Программирование на языке Java. Модели обработки событий

Картузов А.В.

Несмотря на существенные изменения механизма обработки событий в AWT, Java1.1 поддерживает обратную совместимость с моделью обработки событий, принятой в Java 1.0. Однако такая совместимость относится к типу "все или ничего"—эти две модели настолько отличаются друг от друга, что их невозможно использовать в одном приложении одновременно.

1. Модель обработки событий Java 1.0

Все компоненты, которые мы с вами до сих пор создавали, выглядели неплохо, но были абсолютно бесполезны, поскольку мы не говорили о том, как можно обрабатывать ввод пользователя, осуществляемый с помощью этих управляющих элементов пользовательского интерфейса.

Каждый компонент может обрабатывать события, заместив определенные методы, вызываемые используемой по умолчанию реализацией метода handleEvents класса Component. Этот метод вызывается с объектом класса Event, описывающего все возможные типы событий. Наиболее часто используемые события, например, те, что связаны с мышью и клавиатурой, диспетчеризируются другим методам класса Component.

Все события, связанные с мышью, вызываются с копией оригинального события, а также с координатами x и y, в которых это событие произошло.

- mouseEnter вызывается в том случае, когда мышь входит в компонент.
- mouseExit вызывается при выходе мыши из области компонента.
- mouseMove вызывается при перемещении мыши в области компонента.
- mouseDown вызывается при нажатии кнопки мыши.

- mouseDrag вызывается при перемещении мыши с нажатой кнопкой.
- mouseUp вызывается при отпускании кнопки мыши.

Аналогично, keyDown и keyUp вызываются при каждом нажатии и отпускании клавиши. Событие передается методу вместе с кодом нажатой клавиши. Событие можно проверить, чтобы посмотреть, нажаты ли в данный момент какие либо клавиши-модификаторы, для этой цели можно также пользоваться методами shiftDown, controlDown и metaDown. В классе Event определены десятки констант, позволяющих использовать символические имена, например, PGUP и HOME.

Наконец, для работы со специальными событиями, например, с обратными вызовами (callback) из компонентов Button, Scrollbar и Menu, вам придется замещать метод action. Этот метод вызывается с исходным событием и со вторым параметром, который представляет собой компонент пользовательского интерфейса, создавший это событие. Вы должны проверить этот объект, разобраться, какой из компонентов послал вам событие, после чего передать управление соответствующему данному компоненту обработчику. Для того, чтобы перед приведением типа проверить, принадлежит ли объект к определенному классу, например, к классу Button, вы можете использовать оператор instanceof.

А вот и пример на обработку событий. Мы добавили объект Label к примеру с игрой в "пятнашки", а также заместили метод action для того, чтобы обрабатывать события, возникающие при нажатии кнопок. Точно такой же механизм можно использовать для управления вводом через любой из подклассов Component.

```
/* <applet code = "EventDemo" width=200 height=200>
   </applet>
*/
import java.awt.*;
import java.applet.*;
public class EventDemo extends Applet {
        static final int n = 4;
        Label lab = new Label("?", Label.CENTER);
public void init() {
        setLayout(new GridLayout(n, n));
        setFont(new Font("Helvetica", Font.BOLD, 24));
        int width = Integer.parseInt(getParameter("width"));
        int height = Integer.parseInt(getParameter("height"));
```

```
        for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                        int k = i * n + j;
                        if (k > 0)
                                add(new Button("" + k));
                        }
                }
        lab.setFont(new Font("Helvetica", Font.ITALIC, 24));
        add(lab);
        }
public boolean action(Event e, Object o) {
        if (o instanceof String) {
                lab.setText((String) o);
                }
        return false;
        }
}
```

2. Элементы и связанные с ними события

В таблице 1 для каждого элемента пакета AWT перечислены типы событий, которые он может порождать. В первом столбце таблицы указан тип элемента, а во втором—тип соответствующего ему события. Тип события представляет собой константу, которая записывается в переменную id объекта класса Event.

В столбцах с третьего по седьмой указано, устанавливаются ли значения переменных -when (время события), x (координата x курсора мыши), y (координата y курсора мыши), key (нажатая кнопка) и modifiers (специальные клавиши, которые нажаты при этом) для данного события. Если в столбце стоит точка, значит, событие устанавливает значение соответствующей переменной. В восьмом столбце объяснено, что порождает данное событие и приведено значение, которое записывается в переменной arg объекта класса Event.

События, перечисленные для элементов класса Component, применимы ко всем подклассам класса java.awt. Component, а события, приведенные для элементов класса window, относятся как к подклассам класса window, так и к классам Dialog и Frame.

Элемент	Тип события	w h	x	y	k e	m o	Смысл события	Тип и значение

	(id)	e r e			y	d s		переменной arg
Button	ACTION_EVENT						Пользователь нажал кнопку	String: обозначение кнопки
Checkbox	ACTION_EVENT						Пользователь активизировал флажок	Boolean: новое состояние флажка
Choice (список выбора)	ACTION_EVENT						Пользователь выбрал элемент списка	String: обозначение выбранного элемента
Element (элемент)	GOT_FOCUS						Получение фокуса ввода	не используется
Element (элемент)	KEY_ACTION	*	*	*	*	*	Пользователь нажал функциональную клавишу	не используется, поскольку key содержит константу клавиши
Element (элемент)	KEY_ACTION_RELEASE	*	*	*	*	*	Пользователь отпустил функциональную клавишу	не используется, поскольку key содержит константу клавиши
Element (элемент)	KEY_PRESS	*	*	*	*	*	Пользователь нажал клавишу	не используется, поскольку key содержит

								ASCII-код клавиши
Element (элемент)	KEY_RELEASE	*	*	*	*		Пользователь отпустил клавишу	не используется, поскольку key содержит ASCII-код клавиши
Element (элемент)	LOST_FOCUS						Потеря фокуса ввода	не используется
Element (элемент)	MOUSE_ENTER	*	*				Курсор мыши попал в область объекта класса Component	не используется
Element (элемент)	MOUSE_EXIT	*	*				Курсор мыши вышел из области объекта класса Component	не используется
Element (элемент)	MOUSE_DOWN	*	*		*		Пользователь нажал кнопку мыши	не используется
Element (элемент)	MOUSE_UP	*	*		*		Пользователь отпустил кнопку мыши	не используется
Element (элемент)	MOUSE_MOVE	*	*		*		Пользователь переместил	не используется

							мышь	
Element (элемент)	MOUSE_DRAG	*	*	*		*	Пользователь переместил мышь при нажатой кнопке мыши	не используется
List (список)	ACTION_EVENT						Пользователь выполнил двойной щелчок мыши на элементе списка	String: обозначение выбранного элемента
List (список)	LIST_SELECT						Пользователь выбрал элемент списка	Integer: индекс выбранного элемента
List (список)	LIST_DESELECT						Пользователь убрал выделение с определенного элемента	Integer: индекс элемента
Menu Item (меню)	ACTION_EVENT						Пользователь выбрал пункт меню	String: обозначение выбранного пункта
Scrollbar (полоса прокрутки)	SCROLL_LINE_UP						Пользователь осуществил прокрутку вверх на строку	Integer: позиция, до которой осуществляется прокрутка
Scrollbar	SCROLL_LINE_DOWN						Пользователь	Integer:

(полоса прокрутки)							осуществил прокрутку вниз на строку	позиция, до которой осуществляется прокрутка
Scrollbar (полоса прокрутки)	SCROLL_PAGE_UP						Пользователь осуществил прокрутку вверх на страницу	Integer: позиция, до которой осуществляется прокрутка
Scrollbar (полоса прокрутки)	SCROLL_PAGE_DOWN						Пользователь осуществил прокрутку вниз на страницу	Integer: позиция, до которой осуществляется прокрутка
Scrollbar (полоса прокрутки)	SCROLL_ABSOLUTE						Пользователь переместил ползунок полосы прокрутки	Integer: позиция, до которой осуществляется прокрутка
Text Field (текст)	ACTION_EVENT						Пользователь ввел текст и нажал [Enter].	String: введенный текст
Window (окно)	WINDOW_DESTROY						Окно закрыто	не используется
Window (окно)	WINDOW_ICONIFY						Окно представлено в виде пиктограммы	не используется
Window (окно)	WINDOW_DEICONIFY						Окно восстановлено	не используется

Window (окно)	WINDOW_MOVED	*	*			Окно перемещено	не используется	

Таблица 1. Элементы AWT и события Java 1.0, которые порождаются ими

3. Рисование "каракулей" в Java 1.0

Классический апплет, в котором используется модель обработки событий Java 1.0. В нем методы mouseDown() и mouseDragO переопределены таким образом, чтобы пользователь имел возможность рисовать "каракули" с помощью мыши. Также переопределен метод keyDown(), чтобы при нажатии клавиши [C] экран очищался, и метод action(), чтобы экран очищался после щелчка на кнопке Clear.

```
/* <applet code = "Scribble1" width=200 height=200>
   </applet>
*/

import java.applet.*;
import java.awt.*;
/** Простой апплет, в котором используется модель обработки событий 1.0 */
public class Scribble1 extends Applet {
        private int lastx, lasty; // Хранят координаты курсора мыши.
        Button clear_button;      // Кнопка Clear.
        Graphics g;               // Объект Graphics,
                                  // который необходимо нарисовать.
        /** Инициализация кнопки и объекта Graphics */
        public void init() {
                clear_button = new Button("Clear");
                this.add(clear_button);
                g = this.getGraphics();
                }
        /** Реакция на нажатие кнопки мыши */
        public boolean mouseDown(Event e, int x, int y) {
                lastx = x; lasty = y;
                return true;
                }
        /** Реакция на перетаскивание с помощью мыши */
        public boolean mouseDrag(Event e, int x, int y) {
                g.setColor(Color.black) ;
```

```
                g.drawLine(lastx, lasty, x, y);
                lastx = x; lasty = y;
                return true;
                }
        /** Реакция на нажатие клавиши [C] */
        public boolean keyDown(Event e, int key) {
                if ((e.id == Event.KEY_PRESS) && (key == 'c' ) ) {
                        clear() ;
                        return true;
                        }
                        else return false;
                }
        /** Реакция на нажатие кнопки Clear */
        public boolean action(Event e, Object arg) {
                if (e.target == clear_button) {
                        clear();
                        return true;
                        }
                else return false;
        }
        /** Метод для стирания каракулей */
        public void clear() {
                g.setColor(this.getBackground());
                g.fillRect(0, 0, bounds().width, bounds().height);
                }
        }
```

4. Модель обработки событий Java 1.1

Новая модель обработки событий представляет собой, по существу, модель обратных вызовов (callback). При создании GUI-элемента ему сообщается, какой метод или методы он должен вызывать при возникновении в нем определенного события (нажатия кнопки, мыши и т.п.). Эту модель очень легко использовать в C++, поскольку этот язык позволяет оперировать указателями на методы (чтобы определить обратный вызов, необходимо всего лишь передать указатель на функцию). Однако в Java это недопустимо (методы не являются объектами). Поэтому для реализации новой модели необходимо определить класс, реализующий некоторый специальный интерфейс. Затем можно передать экземпляр такого класса GUI-элементу, обеспечивая таким

образом обратный вызов. Когда наступит ожидаемое событие, GUI-элемент вызовет соответствующий метод объекта, определенного ранее.

Модель обработки событий Java 1.1 используется как в пакете AWT, так и в JavaBeans API. В этой модели разным типам событий соответствуют различные классы Java. Каждое событие является подклассом класса java.util.EventObject. События пакета AWT, которые и рассматриваются в данной главе, являются подклассом java.awt.AWTEvent. Для удобства события различных типов пакета AWT (например, MouseEvent или ActionEvent) помещены в новый пакет java.awt.event.

Для каждого события существует порождающий его объект, который можно получить с помощью метода getSource(), и каждому событию пакета AWT соответствует определенный идентификатор, который позволяет получить метод getid(). Это значение используется для того, чтобы отличать события различных типов, которые могут описываться одним и тем же классом событий. Например, для класса FocusEvent возможны два типа событий: FocusEvent.FOCUS_GAINED и FocusEvent.FOCUS_LOST. Подклассы событий содержат информацию, связанную с данным типом события. Например, в классе MouseEvent существуют методы getX(), getY() и getClickCount (). Этот класс наследует, в числе прочих, и методы getModifiers() и getWhen().

Модель обработки событий Java 1.1 базируется на концепции слушателя событий. Слушателем события является объект, заинтересованный в получении данного события. В объекте, который порождает событие (в источнике событий), содержится список слушателей, заинтересованных в получении уведомления о том, что данное событие произошло, а также методы, которые позволяют слушателям добавлять или удалять себя из этого списка. Когда источник порождает событие (или когда объект источника зарегистрирует событие, связанное с вводом информации пользователем), он оповещает все объекты слушателей событий о том, что данное событие произошло.

Источник события оповещает объект слушателя путем вызова специального метода и передачи ему объекта события (экземпляра подкласса EventObject). Для того чтобы источник мог вызвать данный метод, он должен быть реализован для каждого слушателя. Это объясняется тем, что все слушатели событий определенного типа должны реализовывать соответствующий интерфейс. Например, объекты слушателей событий ActionEvent должны реализовывать интерфейс ActionListener. В пакете

Java.awt.event определены интерфейсы слушателей для каждого из определенных в нем типов событий (например, для событий MouseEvent здесь 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 интерфейса-метки, в котором однозначно определены все слушатели событий как таковые.

В интерфейсе слушателя событий может определяться 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. По установленному соглашению, методам слушателей событий может быть передан один единственный аргумент, являющийся объектом того события, которое соответствует данному слушателю. В этом объекте должна содержаться вся информация, необходимая программе для формирования реакции на данное событие. В таблице 2 приведены определенные в пакете java.awt.event типы событий, соответствующие им слушатели, а также методы, определенные в каждом интерфейсе слушателя.

Класс события	Интерфейс слушателя	Методы слушателя
ActionEvent	ActionListener	actionPerformed()
AdjustmentEvent	AdjustmentListener	adjustmentValueChanged()
ComponentEvent	ComponentListener	componentHidden() componentMoved() componentResized() componentShown()
ContainerEvent	ContainerListener	componentAdded() componentRemoved()
FocusEvent	FocusListener	focusGained() focusLost()
ItemEvent	ItemListener	itemStateChanged()
KeyEvent	KeyListener	keyPressed() keyReleased() keyTyped()

MouseEvent	MouseListener	mouseClicked() mouseEntered() mouseExited() mousePressed() mouseReleased()
	MouseMotionListener	mouseDragged() mouseMoved()
TextEvent	TextListener	textValueChanged()
WindowEvent	WindowListener	windowActivated() windowClosed() windowClosing() windowDeactivated() windowDeiconified() windowIconified() windowOpened()

Таблица 2. Типы событий, слушатели и методы слушателей в Java 1.1

Для каждого интерфейса слушателей событий, содержащего несколько методов, в пакете java.awt.event определен простой класс-адаптер, который обеспечивает пустое тело для каждого из методов соответствующего интерфейса. Когда нужен только один или два таких метода, иногда проще получить подкласс класса-адаптера, чем реализовать интерфейс самостоятельно. При получении подкласса 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 так же, как и интерфейсы, которые они реализуют, но в этих названиях Listener заменяется на Adapter: MouseAdapter, WindowAdapter и т.д.

Как только реализован интерфейс слушателя или получены подклассы класса-адаптера, необходимо создать экземпляр нового класса, чтобы определить конкретный объект слушателя событий. Затем этот слушатель должен быть зарегистрирован соответствующим источником событий. В программах пакета AWT источником событий всегда является какой-нибудь элемент пакета. В методах регистрации слушателей событий используются стандартные соглашения об именах: если источник событий порождает события типа X, в нем существует метод addXListener () для добавления слушателя и метод removeXListener() для его удаления.

Одной из приятных особенностей модели обработки событий Java 1.1 является возможность легко определять типы событий, которые могут порождаться данным элементом. Для этого следует просто просмотреть, какие методы зарегистрированы для его слушателя событий. Например, из описания API для объекта класса Button следует, что он порождает события ActionEvent. В таблице 3 приведен список элементов пакета AWT и событий, которые они порождают.

Элемент	Порождаемое событие	Значение
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

Таблица 3. Элементы пакета AWT и порождаемые ими события в Java1.1

5. Рисование "каракулей" в Java 1.1

Модель обработки событий Java 1.1 является достаточно гибкой и предоставляет пользователю ряд возможностей для структуризации программы обработки событий. Первый из этих способов продемонстрирован в примере. В апплете данной версии реализованы интерфейсы MouseListener и MouseMotionListener, регистрирующие себя с помощью своих же методов addMouseListener() и addMouseMotionListener().

```
/* <applet code = "Scribble2" width=200 height=200>
   </applet>
*/

import java.applet.*;
import java.awt.*;
import java.awt.event.*;
        public class Scribble2 extends Applet implements
```

```
        MouseListener, MouseMotionListener {
                private int last_x, last_y;
                public void init() {
                // Сообщает данному апплету о том, какие объекты
                // классов MouseListener и MouseMotionListener
                // он должен оповещать
                // о событиях, связанных с мышью и ее перемещением.
                // Поскольку интерфейс реализуется в самом апплете,
                // при этом будут вызываться методы апплета.
                this.addMouseListener(this) ;
                this.addMouseMotionListener(this);
                }
        // Метод интерфейса MouseListener. Вызывается при нажатии
        // пользователем кнопки мыши.
        public void mousePressed(MouseEvent e) {
                last_x = e.getX();
                last_y = e.getY();
                }
        // Метод интерфейса MouseMotionListener. Вызывается при
        // перемещении мыши с нажатой кнопкой.
        public void mouseDragged(MouseEvent e) {
                Graphics g = this.getGraphics();
                int x = e.getX(), y = e.getY();
                g.drawLine(last_x, last_y, x, y);
                last_x = x; last_y = y;
                }
        // Другие, не используемые методы интерфейса MouseListener.
        public void mouseReleased(MouseEvent e) {;}
        public void mouseClicked(MouseEvent e) {;}
        public void mouseEntered(MouseEvent e) {;}
        public void mouseExited(MouseEvent e) {;}
        // Другой метод интерфейса MouseMotionListener.
        public void mouseMoved(MouseEvent e) {;}
        }
```

6. Рисование "каракулей" с использованием встроенных классов

Модель обработки событий Java 1.1 разработана с учетом того, чтобы хорошо

сочетаться с другой новой особенностью Java 1.1: встроенными классами (глава, посвященная им, еще не написана ;-(). В следующем примере показано, как изменится данный апплет, если слушатели событий будут реализованы в виде анонимных встроенных классов. Обратите внимание на компактность данного варианта программы. Новая особенность, добавленная в апплет—кнопка Clear. Для этой кнопки зарегистрирован объект ActionListener, а сама она выполняет очистку экрана при наступлении соответствующего события.

```
/* <applet code = "Scribble3" width=200 height=200>
   </applet>
*/

import java.applet.*;
import java.awt.*;
import java.awt.event.*;
        public class Scribble3 extends Applet {
                int last_x, last_y;
        public void init() {
                // Определяет, создает и регистрирует объект MouseListener.
                this.addMouseListener(new MouseAdapter() {
                        public void mousePressed(MouseEvent e) {
                                last_x = e.getX(); last_y = e.getY();
                                }
                        }
                );
                // Определяет, создает и
                // регистрирует объект MouseMotionListener.
        this.addMouseMotionListener(new MouseMotionAdapter() {
        public void mouseDragged(MouseEvent e) {
                Graphics g = getGraphics();
                int x = e.getX(), y= e.getY();
                g.setColor(Color.black);
                g.drawLine(last_x, last_y, x, y);
                last_x = x; last_y = y;
                }
        });
        // Создает кнопку Clear.
        Button b = new Button("Clear");
```

```
        // Определяет, создает и регистрирует объект слушателя
        // для обработки события, связанного с нажатием кнопки.
        b.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                // стирание каракулей
                Graphics g = getGraphics();
                g.setColor(getBackground());
                g.fillRect(0, 0, getSize().width, getSize().height);
                }
        });
        // Добавляет кнопку в апплет.
        this.add(b);
        }
}
```