**События и слушатели**

Событие **Event** - это объект, описывающий изменение состояния источника, с которым оно связано. Примером события, в котором участвует пользователь, являются нажатие кнопки, выбор элемента из списка, ввод символа с клавиатуры и т.д. Событие может происходить и без участия пользователя при использовании таймера.

**Слушатель Listener** - это уведомляемый о некотором событии объект. Чтобы слушатель смог реагировать на определенное событие источника он должен быть им зарегистрирован, т.е. подключен к источнику. *Listener* должен реализовывать определенные методы для получения и обработки уведомлений о событии.

**Listener** находится в постоянном ожидании, пока в источнике, в котором он зарегистрирован, не наступит соответствующее событие, при возникновении которого слушатель получает управление. Также слушателю передается объект события (источник), чтобы он смог правильно на него отреагировать. Таким образом, источник вызывает метод-обработчик события, определенный в классе, являющемся блоком прослушивания. В качестве блоков прослушивания иногда используют внутренние классы. В этом случае в методе, регистрирующем блок прослушивания в качестве параметра, используется объект этого внутреннего класса.

После обработки события *слушатель* возвращает управление. Таким образом, для обработки события вызываются только те слушатели, которые на него "подписались", т.е. были зарегистрированы источником.

**Типы событий и слушателей**

В пакете java.awt.event определены интерфейсы слушателей для каждого из определенных в нем типов событий (например, для событий MouseEvent определено два интерфейса слушателей: MouseListener и MouseMotionListener). Все интерфейсы слушателей событий являются расширениями интерфейса java.util.EventListener. В этом интерфейсе не определяется ни один из методов, но он играет роль базового интерфейса, в котором однозначно определены все слушатели событий как таковые.Т.е. слушатель наследуется от интерфейса EventListener и предназначен для обработки определенного типа событий. При этом **Listener** содержит один или несколько методов, которые принимают объект события в качестве единственного параметра и вызываются в определенных ситуациях.

Интерфейс слушателя событий **Listener** может включать несколько методов. Например, класс событий, подобный MouseEvent, описывает несколько событий, связанных с мышью, таких как события нажатия и отпускания кнопки мыши. Эти события вызывают различные методы соответствующего слушателя.

В таблице приведены определенные в пакете java.awt.event типы событий, соответствующие им слушатели, а также методы, определенные в каждом интерфейсе слушателя.

|  |  |  |
| --- | --- | --- |
| **Класс события** | **Интерфейс слушателя** | **Обработчики события** |
| ActionEvent | ActionListener | actionPerformed(ActionEvent e) |
| AdjustmentEvent | AdjustmentListener | adjustmentValueChanged(AdjustmentEvent e) |
| ComponentEvent | ComponentListener | componentResized(ComponentEvent e) |
| componentMoved(ComponentEvent e) |
| componentShown(ComponentEvent e) |
| componentHidden(ComponentEvent e) |
| ContainerEvent | ContainerListener | componentAdded(ContainerEvent e) |
| componentRemoved(ContainerEvent e) |
| FocusEvent | FocusListener | focusGained(FocusEvent e) |
| focusLost(FocusEvent e) |
| ItemEvent | ItemListener | itemStateChanged(ItemEvent e) |
| KeyEvent | KeyListener | keyPressed(KeyEvent e) |
| keyReleased(KeyEvent e) |
| keyTyped(KeyEvent e) |
| MouseEvent | MouseListener | mouseClicked(MouseEvent e) |
| mousePressed(MouseEvent e) |
| mouseReleased(MouseEvent e) |
| mouseEntered(MouseEvent e) |
| mouseExited(MouseEvent e) |
| MouseMotionListener | mouseDragged(MouseEvent e) |
| mouseMoved(MouseEvent e) |
| TextEvent | TextListener | textValueChanged(TextEvent e) |
| WindowEvent | WindowListener | windowOpened(WindowEvent e) |
| windowClosing(WindowEvent e) |
| windowClosed(WindowEvent e) |
| windowIconified(WindowEvent e) |
| windowDeiconified(WindowEvent e) |
| windowActivated(WindowEvent e) |

Корнем иерархии классов событий является суперкласс **EventObject** из пакета *java.util*. Данный класс содержит два метода: *getSource()*, возвращающий источник событий, и *toString()*, возвращающий строчный эквивалент события. Чтобы узнать, в каком объекте произошло событие, нужно вызвать метод getSource(), возвращающий значение типа object. Следовательно, один и тот же слушатель можно подключить к разным источникам.

**Классы-адаптеры, Adapter**

Для каждого интерфейса слушателей событий, содержащего несколько методов, в пакете *java.awt.event* определен **класс-адаптер Adapter**. Когда нужен только один или два таких метода, иногда проще получить подкласс класса-адаптера, чем реализовать интерфейс самостоятельно. *При использовании адаптера требуется лишь переопределить те методы, которые нужны, а при прямой реализации интерфейса необходимо определить все методы, в том числе и ненужные в данной программе*.

Заранее определенные классы-адаптеры называются также, как и интерфейсы, которые они реализуют. Но в этих названиях **Listener** заменяется на **Adapter**; например *MouseAdapter*, *MouseMotionAdapter*, *WindowAdapter* и т.д.

**Описание класса-адаптера действий с мышью, MouseAdapter**

public abstract class MouseAdapter implements MouseListener

{

public void mouseClicked(MouseEvent e){}

public void mousePressed(MouseEvent e){}

public void mouseReleased(MouseEvent e){}

public void mouseEntered(MouseEvent e){}

public void mouseExited(MouseEvent e){}

}

public abstract class MouseMotionAdapter

implements MouseMotionListener

{

public void mouseDragged(MouseEvent e){}

public void mouseMoved(MouseEvent e){}

}

Классов-адаптеров всего семь. Кроме уже упомянутых трех классов, это классы ComponentAdapter, ContainerAdapter, FocusAdapter и KeyAdapter.

**События, связанные с визуальными компонентами AWT**

В следующей таблице приведен список визуальных компонентов пакета AWT и событий, которые они порождают.

|  |  |  |
| --- | --- | --- |
| **Компонент** | **Событие** | **Описание** |
| Button | ActionEvent | Пользователь нажал кнопку |
| CheckBox | ItemEvent | Пользователь установил или сбросил флажок |
| CheckBoxMenuItem | ItemEvent | Пользователь установил или сбросил флажок рядом с пунктом меню |
| Choice | ItemEvent | Пользователь выбрал элемент списка или отменил его выбор |
| Component | ComponentEvent | Элемент либо перемещен, либо он стал скрытым, либо видимым |
| FocusEvent | Элемент получил или потерял фокус ввода |
| KeyEvent | Пользователь нажал или отпустил клавишу |
| MouseEvent | Пользователь нажал или отпустил кнопку мыши, либо курсор мыши вошел или покинул область, занимаемую элементом, либо пользователь просто переместил мышь или переместил мышь при нажатой кнопке мыши |
| Container | ContainerEvent | Элемент добавлен в контейнер или удален из него |
| List | ActionEvent | Пользователь выполнил двойной щелчок мыши на элементе списка |
| ItemEvent | Пользователь выбрал элемент списка или отменил выбор |
| MenuItem | ActionEvent | Пользователь выбрал пункт меню |
| Scrollbar | AdjustmentEvent | Пользователь осуществил прокрутку |
| TextComponent | TextEvent | Пользователь внес изменения в текст элемента |
| TextField | ActionEvent | Пользователь закончил редактирование текста элемента |
| Window | WindowEvent | Окно было открыто, закрыто, представлено в виде пиктограммы, восстановлено или требует восстановления |

**Регистрация слушателя Listener**

Для регистрации слушателя источник использует специальные методы. Как правило, имена методов имеют форму addXxxListener(XxxListener listener) или setXxxListener(XxxListener listener), где Xxx - это имя события, а listener - ссылка на слушателя событий.

**Пример использования слушателя ActionListener**

package test;

import java.awt.Dimension;

import java.awt.FlowLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.JTextField;

public class TestFrame extends JFrame

{

private static final long serialVersionUID = 1L;

private JTextField textField;

private JButton button1;

private JButton button2;

private JButton button3;

public TestFrame() {

super("Test frame");

createGUI();

}

public static void main(String[] args) {

javax.swing.SwingUtilities.invokeLater(new Runnable() {

public void run() {

JFrame.setDefaultLookAndFeelDecorated(true);

TestFrame frame = new TestFrame();

frame.pack();

frame.setLocationRelativeTo(null);

frame.setVisible(true);

}

});

}

//В этом методе мы создаем функциональное окошко*(представленное на рисунке ниже)* и регистрируем для кнопок *(нажатие на которые будут создавать события)* слушателя событий, выполняющего при нажатии на кнопки прописанные в его классе “ TestActionListener ” действия

public void createGUI() {

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

JPanel panel = new JPanel();

panel.setLayout(new FlowLayout());

button1 = new JButton("Button 1");

button1.setActionCommand("Button 1 was pressed!");

panel.add(button1);

button2 = new JButton("Button 2");

button2.setActionCommand("Button 2 was pressed!");

panel.add(button2);

button3 = new JButton("Button 3");

button3.setActionCommand("Button 3 was pressed!");

panel.add(button3);

textField = new JTextField();

textField.setColumns(23);

panel.add(textField);

ActionListener actionListener = new TestActionListener();

button1.addActionListener(actionListener);

button2.addActionListener(actionListener);

button3.addActionListener(actionListener);

getContentPane().add(panel);

setPreferredSize(new Dimension(320, 100));

}

//В этом Классе создаем слушателя событий, который будет, при создании события (нажатии на кнопку), выводить в тектовое поле под кнопками текст, установленный для каждой кнопки командой - например на рисунке ниже это будет команда “button3.setActionCommand("Button 3 was pressed!");”

public class TestActionListener implements ActionListener {

public void actionPerformed(ActionEvent e) {

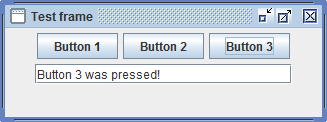
textField.setText(e.getActionCommand());

}

}

}

Интерфейс приложения представлен на рисунке.



**Программный вызов события**

Событие вызывается автоматически, при наступлении определенных условий. Но можно событие создать и вызвать программно (fire event).

В предыдущий пример были внесены изменения во внутренний класс TestActionListener, в результате чего по нажатию на кнопку button3 создается и вызывается новое событие.

public class TestActionListener implements ActionListener {

public void actionPerformed(ActionEvent e) {

JButton button = (JButton) e.getSource();

System.out.println (button.getText() + ", " +

e.getActionCommand());

if (e.getSource() != button3) {

textField.setText(e.getActionCommand());

} else {

ActionEvent e1 = new ActionEvent(button2,

Event.MOUSE\_DOWN,

"Button 2 was pressed programmatically!");

ActionListener[] listeners;

listeners = button2.getActionListeners();

listeners[0].actionPerformed(e1);

}

}

}

После нажатия на кнопку button3 в консоли будет выведена следующая информация :

Button 3, Button 3 was pressed!

Button 2, Button 2 was pressed programmatically!