Лекция 13. Обработка событий AWT. Построение графических приложений SWING

Оглавление

Предыдущая лекция была посвящена заданию внешнего вида пользовательского интерфейса. До сих пор он был статическим. Перейдем теперь к рассмотрению правил обработки различных событий, которые могут возникать как результат действий пользователя, и не только.

Модель обработки событий построена на основе стандартного шаблона проектирования *ООП* Observer/Observable. В качестве наблюдаемого объекта выступает тот или иной *компонент AWT*. Для него можно задать один или несколько классов-наблюдателей. В *AWT* они называются слушателями (listener) и описываются специальными интерфейсами, название которых оканчивается на *слово* Listener. Когда с наблюдаемым объектом что-то происходит, создается *объект* "событие" (*event*), который "посылается" всем слушателям. Так слушатель узнает, например, о действии пользователя и может на него отреагировать.

Каждое событие является подклассом класса java.util.EventObject. События пакета *AWT*, которые и рассматриваются в данной лекции, являются подклассами java.awt.AWTEvent. Для удобства классы различных событий и интерфейсы слушателей помещены в отдельный пакет java.awt.event.

Схема обработки событий в библиотеке AWT

Как уже отмечалось, в Java используется схема обработки событий, реализованная в библиотеке AWT. То есть независимо от того, легкие (библиотека Swing) или тяжелые (библиотека AWT) компоненты используются, схема обработки событий одна и та же.

На вершине иерархии классов, описывающих события, находится класс `EventObject`, который описан в пакете `java.util` и является в Java расширением общего суперкласса `Object`. В свою очередь, класс `EventObject` наследуется аб-

страктным классом AWTEvent. Этот класс описан в библиотеке AWT и, соответственно, находится в пакете java.awt. Все остальные классы обработки событий в библиотеке AWT являются подклассами класса AWTEvent и описаны в пакете java.awt.event. Иерархия классов обработки событий библиотеки AWT показана на рис. 1.

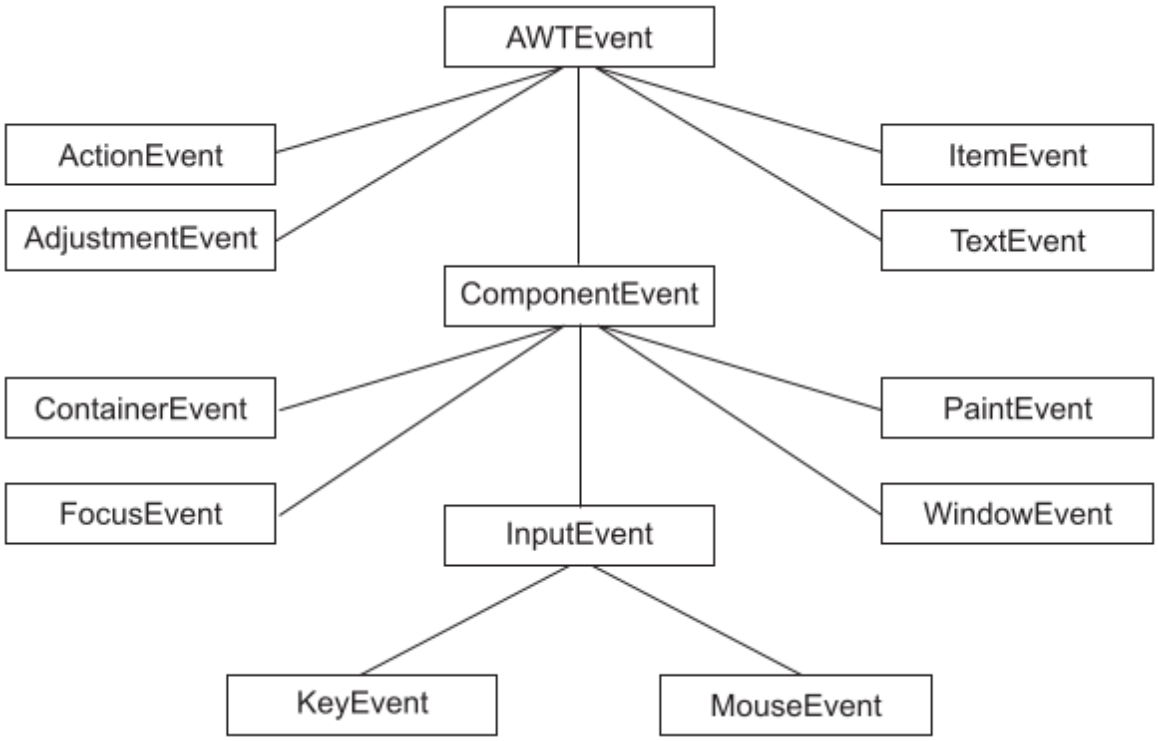

Рис. 1. Иерархия классов для описания событий

События (классы событий библиотеки AWT) кратко описаны в табл.1.

Таблица 1. Классы событий библиотеки AWT

Клас события	Описание	Может возникать
ActionEvent	Генерируется при щелчке мышью на кнопке	Возникает в компонентах классов Button, List и TextField
AdjustmentEvent	Возникает при изменении положения ползунка полосы прокрутки	Возникает в компонентах класса Scrollbar
ComponentEvent	Возникает при перемещении компонента, изменении его размеров, отображении и скрытии компонента	Возникает во всех компонентах

ItemEvent	Возникает при выборе или отказе от выбора элемента в соответствующих компонентах	Возникает в компонентах классов Checkbox, Choice и List
TextEvent	Происходит при изменении текста	Возникает в компонентах классов TextComponent, TextArea и TextField
ContainerEvent	Возникает, если в контейнер добавляется компонент или компонент из контейнера удаляется	Возникает в компонентах классов Container, Dialog, FileDialog, Frame, Panel, ScrollPane и Window
FocusEvent	Возникает, если соответствующий компонент получает или теряет фокус	Возникает во всех компонентах
InputEvent	Абстрактный класс, являющийся суперклассом для классов KeyEvent и MouseEvent. В классе определяются восемь целочисленных констант для получения информации о событии	Возникает при операциях ввода для компонентов
PaintEvent	Происходит при перерисовке компонента	Возникает в основных компонентах
WindowEvent	Происходит при открытии, закрытии, сворачивании, разворачивании окна, получении и передаче окном фокуса	Возникает в компонентах классов Dialog, FileDialog, Frame и Window
KeyEvent	Возникает при нажатии клавиши, отпускании клавиши, ввода символа	Возникает во всех компонентах
MouseEvent	Возникает при манипуляциях мышью с компонентом, таких как щелчок, перемещение указателя, появление указателя мыши на компоненте и т. д.	Возникает во всех компонентах

Общая схема обработки событий, используемая в Java, называется схемой с *делегированием*. В отличие от другой популярной схемы, где при появлении события опрашиваются все доступные обработчики событий, в схеме с делегированием для всех объектов, в которых могут происходить события

(точнее, в тех объектах, в которых предполагается обрабатывать события), явно указывается обработчик события. Удобство такого подхода состоит в значительной экономии времени на поиск нужного обработчика. Недостатки также очевидны — приходится писать больше кода.

Перечисленные в табл. 1 классы описывают события. Для обработки этих событий используют другие классы. Точнее, для каждого из событий существует специальный интерфейс создания класса обработки события. Для обработки события необходимо, как минимум, расширить интерфейс обработчика этого события. Названия интерфейсов для обработки событий связаны с названиями классов соответствующих событий. Чтобы получить название интерфейса, в имени соответствующего класса необходимо вместо слова `Event` вставить слово`Listener`.Например, для события класса WindowEvent интерфейс обработчика имеет название WindowListener. Исключением из этого правила является класс InputEvent — для этого события собственного интерфейса нет.

Таким образом, для обработки события необходимо расширением соответствующего интерфейса создать класс обработки события, а затем объект этого класса. Но этого мало. При создании компонента, в котором предполагается обрабатывать данное событие, необходимо оставить ссылку на объект, позволяющий обработать событие. Делается это с помощью специальных методов. Названия методов формируются, как и названия интерфейсов, на основе названий классов событий. Имя метода начинается словом `add`, далее следует имя события (это имя соответствующего класса без слова `Event`) и, наконец, слово `Listener`. Например, для связывания компонента с обработчиком события `WindowEvent` используют метод `addWindowListener()`. Аргументом метода указывается объект обработчика события.

Вкратце это вся схема обработки событий. Особенности ее применения рассмотрим на конкретных примерах. Рассмотрим, пример простейшего события – ActionEvent.

Событие ActionEvent

Рассмотрим появление события ActionEvent на примере нажатия на кнопку. Предположим, в нашем приложении создается кнопка сохранения файла:

```
Button save = new Button("Save");
add(save);
```

Теперь, когда окно приложения с этой кнопкой появится на экране, пользователь сможет нажать ее. В результате AWT сгенерирует ActionEvent. Чтобы получить и обработать его, необходимо зарегистрировать слушателя. Название нужного интерфейса прямо следует из названия события – ActionListener. В нем всего один метод (в некоторых слушателях их несколько), который имеет один аргумент – ActionEvent.

Объявим класс, который реализует этот интерфейс:

```
class SaveButtonListener
```

```
        implements ActionListener {
        private Frame parent;
        public SaveButtonListener(Frame parentFrame)
        {
        parent = parentFrame;
        }
        public void actionPerformed(ActionEvent e)
        {
        FileDialog fd = new FileDialog(parent,
        "Save file", FileDialog.SAVE);
        fd.setVisible(true);
        System.out.println(fd.getDirectory()+"/"+
        fd.getFile());
        }
       }
```

Конструктор класса требует в качестве параметра ссылку на родительский фрейм, без которого не удастся создать FileDialog. В методе actionPerformed класса ActionListener описываются действия, которые необходимо предпринять по нажатию пользователем на кнопку. А именно, открывается файловый диалог, с помощью которого определяется путь сохранения файла. Для нашего примера достаточно вывести этот путь на консоль.

Следующий шаг – регистрация слушателя. Название соответствующего метода снова прямо следует из названия интерфейса – addActionListener.

```
save.addActionListener(
 new SaveButtonListener(frame));
```

Все необходимое для обработки нажатия пользователем на кнопку сделано. Ниже приведен полный листинг программы:

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

public class Test {
 public static void main(String args[]) {
 Frame frame = new Frame("Test Action");
 frame.setSize(400, 300);
 Panel p = new Panel();
 frame.add(p);
 Button save = new Button("Save");
 save.addActionListener(
 new SaveButtonListener(frame));
 p.add(save);

 frame.setVisible(true);
 }
}

class SaveButtonListener
 implements ActionListener {
 private Frame parent;
 public SaveButtonListener(Frame parentFrame)
 {
 parent = parentFrame;
```

```
 }
 public void actionPerformed(ActionEvent e)
 {
 FileDialog fd = new FileDialog(parent,
        "Save file", FileDialog.SAVE);
 fd.setVisible(true);
 System.out.println(fd.getDirectory()+
        fd.getFile());
 }
}
```

После запуска программы появится фрейм с одной кнопкой "Save". Если нажать на нее, откроется файловый диалог. После выбора файла на консоли отображается полный путь к нему.

Однако, закрыть окно мы не сможем, т.к. не определили обработку соответствующего события. Исправим данное недоразумение. Создадим класс MyAdapter для обработки события закрытия окна. Командой MyAdapter adapter=new MyAdapter() создадим объект обработчика события закрытия окна, то есть объект adapter класса MyAdapter. Команда addWindowListener(adapter) означает, что объект adapter используется для обработки событий, происходящих с окном. В свою очередь, что именно будет происходить в рамках обработки событий, определяется кодом класса MyAdapter . В этом классе описан всего один метод windowClosing() , наследуемый от класса WindowAdapter:

```
class MyAdapter extends WindowAdapter{
public void windowClosing(WindowEvent we){
System.exit(0);}
}
```

Аргументом методу передается объект события (генерируется автоматически при возникновении события). Метод windowClosing() определяет последовательность действий при попытке закрыть окно. Действие всего одно — командой System.exit(0) завершается работа всех процессов программы, в результате чего окно закрывается (аргумент метода напрямую не используется).

Еще раз вспомним, что в Java применяется достаточно нетривиальная, но эффективная модель обработки событий. В частности, для обработки происходящих в окне событий используются объекты-обработчики.

Объект, в котором может произойти событие, должен иметь ссылку на объект, обрабатывающий это событие. Для обработки событий создаются классы, на основе которых затем создаются объекты-обработчики событий. Объект adapter является примером такого объекта. С помощью метода addWindowListener() осуществляется связывание объекта, в котором происходит событие (это фрейм), с объектом, обрабатывающим событие:

```
frame.addWindowListener(adapter);
```

Обработка событий с помощью внутренних классов

Еще в лекции, посвященной объявлению классов, было указано, что в теле класса можно объявлять внутренние классы. До сих пор такая возможность не была востребована в наших примерах, однако обработка событий AWT – как раз удобный случай рассмотреть такие классы на примере анонимных классов.

Предположим, в приложение добавляется кнопка, которой следует добавить слушателя. Зачастую бывает удобно описать логику действий в отдельном методе того же класса. Если вводить слушателя, как делалось раньше – в отдельном классе, то это сразу порождает ряд неудобств: появляется новый, малосодержательный класс, которому к тому же необходимо передать ссылку на исходный класс и так далее.

Гораздо удобнее поступить следующим образом:

```
Button b = new Button();
b.addActionListener(new ActionListener()
{
 public void actionPerformed(ActionEvent e)
 {
 processButton();
 }
});
```

Рассмотрим подробно, что происходит в этом примере. Сначала создается кнопка, у которой затем вызывается метод addActionListener. Обратим внимание на аргумент этого метода. Может сложится впечатление, что производится попытка создать экземпляр интерфейса (new ActionListener()), однако это невозможно. Дело меняет фигурная скобка, которая указывает, что порождается экземпляр нового класса, объявление которого последует за этой скобкой. Класс наследуется от Object и реализует интерфейс ActionListener. Ему необходимо реализовать метод actionPerformed, что и делается. Обратите внимание на еще одну важную деталь – в этом методе вызывается processButton. Это метод, который мы планировали разместить во внешнем классе. Таким образом, внутренний класс может напрямую обращаться к методам внешнего класса.

Такой класс называется анонимным, он не имеет своего имени. Однако правило, согласно которому компилятор всегда создает .class -файл для каждого класса Java, действует и здесь. Если внешний класс называется Test, то после компиляции появится файл Test$1.class.

Создание графического окна средствами Swing

До сих пор мы рассматривали построение графических окон средствами AWT. Несколько проще создать окно с помощью утилит библиотеки Swing. Пример такой программы приведен в листинге:

```
// Подключение библиотеки:
import javax.swing.*;
// Расширение класса JFrame:
class JustAFrame extends JFrame{
```

```
// Конструктор класса:
public JustAFrame(int a,int b){
// Заголовок окна - аргумент конструктора суперкласса:
super("Простое графическое окно");
// Размеры окна:
setSize(a,b);
// Реакция на попытку закрыть окно:
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
// Отображение окна:
setVisible(true);}
}
class MyFrame{
public static void main(String args[]){
// Создание окна:
JustAFrame frame=new JustAFrame(300,200);
}}
```

Класс фрейма `JustAFrame` создается на основе класса `JFrame` библиотеки Swing. Изменился только способ обработки события закрытия окна. В данном случае этой цели служит команда

```
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
```

Аргументом методу `setDefaultCloseOperation()`, определяющему реакцию на попытку закрыть окно, передается статическая константа `EXIT_ON_CLOSE` класса `JFrame`, означающая, что работа программы будет завершена и окно закрыто.

Приложение с кнопкой

В качестве простой иллюстрации использования в главном окне компонентов и обработки базовых событий рассмотрим пример программы с графическим интерфейсом, в которой, помимо фрейма, имеется еще и кнопка. Функциональность этой кнопки реализуется путем создания обработчиков событий. Программный код приведен в листинге:

```
package demoswingbutton;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Random;

class MyFrame extends JFrame{
// Счетчик окон:
public static int count=0;
// Конструктор:
MyFrame(int a,int b){
count++; // Количество открытых окон
// Название окна:
setTitle("Окно с кнопкой: "+count);
// Создание панели:
MyPanel panel=new MyPanel();
setSize(300,200); // Размер окна
// Закрытие окна:
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setLocation(a,b); // Положение окна
add(panel); // Добавление панели
```

```
setVisible(true); // Отображение окна
}}
// Класс панели:
class MyPanel extends JPanel{
// Конструктор:
MyPanel(){
// Создание кнопки:
JButton button=new JButton("Создать новое окно");
add(button); // Добавление кнопки на панель
button.addActionListener(listener);} // Регистрация обработчика
// Обработчик для кнопки - объект анонимного класса:
ActionListener listener=new ActionListener(){
public void actionPerformed(ActionEvent event){
Random rnd=new Random();
// Создание окна со случайными координатами размещения на экране:
MyFrame frame=new MyFrame(rnd.nextInt(800),rnd.nextInt(500));}};
}

public class Demoswingbutton {
    public static void main(String[] args) {
// Создание окна:
MyFrame frame=new MyFrame(100,100);
    }
```

Действие программы состоит в следующем: при запуске программы открывается окно с кнопкой и названием Окно с кнопкой: 1. Кнопка имеет название Создать новое окно. При щелчке на кнопке открывается еще одно окно — практически копия первого. Отличается только номер в названии. Расположение нового окна на экране выбирается случайным образом. Если закрыть хотя бы одно окно щелчком на системной кнопке в строке заголовка окна, закрываются все окна.

Во фрейм панель добавляется командой `add(panel)`, для чего вызывается стандартный метод `add()`, аргументом которого указывается добавляемый в контейнер объект. Сам контейнер определяется как объект, из которого вызывается метод `add()`.

Размер создаваемого окна задается командой `setSize(300, 200)`, расположение на экране — командой `setLocation(a, b)` (аргументы конструктора `a` и `b` определяют координату левого верхнего угла фрейма). Реакция фрейма на щелчок на системной кнопке, как и в предыдущем примере, определяется командой

```
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
```

Отображается окно командой `setVisible(true)`. На этом описание класса фрейма заканчивается. Особенность. в данном случае класса, связана с явным определением позиции окна на экране и добавлением на фрейм панели.

Класс панели `MyPanel` создается на основе класса `JPanel` библиотеки Swing. В конструкторе класса на панель добавляется кнопка. Объект кнопки `button` (объект класса `JButton` библиотеки Swing) создается командой

```
JButton button=new JButton("Создать новое окно")
```

Название кнопки передается аргументом конструктору класса `JButton`. Добавление кнопки на панель осуществляется командой `add(button)`. Командой

button. addActionListener(listener) для кнопки регистрируется обработчик — объект listener анонимного класса, созданного на основе класса ActionListener. При создании объекта описывается метод actionPerformed(), в котором создается окно, чье положение на экране определяется случайным образом. В главном методе приложения в классе FrameAndButton создается первое окно командой

```
MyFrame frame=new MyFrame(100,100)
```