиши мыши: MyRect.java */
package chapt12;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
```

```
public class MyRect extends JApplet {
      private Rectangle rect =
                        new Rectangle(20, 20, 100, 60);
      private class AppletMouseListener//блок обработки событий
                        implements MouseListener {
            /* реализация всех пяти методов интерфейса MouseListener */
            public void mouseClicked(MouseEvent me) {
                  int x = me.getX();
                  int y = me.getY();
                  if (rect.contains(x, y)) {
                        showStatus(
                        "клик в синем прямоугольнике");
                  } else {
                        showStatus("клик в белом фоне");
                  }
            }
            // реализация остальных методов интрефейса пустая
            public void mouseEntered(MouseEvent e) {}
            public void mouseExited(MouseEvent e) {}
            public void mousePressed(MouseEvent e) {}
            public void mouseReleased(MouseEvent e) {}
      }
      public void init() {
            setBackground(Color.WHITE);
            /* регистрация блока прослушивания */
            addMouseListener(new AppletMouseListener());
      }
      public void paint(Graphics g) {
            g.setColor(Color.BLUE);
            g.fillRect(rect.x, rect.y,
                              rect.width, rect.height);
      }
}
```

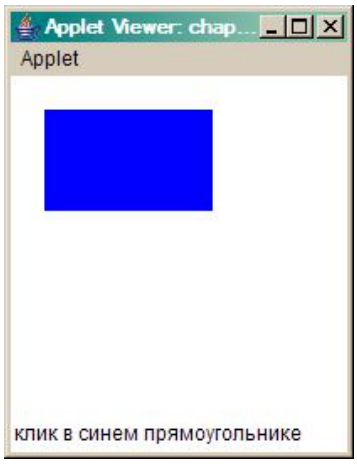

Рис. 12.2. Результат нажатия кнопки отображен в строке состояния

Способ обработки событий в компонентах Swing – это интерфейс (графические компоненты) и реализация (код обработчика события, который запускается при возникновении события). Каждое событие содержит сообщение, которое может быть обработано в разделе реализации.

При использовании компонента **JButton** определяется событие, связанное с нажатием кнопки. Для регистрации заинтересованности блока прослушивания в этом событии вызывается метод **addActionListener()** объектом класса **JButton**. Интерфейс **ActionListener** содержит единственный метод **actionPerformed()**, который нужно реализовать в блоке обработки в соответствии с поставленной задачей: извлечь числа из двух текстовых полей, сложить их и поместить результат в метку.

```
/* пример # 3 : регистрация, генерация и обработка ActionEvent:
SimpleButtonAction.java */
package chapt12;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class SimpleButtonAction extends JApplet {
      private JButton additionBtn = new JButton("Сложить");
      private JTextField txtField1 = new JTextField(3);
      private JTextField txtField2 = new JTextField(3);
      private JLabel answer = new JLabel();

      private class ButtonListener
                        implements ActionListener {
      // реализация класса- обработчика события
      public void actionPerformed(ActionEvent ev) {
            try {
             int t1, t2;
             t1 = Integer.parseInt(txtField1.getText());
             t2 = Integer.parseInt(txtField2.getText());
             answer.setText("Ответ: " + (t1 + t2));
                  showStatus("Выполнено успешно!");
            } catch (NumberFormatException e) {
                  showStatus("Ошибка ввода!");
            }
            /*
             * String s1, s2; извлечение надписи на кнопке из события
             * s1 = ((JButton)ev.getSource()).getText();
             */
            // извлечение команды из события
            // s2 = ev.getActionCommand();
            /*
             * извлечение из события объекта, ассоциированного с кнопкой
             * if (ev.getSource() == additionBtn)
             * применяется если обрабатываются
```

```
                * события нескольких кнопок одним обработчиком
                */
               }
        }
        public void init() {
               Container c = getContentPane();
               setLayout(new FlowLayout());/* «плавающее»
                                              размещение компонентов*/
               c.add(txtField1);
               c.add(txtField2);
               // регистрация блока прослушивания события
               additionBtn.addActionListener(
                                           new ButtonListener());
               c.add(additionBtn);
               c.add(answer);
        }
}
```

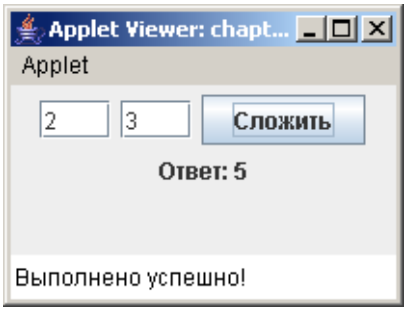

Рис. 12.3. Обработка события кнопки

При создании кнопки вызывается конструктор **JButton** со строкой, которую нужно поместить на кнопке. **JButton** – это компонент, который автоматически заботится о своей перерисовке. Размещение кнопки на форме обычно производится внутри метода **init()** вызовом метода **add()** класса **Container**.

Классы-адаптеры

Реализация всех методов–обработчиков событий, объявленных в интерфейсах, не всегда необходима. Чтобы не реализовывать все методы из соответствующих интерфейсов при создании классов – блоков прослушивания событий, используются классы-адаптеры. Такой класс содержит пустую реализацию всех методов из интерфейсов прослушивания событий, которые он расширяет. При этом определяется новый класс – блок прослушивания событий, который расширяет один из имеющихся адаптеров и реализует только те события, которые требуется обрабатывать.

Например, класс **MouseMotionAdapter** имеет два метода: **mouseDragged()** и **mouseMoved()**. Сигнатуры этих пустых методов точно такие же, как в интерфейсе **MouseMotionListener**. Если существует заинтересованность только в событиях перетаскивания мыши, то можно просто расширить

адаптер **MouseMotionAdapter** и переопределить метод **mouseDragged()** в своем классе. Событие же перемещения мыши обрабатывала бы реализация метода **mouseMoved()**, которую можно оставить пустой.

Рассмотрим события, возникающие в приложениях, связанных с консольным или графическим окнами. Когда происходит событие, связанное с окном, вызываются обработчики событий, определенные для этого окна. При создании оконного приложения используется метод **main()**, создающий для него окно верхнего уровня. После этого программа будет функционировать как приложение GUI, а не консольная программа. Программа поддерживается в работоспособном состоянии, пока не закрыто окно.

Для создания графического интерфейса потребуется предоставить место (окно), в котором он будет отображаться. Если программа является приложением, подобные действия она должна выполнять самостоятельно. В самом общем смысле окно является контейнером, т.е. областью, на которой рисуется пользовательский интерфейс. Графический контекст инкапсулирован в классе и доступен двумя способами:

- при переопределении методов **paint()** и **update()**;
- через возвращаемое значение метода **getGraphics()** класса **Component**.

Событие **FocusEvent** предупреждает программу, что компонент получил или потерял фокус ввода. Событие **WindowEvent** извещает программу, что был активизирован один из системных элементов управления окна.

В следующем примере реализован простейший графический редактор, позволяющий выбрать цвет и рисовать пером в поле приложения. Приложение создает объект **PaintEditor** и передает ему управление сразу же в методе **main()**. Выбор цвета осуществляется с помощью объекта класса **JColorChooser**.

```
/* пример # 4 : применение адаптеров: PaintEditor.java */
package chapt12;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class PaintEditor extends JFrame {
      private int prevX, prevY;
      private Color color = Color.BLACK;
      private JButton jButton =
                        new JButton("ColorChooser");

      public PaintEditor() {
            Container c = getContentPane();
            c.setLayout(new FlowLayout());
            c.add(jButton);

            jButton.addActionListener(
                        new ButtonActionListener());
            addMouseListener(new PaintMouseAdapter());
```

```
            addMouseMotionListener(
                        new PaintMouseMotionAdapter());
      }
      private class ButtonActionListener
                              implements ActionListener {
            public void actionPerformed(ActionEvent e) {
                  color = JColorChooser.
                  showDialog(((Component) e.getSource())
                              .getParent(), "Demo", color);
            }
      }
      private class PaintMouseAdapter extends MouseAdapter{
            public void mousePressed(MouseEvent ev) {
                  setPreviousCoordinates(
                                    ev.getX(), ev.getY());
            }
      }
      private class PaintMouseMotionAdapter
                              extends MouseMotionAdapter {
            public void mouseDragged(MouseEvent ev) {
                  Graphics g = getGraphics();
                  g.setColor(color);
                  g.drawLine(
                  prevX, prevY, ev.getX(), ev.getY());
                  setPreviousCoordinates(
                                    ev.getX(), ev.getY());
            }
      }
      public void setPreviousCoordinates(
                              int aPrevX, int aPrevY) {
            prevX = aPrevX;
            prevY = aPrevY;
      }
      public static void main(String[] args) {
            PaintEditor pe = new PaintEditor();
            pe.addWindowListener(new WindowAdapter() {
                  public void windowClosing(WindowEvent ev)
                  {
                        System.exit(0);
                  }
            });
            pe.setBounds(200, 100, 300, 200);
            pe.setTitle("MicroPaint");
            pe.setVisible(true);
      }
}
```

Рис. 12.4. Простейший текстовый редактор с классами-адаптерами

Конструктор класса **PaintEditor** использует методы **addMouseListener(new PaintMouseAdapter(this))** и **addMouseMotionListener(new PaintMouseMotionAdapter(this))** для регистрации событий мыши. При создании объекта класса **PaintEditor** эти методы сообщают объекту, что он заинтересован в обработке определенных событий. Однако вместо того, чтобы известить его об этом прямо, конструктор организует посылку ему предупреждений через объекты классов **PaintMouseAdapter** и **PaintMouseMotionAdapter**. Абстрактный класс **MouseAdapter** используется для обработки событий, связанных с мышью при создании блока прослушивания, и содержит следующие переопределяемые методы: **mousePressed(MouseEvent e)**, **mouseReleased(MouseEvent e)**. Абстрактный класс **MouseMotionAdapter** используется для обработки событий, связанных с движениями мыши при создании блока прослушивания, и содержит следующие переопределяемые методы: **mouseDragged(MouseEvent e)**, **mouseMoved(MouseEvent e)**.

Класс **PaintEditor** также обрабатывает событие класса **WindowEvent**. Когда объект генерирует событие **WindowEvent**, объект **PaintEditor** анализирует, является ли оно событием **WindowClosing**. Если это не так, объект **PaintEditor** игнорирует его. Если получено ожидаемое событие, в программе запускается процесс завершения ее работы.

Абстрактный класс **WindowAdapter** используется для приема и обработки событий окна при создании объекта прослушивания. Класс содержит методы: **windowActivated(WindowEvent e)**, вызываемый при активизации окна; **windowClosing(WindowEvent e)**, вызываемый при закрытии окна, и др.

Задания к главе 12

1. Создать фрейм с областью для рисования “пером”. Создать меню для выбора цвета и толщины линии.
2. Создать апплет с областью для рисования “пером”. Создать меню для выбора цвета и толщины линии.
3. Создать фрейм с областью для рисования. Добавить кнопки для выбора цвета (каждому цвету соответствует своя кнопка), кнопку для очистки окна. Рисование на панели со скроллингом.
4. Создать апплет с областью для рисования. Добавить кнопки для выбора цвета (каждому цвету соответствует своя кнопка), кнопку для очистки окна. Рисование на панели со скроллингом.

5. Создать простой текстовый редактор, содержащий меню и использующий классы диалогов для открытия и сохранения файлов.
6. Создать апплет, содержащий (**JLabel**) с текстом "Простой апплет", кнопку и текстовое поле (**JTextField**), в которое при каждом нажатии на кнопку выводится по одной строке из текстового файла.
7. Изменить апплет для предыдущей задачи таким образом, чтобы он мог работать и как апплет, и как приложение.
8. Составить программу для управления скоростью движения точки по апплету. Одна кнопка увеличивает скорость, другая – уменьшает. Каждый щелчок изменяет скорость на определенную величину.
9. Изобразить в окне гармонические колебания точки вдоль некоторого горизонтального отрезка. Если длина отрезка равна q, то расстояние от точки до левого конца в момент времени t можно считать равным q(1 + cos(wt))/2, где w – некоторая константа. Предусмотреть поля для ввода указанных величин и кнопку для остановки и пуска процесса.
10. Для предыдущей задачи предусмотреть возможность управления частотой колебаний с помощью двух кнопок. С помощью других двух кнопок (можно клавиш) управлять амплитудой, т.е. величиной q.
11. Построить в апплете ломаную линию по заданным вершинам. Координаты вершин вводятся через текстовое поле и фиксируются нажатием кнопки.
12. Нарисовать в апплете окружность с координатами центра и радиусом, вводимыми через текстовые поля.
13. Создать апплет со строкой, которая движется горизонтально, отражаясь от границ апплета и меняя при этом свой цвет на цвет, выбранный из выпадающего списка.
14. Создать апплет со строкой, движущейся по диагонали. При достижении границ апплета все символы строки случайным образом меняют регистр. При этом шрифт меняется на шрифт, выбранный из списка.

Тестовые задания к главе 12

Вопрос 12.1.

Выбрать необходимое условие принадлежности класса к апплетам.

1) класс – наследник класса **Applet** при отсутствии метода **main();**
2) класс – наследник класса **Applet** или его подкласса;
3) класс – наследник класса **Applet** с переопределенным методом **paint();**
4) класс – наследник класса **Applet** с переопределенным методом **init();**
5) класс – наследник класса **Applet,** и все его методы объявлены со спецификатором доступа **public.**

Вопрос 12.2.

Дан код:

```
import java.awt.*;
public class Quest2 extends Frame{
```

```
public static void main(String[] args){
 Quest2 fr = new Quest2();
 fr.setSize(222, 222);
 fr.setVisible(true);
 } }
```

Как сделать поверхность фрейма белой?

1) `fr.setBackground(Color.white);`
2) `fr.setColor(Color.white);`
3) `fr.Background(Color.white);`
4) `fr.color=Color.White;`
5) `fr.setColor(0,0,0).`

Вопрос 12.3.

Что произойдет при попытке компиляции и запуска следующего кода?

```
import java.awt.*;
import java.awt.event.*;
public class Quest3 extends Frame
      implements WindowListener {
      public Quest3(){
        setSize(300,300);
        setVisible(true);
        }
public void windowClosing(WindowEvent e){
               System.exit(0);
               }
public static void main(String args[]){
        Quest3 q = new Quest3();
        } }
```

1) ошибка компиляции;
2) компиляция и запуск с выводом пустого фрейма;
3) компиляция без запуска;
4) ошибка времени выполнения.

Вопрос 12.4.

Какие из приведенных классов являются классами-адаптерами?

1) `WindowAdapter;`
2) `WindowsAdapter;`
3) `AdjustmentAdapter;`
4) `ItemAdapter;`
5) `FocusAdapter.`

Вопрос 12.5.

Выберите из предложенных названий интерфейсы Event Listener.

1) `MouseMotionListener;`
2) `WindowListener;`
3) `KeyTypedListener;`
4) `ItemsListener.`