**События и слушатели в Java**

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.**add**Событие**Listener(**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 **addКеуListener(KeyListener** el**)**, регистрирующий блок прослушивания этого события. Здесь **el** – ссылка на блок прослушивания события.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | /\* 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  // реализация всех трех методов интерфейса KeyListener       private class AppletKeyListener  implements 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);       }   } |

[](http://crypto.pp.ua/wp-content/uploads/2010/06/clip_image00211.jpg)

**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**.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75 | /\* 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);        }    } |

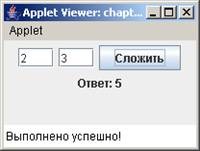
[](http://crypto.pp.ua/wp-content/uploads/2010/06/clip_image0043.jpg)

**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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101 | /\* 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);        }    } |

*[](http://crypto.pp.ua/wp-content/uploads/2010/06/clip_image0062.jpg)*

**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**.