Начинаем программировать на языке Java. Часть 5

Дмитрий Рамодин

Как и было обещано, с этого занятия мы начинаем практиковаться в написании программ на языке Java. Иногда, правда, по ходу дела мы коснемся некоторых теоретических моментов, если это будет необходимо для понимания работы программы. На этом занятии мы будем разрабатывать аплет, который поможет понять механизм сообщений языка Java. Поэтому для начала рассмотрим основные события, происходящие в процессе работы Java-программ, и методы их перехвата.

1. Мир сообщений

Как известно, работа любого Windows-приложения основана на обработке сообщений. Сообщения — это асинхронные (т. е. они могут произойти в любой момент) вызовы специальных методов, называемых обработчиками сообщений. Через вызовы обработчиков сообщений система уведомляет, что в программе произошло некоторое событие, например в окне программы был произведен щелчок мышью. Если система Windows имеет список из сотен событий, которые могут произойти в процессе работы программы, то Java-приложения в этом плане гораздо проще. Всего лишь 13 элементарных событий могут случиться в программе Java.

Эти так называемые элементарные события в свою очередь могут вызвать возникновение более сложных событий. К примеру, если пользователь щелкнул мышью в окне Java-программы, то происходит событие MOUSE_DOWN, но если он это сделал на полосе прокрутки, то порождается другое событие, скажем, SCROLL_LINE_UP. Это уже вторичное событие.

Пусть вас не пугает, что описания сообщений выглядят несколько запутанно, — вам очень редко придется иметь с ними дело. Сам язык Java предлагает вам посильную

помощь, вызывая для каждого из описанных выше собщений отдельный метод-обработчик. Все, что вам требуется, это описать в своем классе нужный обработчик события, а виртуальная машина Java сама вызовет его в тот момент, когда это событие произойдет. Единственное условие при написании таких методов обработки событий — соблюдайте синтаксис, который задан спецификацией языка Java. Если же вы ошибетесь в описании, то в лучшем случае компилятор сообщит об этом, в худшем — будет создан перегруженный метод (помните, Java, как и Си++, допускает перегруженные методы), который не будет ни разу вызван.

Заметим, что аргумент evt типа Event — это то самое событие, которое заставило систему вызвать обработчик того или иного события. Этот параметр редко используется, если только нет необходимости узнать какие-то частности относительно произошедшего события, как, например, точное время, когда оно произошло.

Еще один важный момент, который необходимо помнить, это возвращаемое обработчиками значение. Если сообщение обработано вашим методом, то он возвращает значение true, иначе сообщение будет передаваться по цепочке вверх по иерархии компонентов, пока не будет обработано. За редким исключением в обработчиках событий своих программ всегда будет возвращаться true.

2. За работу!

Самое время написать какой-нибудь аплет. Я пользуюсь в своей работе набором Java Developer Kit 1.1 (JDK). И не потому, что это удобно. Как раз наоборот: Symantec Visual Cafe или Microsoft Visual J++ гораздо лучше подходят для работы. Зато JDK — это стандартное средство, и к тому же его можно получить совершенно бесплатно с сервера http://java.sun.com.

Чтобы разобраться с тем, как сообщения курсируют по программе, напишем такой аплет, который будет говорить нам обо всех происходящих событиях. Для показа сообщений будем использовать стандартный поток вывода, т. е. экран монитора. Вывод сообщений будет производиться вызовами функций вывода System. out.println(), которые часто используются для трассировки различных данных во время отладки. System. out.println() показывает на экране заданную вами строку и переводит курсор на следующую строку. Если перевод курсора не требуется, то можно

воспользоваться функцией System.out.print().

Наберите следующий исходный текст в файл с именем Events.java:

```
import java.applet.*;
// Импортировать данные для класса Applet
import java.awt.*;
// Импортировать данные для классов
// визуальных элементов
import java.io.*;
// Нужно для вызовов
// System.out.println
// Создание класса аплета
public class Events extends Applet
{
Button button;
MyTextField text;

public Events()
// Конструктор класса
// Events
{
        // Вывести сообщение на дисплей
        System.out.println("-->
        Constructor     called...");
        // Создать объект "кнопка"
        // с надписью "Button"
        button = new Button("Button");
        // Создать объект
        // "строка редактирования"
        text = new TextField();
        // Добавить компоненты в окно аплета
        add(button);
        add(text);
        // Проверка правильности
        // расположения компонентов
        validate();
}

// Вызывается при начальной
```

```
// инициализации аплета
public void init()
{
 System.out.println("--> init()
called...");
        // this.requestFocus();
}

// Обработчик перерисовки
public void paint(Graphics g)
{
 System.out.println("--> paint()
called...");
        // Разместить кнопку с координата-
        // ми (10,10) и сделать ее размером
        // 100x20 button.reshape(10, 10,
        // 100, 20); разместить строку
        // ввода с координатами (10,40)
        // и сделать ее размером 100x20
        text.reshape(10, 40, 100, 20);
}

        public boolean mouseDown(Event evt,
int x, int y)
        {
 System.out.println("--> mouseDown()
called...");
                return true;
        }

 public boolean mouseUp(Event evt,
int x, int y)
        {
 System.out.println("--> mouseUp()
called...");
                return true;
        }

 public boolean mouseDrag(Event evt,
int x, int y)
```

```
        {
 System.out.println("--> mouseDrag()
called...");
 return true;
 }

 public boolean mouseMove(Event evt,
int x, int y)
        {
 System.out.println("--> mouseMove()
called...");
 return true;
 }

 public boolean mouseEnter(Event evt,
int x, int y)
        {
 System.out.println("-->
mouseEnter() called...");
 return true;
        }

 public boolean mouseExit(Event evt,
int x, int y)
        {
 System.out.println("--> mouseExit()
called...");
 return true;
        }

 public boolean keyDown(Event evt,
int key)
        {
 System.out.println("--> keyDown()
called...");
 return true;
        }

 public boolean keyUp(Event evt,
int key)
```

```
        {
 System.out.println("--> keyUp()
called...");
 return true;
        }

 public boolean gotFocus(Event evt,
Object o)
        {
 System.out.println("--> gotFocus()
called...");
 return true;
        }

 public boolean lostFocus(Event evt,
Object o)
        {
 System.out.println("--> lostFocus()
called...");
 return true;
        }

 public boolean action(Event evt,
Object o)
        {
 System.out.println("--> action()
called...");
 return true;
        }
}
```

Если вы будете использовать JDK, то скомпилируйте исходный текст командой

```
javac.exe Events.java
```

Разумеется, вы должны проследить за тем, чтобы ваш проект был виден компилятору, иначе компилятор не сможет найти файл с исходным текстом.

Следующий ход — создание Web-страницы со ссылкой на полученный аплет. Сделаем файл Events.html и наберем следующий исходный текст на языке HTML:

```
<html>
<head>
<title>Events</title>
</head>
<body>
<applet code=Events.class
        width=150 height=200 >
</applet>
</body>
</html>
```

Все готово? Тогда запускаем наш аплет:

```
appletviewer.exe
Events.html
```

Поэкспериментируйте, нажимая кнопки мыши и клавиатуры. Посмотрите в DOS-окно на протокол, полученный в результате нажатий. Обратите внимание на некоторые странности в поведении нашего аплета. Во-первых, при попадании указателя мыши на элементы управления возникает событие MOUSE_EXIT, а при переводе указателя назад в окно аплета следует событие MOUSE_ENTER.

Вывод: области, занятые элементами интерфейса, исключаются из окна аплета, и события от мыши в этих областях не обрабатываются.

Во-вторых, не происходит ни одного вызова обработчиков нажатий и отпусканий клавиш клавиатуры. Это говорит о том, что аплет не имеет фокуса ввода. Нет фокуса и у элементов интерфейса. Если нажать клавишу <Tab>, то передачи фокуса не происходит.

Вывод: после создания аплета фокуса ввода нет ни у одного элемента интерфейса.

Если вы хотите, чтобы нажатия клавиш отслеживались аплетом, вам необходимо сделать так, чтобы аплет получил фокус. На самом деле такая строка уже есть:

```
// this.requestFocus();
```

Все, что от вас требуется, это убрать признак комментария — две косые черты перед строкой, и перекомпилировать проект. Если теперь запустить аплет и начать нажимать

кнопки клавиатуры, то вы увидите, что обработчики нажатий и отпусканий клавиш начали вызываться. Если нажать клавишу <Tab>, фокус ввода перейдет к кнопке. При этом вы увидите, что вызываются обработчики lostFocus() и gotFocus(), отмечающие потерю фокуса ввода одним элементом и получение его другим.

Кстати говоря, мы с вами еще не открыли тайну обработчика action(), а ведь он один из основных. Он вызывается в том случае, когда происходит главное функциональное событие элемента интерфейса. Для классов Button и Checkbox это будет щелчок мыши, для классов Choice, List и MenuItem — выбор элемента, для класса TextField — нажатие клавиши Return. В большинстве случаев вам как программисту будет достаточно именно этого обработчика при решении практически всех задач. Для того чтобы узнать, какой именно элемент интерфейса вызвал обработчик action(), нужно проверить параметр target переданного класса Event. Это ссылка на сработавший объект. Проверить правильность этого факта можно, добавив в обработчик action() пару строчек:

```
if(evt.target==button)
System.out.println("--> Button pressed");
if(evt.target==text)
System.out.println("--> Text edited");
```

Запустите перекомпилированный аплет, понажимайте на кнопку и понабирайте текст в строке ввода, завершая набор нажатием клавиши Return. Сообщения не замедлят появиться в DOS-окне.

Основная работа, связанная с обработкой сообщений аплета, приходится на метод handleEvent(). Для того чтобы убедиться в этом, напишем свой обработчик сообщений handleEvent():

```
public boolean handleEvent(Event evt)
{
  System.out.println
("-- > handleEvent() called_");
  return false;
}
```

Последняя строка возвращает константу false, говорящую о том, что сообщение не обработано. Откомпилируем и запустим модифицированный вариант аплета, наблюдая

за диагностическими сообщениями в окне DOS. Как вы можете видеть, теперь весь процесс обработки сообщений сводится к вызовам метода handleEvent(), после которого ничего не происходит. Изменим строку

```
return false
```

на вызов метода handleEvent() класса Applet, который является предком для нашего аплета:

```
return super.handleEvent(evt);
```

Напомню, что ключевое слово super обозначает ссылку на непосредственного предка класса. В принципе это эквивалентно следующей строке:

```
return Applet.handleEvent(evt);
```

К сожалению, компилятор Java выдаст на это выражение сообщение об ошибке, отказываясь делать статическую ссылку на метод объекта. Поэтому все равно придется использовать слово super.

После того как мы изменили содержимое метода handleEvent(), вызовы обработчиков элементарных сообщений (например, mouseMove() и т. п.) снова посыпались как из рога изобилия. Также восстановился поток вызовов метода action(). Обратите внимание, что каждому вызову обработчика элементарного сообщения предшествует вызов метода handleEvent().

Вывод: любое сообщение передается методу handleEvent(), который должен либо обработать его самостоятельно, либо передать методу handleEvent() класса-предка. Последний уже производит разбор сообщений на элементарные и вызывает соответствующие им обработчики. Метод handleEvent() класса-предка также производит диспетчеризацию вызовов метода action().

Следующий этап нашего эксперимента — проверка пересылки сообщений между элементами пользовательского интерфейса и аплетом. Создадим свой класс кнопки и назовем его MyButton. Добавьте в файл Events.java следующий исходный текст:

```
class MyButton extends Button
{
```

```
        public MyButton(String text)
        {
                super(text);
        }

        public boolean mouseDown(Event evt,
int x, int y)
        {
                System.out.println
("--> MyButton.mouseDown() called...");
                return true;
        }

        public boolean mouseUp(Event evt,
int x, int y)
        {
                System.out.println("-->
MyButton.mouseUp() called...");
                return true;
        }

        public boolean mouseDrag(Event evt,
int x, int y)
        {
                System.out.println("-->
MyButton.mouseDrag() called...");
                return true;
        }

        public boolean mouseMove(Event evt,
int x, int y)
        {
                System.out.println("-->
MyButton.mouseMove() called...");
                return true;
        }

        public boolean mouseEnter(Event evt,
int x, int y)
        {
```

```
                System.out.println("-->
MyButton.mouseEnter() called...");
                return true;
        }

        public boolean mouseExit(Event evt,
int x, int y)
        {
                System.out.println("-->
MyButton.mouseExit() called...");
                return true;
        }

        public boolean keyDown(Event evt,
int key)
        {
                System.out.println("-->
MyButton.keyDown() called...");
        }

        public boolean keyUp(Event evt,
int key)
        {
                System.out.println("-->
MyButton.keyUp() called...");
                return true;
        }

        public boolean gotFocus(Event evt,
Object o)
        {
                System.out.println("-->
MyButton.gotFocus() called...");
                return true;
        }

        public boolean lostFocus
(Event evt, Object o)
        {
                System.out.println("-->
```

```
MyButton.lostFocus() called...");
                return true;
        }
}
```

Замените также в исходном тексте аплета все упоминания класса Button на MyButton. Перекомпилируйте и запустите аплет. Попробуйте поэкспериментировать с нажатиями на кнопку. Как вы заметили, все события стали передаваться непосредственно классу кнопки. Это же происходит и с нажатиями на кнопки клавиатуры.

Вывод: все события сначала передаются элементу интерфейса, с которым работает пользователь, и лишь потом, если разрешит обработчик события, передаются классу контейнера, в котором этот элемент содержится. Для такой передачи сообщения обработчик должен вернуть значение false.

Еще одна странность наблюдается при установленном на кнопке фокусе ввода. Если теперь нажать клавишу <Tab>, то ничего не произойдет. Фокус останется на кнопке и не будет передан на строку ввода, как это ожидается. Секрет прост: наш класс не пропускает нажатие <Tab> далее. Чтобы исправить ситуацию, вставим следующую строку в обработчик keyDown():

```
return key == 9 ? false : true;
```

Теперь, если код нажатой клавиши равен коду клавиши <Tab>, т. е. равен 9, то мы не обрабатываем событие, а возвращаем false, передавая его дальше в систему. Вывод: в Java-программах нет различия между функциональными клавишами и обычными. Поэтому вы должны сами отслеживать сомнительные моменты.

На сегодня достаточно. Вам предоставляется возможность самостоятельно поэкспериментировать с готовым аплетом.

3. Элементарные события в Java-программе

Действия пользователя	Событие	Вызов обработчика события
Пользователь произвел некоторое действие, требующее получения некоторого результата, например, нажал	ACTION_EVENT	public boolean action(Event evt /*Событие, которое вызвало действие*/, Object what /*Ссылка на объект, который

кнопку		вызвал появление события*/)
Указатель мыши переместился внутрь окна Java-программы	MOUSE_ENTER	public boolean mouseEnter(Event evt /*Сообщение, вызывавшее обработчик*/, int x /*x-координата указателя мыши*/, int y /*y-координата указателя мыши*/)
Указатель мыши переместился за пределы окна Java-программы или переместился на какой-либо элемент пользовательского интерфейса	MOUSE_EXIT	public boolean mouseExit(Event evt /*Сообщение, вызывавшее обработчик*/, int x /*x-координата указателя мыши*/, int y /*y-координата указателя мыши*/)
Указатель мыши движется в окне Java-программы	MOUSE_MOVE	public boolean mouseMove(Event evt /*Сообщение, вызывавшее обработчик*/, int x /*x-координата указателя мыши*/, int y /*y-координата указателя мыши*/)
Произошло нажатие кнопки мыши в окне Java-программы	MOUSE_DOWN	public boolean mouseDown(Event evt /*Сообщение, вызывавшее обработчик*/, int x /*x-координата указателя мыши*/, int y /*y-координата указателя мыши*/)
Произошло отпускание нажатой ранее кнопки мыши	MOUSE_UP	public boolean mouseUp(Event evt /*Сообщение, вызывавшее обработчик*/, int x /*x-координата указателя мыши*/, int y /*y-координата указателя мыши*/)
Кнопка мыши нажата, и указатель мыши движется в окне Java-программы	MOUSE_DRAG	public boolean mouseDrag(Event evt /*Сообщение, вызывавшее обработчик*/, int x /*x-координата указателя мыши*/, int y /*y-координата указателя мыши*/)

Нажата кнопка на клавиатуре	KEY_PRESS	public boolean keyDown(Event evt /*Сообщение, вызывавшее обработчик*/, int key /*Код нажатой клавиши*/)
Нажата функциональная клавиша, например <F1> или "стрелка вправо"	KEY_ACTION	public boolean keyDown(Event evt /*Сообщение, вызывавшее обработчик*/, int key /*Код нажатой клавиши*/)
Отпущена ранее нажатая кнопка клавиатуры	KEY_RELEASE	public boolean keyUp(Event evt /*Сообщение, вызывавшее обработчик*/, int key /*Код отпущенной клавиши*/)
Отпущена ранее нажатая функциональная клавиша	KEY_ACTION_	public boolean keyUp(Event evt /*Сообщение, вызывавшее RELEASE обработчик*/, int key /*Код отпущенной клавиши*/)
Компонент получил фокус, т. е. любое действие пользователя переназначено теперь на этот компонент	GOT_FOCUS	public boolean gotFocus(Event evt /*Событие, которое вызвало получение фокуса*/, Object what /*Почти всегда равно нулю*/)
Компонент "потерял" фокус, т. е. любое действие пользователя переназначено теперь на другой компонент	LOST_FOCUS	public boolean lostFocus(Event evt /*Событие, которое вызвало потерю фокуса*/, Object what /*Почти всегда равно нулю*/)

Мир ПК #05/97