Графический интерфейс в Java

# Библиотеки для GUI

# AWT - пакет java.awt

AWT (Abstract Window Toolkit) первая технология java для создания графического (оконного) интерфейса пользователя – GUI. Пакет java.awt претерпел, пожалуй, больше всего изменений с развитием версий Java.

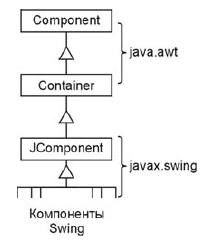
# Swing пакет javax.swing

<http://java-online.ru/libs-swing.xhtml>

http://www.quizful.net/post/swing-layout-managers

Вслед за **AWT** Sun разработала графическую библиотеку компонентов **Swing**, полностью написанную на Java. Набор стандартных компонентов значительно превосходит AWT по разнообразию и функциональности. Swing позволяет легко создавать новые компоненты, наследуясь от существующих, и поддерживает различные стили и скины.

Создатели новой библиотеки пользовательского интерфейса **Swing** не стали «изобретать велосипед» и в качестве основы для своей библиотеки выбрали AWT. Конечно, речь не шла об использовании конкретных тяжеловесных компонентов AWT (представленных классами Button, Label и им подобными). Нужную степень гибкости и управляемости обеспечивали только легковесные компоненты. На диаграмме наследования представлена связь между AWT и Swing.



Важнейшим отличием **Swing** от AWT является то, что компоненты Swing вообще не связаны с операционной системой и поэтому гораздо более стабильны и быстры. Такие компоненты в Java называются легковесными и понимание основных принципов их работы во многом объяснит работу Swing.

# JavaFX

JavaFX - это программная платформа для создания и доставки настольных приложений и интернет-приложений (RIA), которые могут работать на самых разных устройствах. JavaFX предназначен для замены Swing в качестве стандартной библиотеки GUI для Java SE.

# Создание графического интерфейса swing

# Классы для реализации GUI в swing

Они представляют собой окна операционной системы, в которых размещаются компоненты пользовательского интерфейса.

К контейнерам высшего уровня относятся окна JFrame и JWindow, диалоговое окно JDialog, а также апплет JApplet (который не является окном, но тоже предназначен для вывода интерфейса в браузере, запускающем этот апплет).

Контейнеры высшего уровня Swing представляют собой тяжеловесные компоненты и являются исключением из общего правила.

Все остальные компоненты Swing являются легковесными.

Все элементы в форме размещаются в панели содержания, которая доступна методу getContentPane(). Чтобы элемент разместить на форме, его надо добавить на панель формы, например, так frame.getContentPane().add(label);

Пример создания оконного интерфейса **JFrame**.

import java.awt.Dimension;

import javax.swing.JFrame;

import javax.swing.JLabel;

public class JFrameTest

{

public static void createGUI()

{

JFrame frame = new JFrame("Test frame");

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

JLabel label = new JLabel("Test label");

frame.getContentPane().add(label);

frame.setPreferredSize(new Dimension(200, 100));

frame.pack();

frame.setVisible(true);

}

public static void main(String[] args)

{

JFrame.setDefaultLookAndFeelDecorated(true);

//выполнение приложения с формой

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

public void run() {

createGUI();

}

});

}

}

# Основной контейнер JFrame

* Окно **класс JFrame**

JFrame – класс для создания окна приложения.

Представляет окно с рамкой и строкой заголовка (с кнопками «Свернуть», «Во весь экран» и «Закрыть»). Оно может изменять размеры и перемещаться по экрану.

* Конструкторы

Конструктор без параметров создает пустое окно.

JFrame()

Конструктор создает пустое окно с заголовком

JFrame(String title)

* Методы для создания простого окна

|  |  |
| --- | --- |
|  |  |
| void setSize(int width, int height) | устанавливает размеры окна. Если не задать размеры, окно будет иметь нулевую высоту независимо от того, что в нем находится и пользователю после запуска придется растягивать окно вручную. Размеры окна включают не только «рабочую» область, но и границы и строку заголовка |
| setDefaultCloseOperation(int operation) | позволяет указать действие, которое необходимо выполнить, когда пользователь закрывает окно нажатием на крестик. Обычно в программе есть одно или несколько окон, при закрытии которых программа прекращает работу. Для того, чтобы запрограммировать это поведение, следует в качестве параметра operation передать константу EXIT\_ON\_CLOSE, описанную в классе JFrame. |
| void setVisible(boolean visible) | когда окно создается, оно по умолчанию невидимо. Чтобы отобразить окно на экране, вызывается данный метод с параметром true. Если вызвать его с параметром false, окно снова станет невидимым. |

# Элементы управления

# Панели

JPanel – класс для определения объекта, на котором можно размещать элементы управления (JRootPanel – основной объект окна – слой).

### Менеджеры компоновки Layout в Swing

Определяют способ размещения элементов управления в окне JFrame, определяют размер и расположение компонентов, а так же при изменении размера окна пропорционально масштабируют компоненты формы.

Менеджеры компоновки Layout в Swing применяются для компонентов(JFrame,JPanel,JButton и др.).

Метод setLayout() – устанавливает менеджер.

Стандартные менеджеры компоновки AWT и Swing.

**BorderLayout**

Используется в Swing по умолчанию. Он предоставляет области для для установки компонентов:BorderLayout.NORTH(верх), BorderLayout.SOUTH (низ), BorderLayout.EAST (справа), BorderLayout.WEST (слева), BorderLayout.CENTER (заполнить середину до других компонент или до краев). По умолчанию принимается константа Center.¶

Пример использование BorderLayout

package layoutdemo;

import java.awt.BorderLayout; ¶

import javax.swing.JButton; ¶

import javax.swing.JFrame; ¶

import javax.swing.JPanel; ¶

¶

public class BorderLayoutTest { ¶

public static void main(String[] args) { ¶

// создаем фрейм и устанавливаем его размер. ¶

JFrame jf = new JFrame(); ¶

jf.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); ¶

jf.setSize(400, 300); ¶

jf.setVisible(true); ¶

// создаем панель. ¶

JPanel p = new JPanel(); ¶

jf.add(p); ¶

// к панели добавляем менеджер BorderLayout. ¶

p.setLayout(new BorderLayout()); ¶

// к панели добавляем кнопку и устанавливаем для нее менеджер в

//верхнее расположение. ¶

p.add(new JButton("Okay"), BorderLayout.NORTH); ¶

} ¶

}

**FlowLayout**

FlowLayout менеджер устанавливает компоненты слева направо последовательно и при заполнении переходит на следующую строку.

Пример использования FlowLayout.

package layoutdemo; ¶

¶

import java.awt.FlowLayout;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JPanel;

public class FlowLayoutDemo { ¶

public static void main(String[] args) { ¶

// создаем окно и устанавливаем его размер. ¶

JFrame jf = new JFrame(); ¶

jf.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); ¶

jf.setSize(400, 300); ¶

jf.setVisible(true); ¶

// создаем панель. ¶

JPanel p = new JPanel(); ¶

jf.add(p); ¶

// к панели добавляем менеджер FlowLayout. ¶

p.setLayout(new FlowLayout()); ¶

// к панели добавляем кнопки. ¶

p.add(new JButton("start 2")); ¶

p.add(new JButton("start 2")); ¶

p.add(new JButton("start 3")); ¶

p.add(new JButton("start 4")); ¶

p.add(new JButton("start 5")); ¶

p.add(new JButton("start 6")); ¶

p.add(new JButton("Okay")); ¶

} ¶

}

**GridLayout**

GridLayout это менеджер, который помещает компоненты в ячейки таблицы.

Пример. Использование табличного менеджера компоновки.

package layoutdemo;

¶import java.awt.GridLayout;

¶import javax.swing.JButton;

¶import javax.swing.JFrame;

¶import javax.swing.JPanel;

¶

¶public class GridLayoutDemo {

¶ public static void main(String[] args) {

¶ JFrame jf = new JFrame();

¶ jf.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

¶ jf.setSize(400, 300);

¶ jf.setVisible(true);

¶ // создаем панель.

¶ JPanel p = new JPanel();

¶ jf.add(p);

¶ // к панели добавляем менеджер GridLayout и устанавливаем размеры таблицы 3\*3.

¶ p.setLayout(new GridLayout(3,3));

¶

¶ // к панели добавляем кнопку и устанавливаем для нее менеджер в верхнее расположение.

¶ p.add(new JButton("start 2"));

¶ p.add(new JButton("start 2"));

¶ p.add(new JButton("start 3"));

¶ p.add(new JButton("start 4"));

¶ p.add(new JButton("start 5"));

¶ p.add(new JButton("start 6"));

¶ p.add(new JButton("Okay"));

¶ }

¶}

**GridBagLayout**

Этот менеджер подобно GridLayout менеджеру устанавливает компоненты в таблицу, но он более гибок, так как предоставляет возможность определять для компонентов разную ширину и высоту колонок и строк таблицы. По существу, GridBagLayout помещает компоненты в ячейки, и затем использует привилегированные размеры компонентов, чтобы определить, насколько большой ячейка должна быть.

import java.awt.\*;

¶import javax.swing.JButton;

¶import javax.swing.JFrame;

¶

¶public class GridBagLayoutDemo {

¶ final static boolean shouldFill = true;

¶ final static boolean shouldWeightX = true;

¶ final static boolean RIGHT\_TO\_LEFT = false;

¶ public static void addComponentsToPane(Container pane)

{ if (RIGHT\_TO\_LEFT)

{

pane.setComponentOrientation(ComponentOrientation.RIGHT\_TO\_LEFT);

¶ }

¶ JButton button;

¶ pane.setLayout(new GridBagLayout());

¶ GridBagConstraints c = new GridBagConstraints();

¶ if (shouldFill) {

¶ // натуральная высота, максимальная ширина

¶ c.fill = GridBagConstraints.HORIZONTAL;

¶ }

¶ button = new JButton("Button 1");

¶ if (shouldWeightX) {

¶ c.weightx = 0.5;

¶ }

¶

¶ c.fill = GridBagConstraints.HORIZONTAL;

¶ c.gridx = 0;

¶ c.gridy = 0;

¶ pane.add(button, c);

¶ button = new JButton("Button 2");

¶ c.fill = GridBagConstraints.HORIZONTAL;

¶ c.weightx = 0.5;

¶ c.gridx = 1;

¶ c.gridy = 0;

¶ pane.add(button, c);

¶ button = new JButton("Button 3");

¶ c.fill = GridBagConstraints.HORIZONTAL;

¶ c.weightx = 0.5;

¶ c.gridx = 2;

¶ c.gridy = 0;

¶ pane.add(button, c);

¶ button = new JButton("Long-Named Button 4");

¶ c.fill = GridBagConstraints.HORIZONTAL;

¶ c.ipady = 40; // сделать эту кнопку высокой

¶ c.weightx = 0.0;

¶ c.gridwidth = 3;

¶ c.gridx = 0;

¶ c.gridy = 1;

¶ pane.add(button, c);

¶

¶ button = new JButton("5");

¶ c.fill = GridBagConstraints.HORIZONTAL;

¶ c.ipady = 0; // установить первоночальный размер кнопки

¶ c.weighty = 1.0; // установить отступ

¶ c.anchor = GridBagConstraints.PAGE\_END; // установить кнопку в конец окна

¶ c.insets = new Insets(10, 0, 0, 0); // поставить заглушку

¶ c.gridx = 1; // выравнять компонент по Button 2

¶ c.gridwidth = 2; // установить в 2 колонку

¶ c.gridy = 2; // и 3 столбец

¶ pane.add(button, c);

¶

¶ }

¶

¶ private static void createAndShowGUI() {

¶ // Создание окна

¶ JFrame frame = new JFrame("GridBagLayoutTest");

¶ frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

¶ // Установить панель содержания

¶ addComponentsToPane(frame.getContentPane());

¶ // Показать окно

¶ frame.pack();

¶ frame.setVisible(true);

¶ }

¶

¶ public static void main(String[] args) {

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

¶ public void run() {

¶ createAndShowGUI();

¶ }

¶ });

¶ }

¶}

**BoxLayout**

BoxLayout позволяет управлять размещением компонентов, отдельно в вертикальном либо горизонтальном направлении помещая их, друг за другом, и управлять пространством между компонентами, используя вставки.

package layoutdemo;

¶

¶import java.awt.Component;

¶import java.awt.Container;

¶import javax.swing.BoxLayout;

¶import javax.swing.JButton;

¶import javax.swing.JFrame;

¶

¶public class BoxLayoutDemo {

¶ public static void addComponentsToPane(Container pane) {

¶ pane.setLayout(new BoxLayout(pane, BoxLayout.Y\_AXIS));

¶ addAButton("Button 1", pane);

¶ addAButton("Button 2", pane);

¶ addAButton("Button 3", pane);

¶ addAButton("Long-Named Button 4", pane);

¶ addAButton("5", pane);

¶ }

¶

¶ private static void addAButton(String text, Container container) {

¶ JButton button = new JButton(text);

¶ button.setAlignmentX(Component.CENTER\_ALIGNMENT);

¶ container.add(button);

¶ }

¶

¶ private static void createAndShowGUI() {

¶ // Создание фрейма

¶ JFrame frame = new JFrame("BoxLayoutDemo");

¶ frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

¶ addComponentsToPane(frame.getContentPane());

¶

¶ frame.pack();

¶ frame.setVisible(true);

¶ }

¶

¶ public static void main(String[ ] args) {

¶ // запустить приложение

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

¶ public void run() {

¶ createAndShowGUI();

¶ }

¶ });

¶ }

¶}

**CardLayout**

Этот менеджер предназначен для использования нескольких менеджеров.

package layout;

¶import java.awt.\*;

¶import java.awt.event.\*;

¶import javax.swing.\*;

¶

¶public class CardLayoutDemo implements ItemListener {

¶ JPanel cards;

¶ final static String BUTTONPANEL = "Card with JButtons";

¶ final static String TEXTPANEL = "Card with JTextField";

¶ public void addComponentToPane(Container pane) {

¶ // поместить JComboBox в JPanel для наглядности.

¶ JPanel comboBoxPane = new JPanel();

¶ String comboBoxItems[] = { BUTTONPANEL, TEXTPANEL };

¶ JComboBox cb = new JComboBox(comboBoxItems);

¶ cb.setEditable(false);

¶ cb.addItemListener(this);

¶ comboBoxPane.add(cb);

¶ // Создание "cards".

¶ JPanel card1 = new JPanel();

¶ card1.add(new JButton("Button 1"));

¶ card1.add(new JButton("Button 2"));

¶ card1.add(new JButton("Button 3"));

¶ JPanel card2 = new JPanel();

¶ card2.add(new JTextField("TextField", 20));

¶ // Создаем панель

¶ cards = new JPanel(new CardLayout());

¶ cards.add(card1, BUTTONPANEL);

¶ cards.add(card2, TEXTPANEL);

¶ pane.add(comboBoxPane, BorderLayout.PAGE\_START);

¶ pane.add(cards, BorderLayout.CENTER);

¶ }

¶ public void itemStateChanged(ItemEvent evt) {

¶ CardLayout cl = (CardLayout)(cards.getLayout());

¶ cl.show(cards, (String)evt.getItem());

¶ }

¶ private static void createAndShowGUI() {

¶ // Создание и настройка окна

¶ JFrame frame = new JFrame("CardLayoutDemo");

¶ frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

¶ CardLayoutDemo demo = new CardLayoutDemo();

¶ demo.addComponentToPane(frame.getContentPane());

¶ // Показ окна

¶ frame.pack();

¶ frame.setVisible(true);

¶ }

¶ public static void main(String[] args) {

¶ try {

UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");

¶ } catch (UnsupportedLookAndFeelException ex) {

¶ ex.printStackTrace();

¶ } catch (IllegalAccessException ex) {

¶ ex.printStackTrace();

¶ } catch (InstantiationException ex) {

¶ ex.printStackTrace();

¶ } catch (ClassNotFoundException ex) {

¶ ex.printStackTrace();

¶ }

¶

¶ UIManager.put("swing.boldMetal", Boolean.FALSE);

¶

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

¶ public void run() {

¶ createAndShowGUI();

¶ }

¶ });

¶ }

¶}

**GroupLayout**

GroupLayout менеджер имеет возможность независимо устанавливать горизонтальное и вертикальное расположение компонентов на форме.

Он использует два типа добавления компонентов параллельный и последовательный объединенный с иерархическим составом.

1. Последовательным добавляет компоненты просто помещая один за другим, точно так же как BoxLayout или FlowLayout вдоль одной оси. Положение каждого компонента определяется относительно предыдущего компонента.

2. Помещает компонентов параллельно относительно друг друга в то же самом месте. Они добавляются к верху формы или выравниваются к основанию вдоль вертикальной оси. Вдоль горизонтальной оси они устанавливаются влево или по центру, если у компонентов разный размер.

Пример GroupLayout

package layout;

import javax.swing.\*;

import static javax.swing.GroupLayout.Alignment.\*;

public class Find extends JFrame {

public Find() {

JLabel label = new JLabel("Find What:");

JTextField textField = new JTextField();

JCheckBox caseCheckBox = new JCheckBox("Match Case");

JCheckBox wrapCheckBox = new JCheckBox("Wrap Around");

JCheckBox wholeCheckBox = new JCheckBox("Whole Words");

JCheckBox backCheckBox = new JCheckBox("Search Backwards");

JButton findButton = new JButton("Find");

JButton cancelButton = new JButton("Cancel");

caseCheckBox.setBorder(BorderFactory.createEmptyBorder(0, 0, 0, 0));

wrapCheckBox.setBorder(BorderFactory.createEmptyBorder(0, 0, 0, 0));

wholeCheckBox.setBorder(BorderFactory.createEmptyBorder(0, 0, 0, 0));

backCheckBox.setBorder(BorderFactory.createEmptyBorder(0, 0, 0, 0));

GroupLayout layout = new GroupLayout(getContentPane());

getContentPane().setLayout(layout);

layout.setAutoCreateGaps(true);

layout.setAutoCreateContainerGaps(true);

¶

layout.setHorizontalGroup(layout.createSequentialGroup().addComponent(label)

.addGroup(layout.createParallelGroup(LEADING

.addComponent(textField

.addGroup(layout.createSequentialGroup

.addGroup(layout.createParallelGroup(LEADING

.addComponent(caseCheckBox

.addComponent(wholeCheckBox

.addGroup(layout.createParallelGroup(LEADING

.addComponent(wrapCheckBox

.addComponent(backCheckBox

.addGroup(layout.createParallelGroup(LEADING

.addComponent(findButton

.addComponent(cancelButton)));

layout.linkSize(SwingConstants.HORIZONTAL, findButton, cancelButton);

layout.setVerticalGroup(layout.createSequentialGroup().addGroup(layout.createParallelGroup(BASELINE

.addComponent(label

.addComponent(textField

.addComponent(findButton

.addGroup(layout.createParallelGroup(LEADING

.addGroup(layout.createSequentialGroup

.addGroup(layout.createParallelGroup(BASELINE

.addComponent(caseCheckBox

.addComponent(wrapCheckBox

.addGroup(layout.createParallelGroup(BASELINE)

.addComponent(wholeCheckBox

.addComponent(backCheckBox)))

.addComponent(cancelButton

setTitle("Find");

pack();

setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

}

¶ public static void main(String args[]) {

¶ java.awt.EventQueue.invokeLater(new Runnable() {

¶ public void run() {

¶ try { UIManager.setLookAndFeel(¶"javax.swing.plaf.metal.MetalLookAndFeel

} catch (Exception ex)

ex.printStackTrace =

new Find().setVisible(true);

}

});

}

¶}

**NullLayout – размещение по усмотрению разработчика**

Если по каким либо причинам вам необходимо самостоятельно расположить компоненты, то можно воспользоваться менеджером NullLayout установив в метод setLayout() значение null.

Пример NullLayout.

package layout;

¶

import java.awt.Container;

import java.awt.Insets;

import java.awt.Dimension;

import javax.swing.JButton;

import javax.swing.JFrame;

public class AbsoluteLayoutDemo {

public static void addComponentsToPane(Container pane) {

pane.setLayout(null);

JButton b1 = new JButton("one");

JButton b2 = new JButton("two");

JButton b3 = new JButton("three");

pane.add(b1);

pane.add(b2);

pane.add(b3);

Insets insets = pane.getInsets();

Dimension size = b1.getPreferredSize();

b1.setBounds(25 + insets.left, 5 + insets.top,

size.width, size.height);

size = b2.getPreferredSize();

b2.setBounds(55 + insets.left, 40 + insets.top,

size.width, size.height);

size = b3.getPreferredSize();

b3.setBounds(150 + insets.left, 15 + insets.top,

size.width + 50, size.height + 20);

}

private static void createAndShowGUI() {

JFrame.setDefaultLookAndFeelDecorated(true);

JFrame frame = new JFrame("AbsoluteLayoutDemo");

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

addComponentsToPane(frame.getContentPane());

Insets insets = frame.getInsets();

frame.setSize(300 + insets.left + insets.right,

125 + insets.top + insets.bottom);

frame.setVisible(true);

}

public static void main(String[] args) {

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

public void run() {

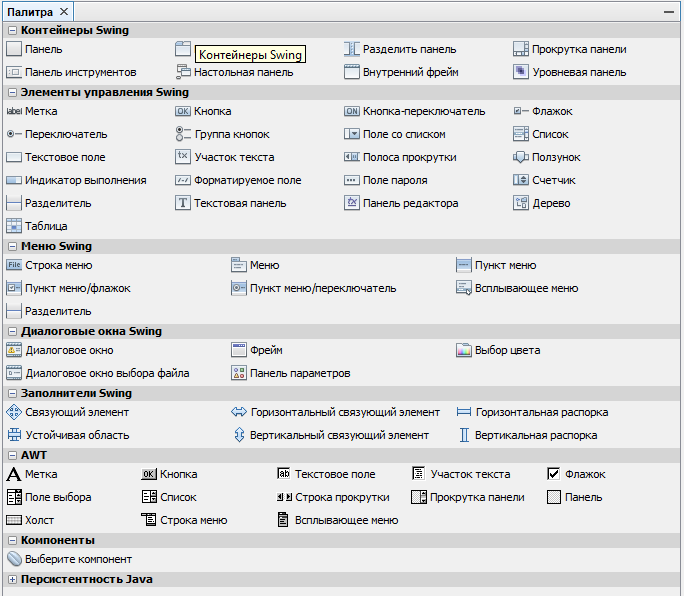
createAndShowGUI();

}

});

} ¶}

# Другие элементы управления Swing

Подробнее об элементах управления смотри документ [Библиотека swing](Библиотека%20swing.docx) в папке

# Обработка событий

Любое действие пользователя с GUI, приведет к какому – то событию. При разработке приложения необходимо предусмотреть реакцию на полезные события.

* Слушатели

В Java для реализации перехвата события используются *слушатели (Listener)–* это объекты, которые могут получать уведомление, когда случается особо интересное событие.

# Способы реализации обработки событий

### Анонимный внутренний класс

Через анонимный класс реализуются простые слушатели, когда нужен только один конкретный компонент.

Достоинство: данная технология легко применима, хорошо работает с простыми компонентами.

Недостатки: если в классе много слушателей реализуются через анонимные классы, то класс становится сильно загруженным кодом этих классов; не позволяет задействовать тот же слушатель несколько раз с разными компонентами.

Пример. При нажатии на кнопку, изменить значение метки.

|  |
| --- |
|  |

### Контейнер(форма) становится слушателем события.

Это удобно при наличии нескольких компонентов, которые необходимо прослушивать, и когда уже есть пользовательский класс для контейнера. В этом случае используется паттерн вида: Class Имя extends JConteinerName implements ListenerName, как в примере

|  |
| --- |
|  |

### Отдельный внешний класс слушатель

Преимущество такого подхода в легкости применения одного и того же класса слушатель и даже экземпляра класса, для различных компонентов. Недостаток – в том, что придется настраивать класс при применении.

|  |
| --- |
|  |

# Механизм обработки событий библиотеки Swing

Графический интерфейс пользователя включает в себя не только расположение в окне необходимых элементов управления, но и назначение реакции на пользовательские события. Большая честь действий в оконных программах выполняется в ответ на выбор пользователем команд меню, нажатие кнопок, а иногда даже просто в ответ на ввод нового символа в текстовое поле.

Таким образом, при разработке программы необходимо:

1. Выявить события, в ответ на которые потребуется реакция программы.
2. Написать код, реализующий эту реакцию (так называемый обработчик событий).
3. Связать обработчик события с соответствующим событием.

Первый пункт зависит от логики работы приложения и находится всецело на усмотрении программиста. Обработчик события — это, как правило, обычный метод и ничего особенного в его написании нет.

# Паттерн проектирования «наблюдатель»

В разных языках программирования и визуальных библиотеках используются самые различные способы привязки процедуры (метода), которая должна вызываться в ответ на некоторое событие, к самому этому событию. В частности, в Delphi процедура просто должна иметь особое название, состоящее из имени компонента и типа обрабатываемого события.

В библиотеке Swing для привязки обработчика события к вызывающему его компоненту используется паттерн проектирования «наблюдатель».

Паттерны проектирования — это стандартные приемы объектно-ориентированного программирования, позволяющие оптимальным образом справиться с нетривиальными, но часто возникающими в программировании ситуации. Паттерн проектирования описывает классы, которые необходимо ввести для разрешения такой ситуации и взаимодействие между классами. В частности, ранее мы рассматривали паттерн проектирования «итератор», позволяющий обойти все элементы коллекции с помощью двух простейших методов в порядке, скрытом в самом итераторе.

Паттерн проектирования «наблюдатель» применяется, когда один объект должен оповещать другие о произошедших с ним изменениях или внешних воздействиях. Такой объект называется наблюдаемым, а объекты, которые следует оповестить — наблюдателями.

Для того, чтобы подобное взаимодействие было возможным, объект-наблюдатель должен иметь метод (или несколько методов) с заранее определенной сигнатурой (именем и параметрами). Когда с наблюдаемым объектом произойдет ожидаемое событие, он вызовет соответствующий метод у своего наблюдателя. В этом методе и будет запрограммирована реакция на событие.

Для того, чтобы наблюдаемый объект мог вызвать метод наблюдателя, он должен знать о том, что такой наблюдатель существует. Поэтому у наблюдаемого предварительно должен быть вызван специальный метод, регистрирующий его наблюдателя.

Заметим, что в данной схеме один наблюдатель может быть зарегистрирован у нескольких объектов (т.е. одинаково реагировать на изменения в каждом из них), а у одного объекта может быть несколько наблюдателей (т.е. при возникновении события выполняется несколько независимых методов-обработчиков). Это весьма увеличивает гибкость программирования.

# Механизм обработки событий библиотеки Swing

В контексте графического интерфейса пользователя наблюдаемыми объектами являются элементы управления: кнопки, флажки, меню и т.д. Они могут сообщить своим наблюдателям об определенных событиях, как элементарных (наведение мышкой, нажатие клавиши на клавиатуре), так и о высокоуровневых (изменение текста в текстовом поле, выбор нового элемента в выпадающем списке и т.д.).

Наблюдателями должны являться объекты классов, поддерживающих специальные интерфейсы (вспомним, что в классе наблюдателя должны быть определенные методы, о которых «знает» наблюдаемый и вызывает их при наступлении события). Такие классы в терминологии Swing называются *слушателями*.

# Интерфейс MouseListener и обработка событий от мыши

Мы рассмотрим технологию написания слушателей на примере слушателей событий мыши.

События от мыши — один из самых популярных типов событий. Практически любой элемент управления способен сообщить о том, что на него навели мышь, щелкнули по нему и т.д. Об этом будут оповещены все зарегистрированные слушатели событий от мыши.

Так например, кнопка для входа в систему из примера в конце предыдущей главы должна реагировать на щелчок по ней, проверяя имя и пароль, введенные пользователем.

Слушатель событий от мыши должен реализовать интерфейс MouseListener. В этом интерфейсе перечислены следующие методы:

public void mouseClicked(MouseEvent event) — выполнен щелчок мышкой на наблюдаемом объекте

public void mouseEntered(MouseEvent event) — курсор мыши вошел в область наблюдаемого объекта

public void mouseExited(MouseEvent event) — курсор мыши вышел из области наблюдаемого объекта

public void mousePressed(MouseEvent event) — кнопка мыши нажата в момент, когда курсор находится над наблюдаемым объектом

public void mouseReleased(MouseEvent event) — кнопка мыши отпущена в момент, когда курсор находится над наблюдаемым объектом

О параметре типа MouseEvent

Чтобы обработать нажатие на кнопку, требуется описать класс, реализующий интерфейс MouseListener, причем метод mouseClicked() должен содержать обработчик события. Далее необходимо создать объект этого класса и зарегистрировать его как слушателя интересующей нас кнопки. Для регистрации слушателя используется метод addMouseListener(MouseListener listener).

Опишем класс слушателя в пределах класса окна SimpleWindow, после конструктора. Обработчик события будет проверять, ввел ли пользователь логин «Иван» (пароль проверять не будем) и выводить сообщение об успехе или неуспехе входа в систему (работа стандартных диалоговых окон обсуждается дальше в этой главе):

class MouseL implements MouseListener

{ public void mouseClicked(MouseEvent event)

{ if (loginField.getText().equals("Иван"))

JOptionPane.showMessageDialog(null, "Вход выполнен");

else

JOptionPane.showMessageDialog(null, "Вход НЕ выполнен"); }

public void mouseEntered(MouseEvent event) {}

public void mouseExited(MouseEvent event) {}

public void mousePressed(MouseEvent event) {}

public void mouseReleased(MouseEvent event) {}

}

Мы сделали слушателя вложенным классом класса SimpleWindow, чтобы он мог легко получить доступ к его внутренним полям loginField и passwordField. Кроме того, хотя реально мы обрабатываем только одно из пяти возможных событий мыши, описывать пришлось все пять методов (четыре имеют пустую реализацию). Дело в том, что в противном случае класс пришлось бы объявить абстрактным (ведь он унаследовал от интерфейса пустые заголовки методов) и мы не смогли бы создать объект этого класса. А мы должны создать объект слушателя и прикрепить его к кнопке. Для этого в код конструктора SimpleWindow() необходимо добавить команду:

ok.addMouseListener(new MouseL());

Это можно сделать сразу после команды:

JButton ok = new JButton("OK");

# Создание слушателей с помощью анонимных классов

Чтобы кнопка ok обрела слушателя, который будет обрабатывать нажатие на нее, нам понадобилось описать новый (вложенный) класс. Иногда вместо вложенного класса можно обойтись анонимным. Как вы помните, анонимный класс не имеет имени и в программе может быть создан только один объект этого класса (создание которого совмещено с определением класса). Но очень часто, как и в нашем примере, слушатель пишется для того, чтобы обрабатывать события единственного объекта — в нашем случае кнопки ok, а значит, используется в программе только однажды: во время привязки к этому объекту. Таким образом, мы можем заменить вложенный класс анонимным. Для этого описание класса MouseL можно просто удалить, а команду

ok.addMouseListener(new MouseL());

заменить на:

ok.addMouseListener(new MouseListener() {   public void mouseClicked(MouseEvent event) { if (loginField.getText().equals("Иван")) JOptionPane.showMessageDialog(null, "Вход выполнен"); else JOptionPane.showMessageDialog(null, "Вход НЕ выполнен"); }   public void mouseEntered(MouseEvent event) {}   public void mouseExited(MouseEvent event) {}   public void mousePressed(MouseEvent event) {}   public void mouseReleased(MouseEvent event) {}   });

Новый вариант выглядит более громоздко, чем первый. Злоупотребление анонимными классами может сделать программу плохо читаемой. Однако в результате все действия с кнопкой (создание, настройка ее внешнего вида и команды обработки щелчка по ней) не разнесены, как в случае вложенных классов, а находятся рядом, что облегчает сопровождение (внесение изменений) программы. В случае простых (в несколько строк) обработчиков разумно делать выбор в пользу анонимных классов.

Вообще, можно реализовать интерфейс слушателя в любом классе, добавив к его описанию директиву implements MouseListener и определив в нем необходимые методы. В частности, это может быть сам класс окна SimpleWindow. Но чаще всего пользуются анонимными и вложенными классами (не в последнюю очередь благодаря наличию адаптеров (см. далее).

# Класс MouseAdapter

Программа стала выглядеть загроможденной главным образом из-за того, что помимо полезного для нас метода mouseClicked() нам пришлось определять пустые реализации всех остальных, не нужных методов. В принципе, этого можно избежать.

Класс MouseAdapter реализует интерфейс MouseListener, определяя пустые реализации для каждого из его методов. Можно унаследовать своего слушателя от этого класса и переопределить те методы, которые нам нужны.

В результате предыдущее описание слушателя будет выглядеть более компактно:

ok.addMouseListener(new MouseAdapter() {

public void mouseClicked(MouseEvent event) {

if (loginField.getText().equals("Иван")) JOptionPane.showMessageDialog(null, "Вход выполнен"); else JOptionPane.showMessageDialog(null, "Вход НЕ выполнен"); }   });

# Общая структура слушателей

Кроме слушателей MouseListener визуальные компоненты Swing поддерживают целый ряд других слушателей (заметим, что в разных компонентах набор возможных слушателей может отличаться).

Каждый слушатель должен реализовывать интерфейс \*\*\*Listener, где \*\*\* — тип слушателя. Практически каждому из этих интерфейсов (за исключением тех, в которых всего один метод) соответствует пустой класс-заглушка \*\*\*Adapter. Каждый метод интерфейса слушателя принимает один параметр типа \*\*\*Event, в котором собрана вся информация, относящаяся к событию. Мы не будем подробно рассматривать методы получения этой информации. Как правило их немного и их назначение легко понять, а перечень методов даст контекстная подсказка Eclipse.

Чтобы привязать слушателя к объекту (который поддерживает соответствующий тип слушателей) используется метод add\*\*\*Listener(\*\*\*Listener listener).

Например, слушатель MouseListener должен реализовать интерфейс с таким же именем, которому соответствует класс-заглушка MouseAdapter. Методы этого интерфейса обрабатывают параметр типа MouseEvent, а регистрируется слушатель методом addMouseListener(MouseListener listener). Мы не будем повторять эту информацию для других типов слушателей, они работают аналогично.

# Слушатель фокуса FocusListener

Слушатель FocusListener отслеживает моменты, когда объект получает фокус (то есть становится активным) или теряет его. Концепция фокуса очень важна для оконных приложений. В каждый момент времени в окне может быть только один активный (находящийся в фокусе) объект, который получает информацию о нажатых на клавиатуре клавишах (т.е. реагирует на события клавиатуры), о прокрутке колесика мышки и т.д. Пользователь активирует один из элементов управления нажатием мышки или с помощью клавиши Tab (переключаясь между ними).

Интерфейс FocusListener имеет два метода:

public void focusGained(FocusEvent event) — вызывается, когда наблюдаемый объект получает фокус

public void focusLost(FocusEvent event) — вызывается, когда наблюдаемый объект теряет фокус.

# Слушатель колесика мышки MouseWheelListener

Слушатель MouseWheelListener оповещается при вращении колесика мыши в тот момент, когда данный компонент находится в фокусе. Этот интерфейс содержит всего один метод:

public void mouseWheelMoved(MouseWheelEvent event).

# Слушатель клавиатуры KeyListener

Слушатель KeyListener оповещается, когда пользователь работает с клавиатурой в тот момент, когда данный компонент находится в фокусе. В интерфейсе определены методы:

public void mouseKeyTyped(KeyEvent event) — вызывается, когда с клавиатуры вводится символ

public void mouseKeyPressed(KeyEvent event) — вызывается, когда нажата клавиша клавиатуры

public void mouseKeyReleased(KeyEvent event) — вызывается, когда отпущена клавиша клавиатуры.

Аргумент event этих методов способен дать весьма ценные сведения. В частности, команда event.getKeyChar() возвращает символ типа char, связанный с нажатой клавишей. Если с нажатой клавишей не связан никакой символ, возвращается константа CHAR\_UNDEFINED. Команда event.getKeyCode() возвратит код нажатой клавиши в виде целого числа типа int. Его можно сравнить с одной из многочисленных констант, определенных в классе KeyEvent: VK\_F1, VK\_SHIFT, VK\_D, VK\_MINUS и т.д. Методы isAltDown(), isControlDown(), isShiftDown() позволяют узнать, не была ли одновременно нажата одна из клавиш-модификаторов Alt, Ctrl или Shift.

# Слушатель изменения состояния ChangeListener

Слушатель ChangeListener реагирует на изменение состояния объекта. Каждый элемент управления по своему определяет понятие «изменение состояния». Например, для панели со вкладками JTabbedPane это переход на другую вкладку, для ползунка JSlider — изменение его положения, кнопка JButton рассматривает как смену состояния щелчок на ней. Таким образом, хотя событие это достаточно общее, необходимо уточнять его специфику для каждого конкретного компонента. В интерфейсе определен всего один метод:

public void stateChanged(ChangeEvent event).

# Слушатель событий окна WindowListener

Слушатель WindowListener может быть привязан только к окну и оповещается о различных событиях, произошедших с окном:

public void windowOpened(WindowEvent event) — окно открылось.

public void windowClosing(WindowEvent event) — попытка закрытия окна (например, пользователя нажал на крестик). Слово «попытка» означает, что данный метод вызовется до того, как окно будет закрыто и может воспрепятствовать этому (например, вывести диалог типа «Вы уверены?» и отменить закрытие окна, если пользователь выберет «Нет»).

public void windowClosed(WindowEvent event) — окно закрылось.

public void windowIconified(WindowEvent event) — окно свернуто.

public void windowDeiconified(WindowEvent event) — окно развернуто.

public void windowActivated(WindowEvent event) — окно стало активным.

public void windowDeactivated(WindowEvent event) — окно стало неактивным.

# Слушатель событий компонента СomponentListener

Слушатель ComponentListener оповещается, когда наблюдаемый визуальный компонент изменяет свое положение, размеры или видимость. В интерфейсе четыре метода:

public void componentMoved(ComponentEvent event) — вызывается, когда наблюдаемый компонент перемещается (в результате вызова команды setLocation(), работы менеджера размещения или еще по какой-то причине).

public void componentResized(ComponentEvent event) — вызывается, когда изменяются размеры наблюдаемого компонента.

public void componentHidden(ComponentEvent event) — вызывается, когда компонент становится невидимым.

public void componentShown(ComponentEvent event) — вызывается, когда компонент становится видимым.

# Слушатель выбора элемента ItemListener

Слушатель ItemListener реагирует на изменение состояния одного из элементов, входящих в состав наблюдаемого компонента. Например, выпадающий список JComboBox состоит из множества элементов и слушатель реагирует, когда изменяется выбранный элемент. Также данный слушатель оповещается при выборе либо отмене выбора флажка JCheckBox или переключателя JRadioButton, изменении состояния кнопки JToggleButton и т.д. Слушатель обладает одним методом:

public void itemStateChanged(ItemEvent event).

# Универсальный слушатель ActionListener

Среди многочисленных событий, на которые реагирует каждый элемент управления (и о которых он оповещает соответствующих слушателей, если они к нему присоединены), есть одно основное, вытекающее из самой сути компонента и обрабатываемое значительно чаще, чем другие. Например, для кнопки это щелчок на ней, а для выпадающего списка — выбор нового элемента.

Для отслеживания и обработки такого события может быть использован особый слушатель ActionListener, имеющий один метод:

public void actionPerformed(ActionEvent event).

У использования ActionListener есть небольшое преимущество в эффективности (так, при обработке нажатия на кнопку не надо реагировать на четыре лишних события — ведь даже если методы-обработчики пустые, на вызов этих методов все равно тратятся ресурсы). А кроме того очень удобно запомнить и постоянно использовать один класс с одним методам и обращаться к остальным лишь в тех относительно редких случаях, когда возникнет такая необходимость.

Обработка нажатия на кнопку ok в нашем примере легко переписывается для ActionListener:

ok.addMouseListener(new ActionListener()

{ public void actionPerformed(ActionEvent event)

{

if(loginField.getText().equals("Иван"))

JOptionPane.showMessageDialog(null, "Вход выполнен");

Else

JOptionPane.showMessageDialog(null, "Вход НЕ выполнен");

}

});

# Интерфейс KeyListener и абстрактный класс KeyAdapter

События клавиатуры также можно отслеживать. Для получения такого рода событий используется интерфейс KeyListener. Объекты-слушатели регистрируются для таких компонентов, как окно или панель, при помощи метода addKeyListener компонента.

Класс, заинтересованный в обработке событий клавиатуры, или реализует данный интерфейс вместе со всеми его методами, или же расширяет абстрактный класс KeyAdapter. В последнем случае вы можете переопределить только те методы, которые имеют значение для приложения.

В качестве примера в следующем приложении создается объект, который устанавливает, вводятся ли с клавиатуры какие-либо символы. Введенный текст отображается в метке внизу экрана:

import java.awt.\*;  
import java.awt.event.\*;  
import javax.swing.\*;  
  
public class KeyAdapterExample extends JFrame {  
  // Метка, отображающая введенный текст.  
  JLabel label;  
  // Окно, в котором перемещается мышь и вводится текст  
  JPanel p;  
  
  public KeyAdapterExample() {  
    label = new JLabel("You typed: ");  
  
    p = new JPanel();  
    p.setBackground(Color.white);  
    // Зарегистрировать введенный символ  
    p.addKeyListener(new KeyHandler());  
    // Зарегистрировать перемещение мыши в области глухой панели  
    p.addMouseListener(new MyMouseAdapter(p));  
  
    // Получить ContentPane и отформатировать  
    getContentPane().setLayout(new BorderLayout());  
    getContentPane().add(p, BorderLayout.CENTER);  
    getContentPane().add(label, BorderLayout.SOUTH);  
    // зарегистрировать выход из программы  
    addWindowListener(new WinClosing());  
    setBounds(100, 100, 200, 200);  
    setVisible(true);  
  }  
  
  // закрыть конструктор  
  
  // Класс, обрабатывающий набор символов  
  class KeyHandler extends KeyAdapter {  
    // Метод, извлекающий набранные символы и  
    // устанавливающий их в качестве значений метки  
    public void keyTyped(KeyEvent ke) {  
      label.setText("You typed: " + ke.getKeyChar());  
      label.invalidate();  
      invalidate();  
      validate();  
    }  
  }  
  
  // Класс, обрабатывающий перемещение мыши  
  class MyMouseAdapter extends MouseAdapter {  
    MyMouseAdapter(Component c) {  
      this.c = c;  
    }  
  
    public void mousePressed(MouseEvent e) {  
      c.requestFocus();  
    }  
  
    private Component c;  
  }  
  
  public static void main(String args[]) {  
    KeyAdapterExample kae = new KeyAdapterExample();  
  }  
}  
  
class WinClosing extends WindowAdapter  
  
{  
  public void windowClosing(WindowEvent we) {  
    System.exit(0);  
  }  
}

