1. Обработка событий

1.1. Модель делегирования событий

Современный подход к обработке событий основан на *модели делегирования событий* (delegation event model), которая определяет стандартные и непротиворечивые механизмы для генерации и обработки событий. Источник генерирует событие и посылает его одному или нескольким блокам прослушивания (listeners) событий. В этой схеме, блок прослушивания просто ждет поступления события. Получив событие, блок прослушивания обрабатывает его и затем возвращает управление. Преимущество указанного способа состоит в том, что логика приложения, обрабатывающего события, четко отделена от логики интерфейса пользователя, генерирующего эти события. Элемент интерфейса пользователя способен "делегировать" обработку события отдельной части кода.

В модели делегирования событий блоки прослушивания должны зарегистрироваться в источнике для того, чтобы принимать уведомление (сообщение) события. Это обеспечивает важное преимущество: уведомления посылаются только блокам прослушивания, которые хотят их принять.

События

В модели делегирования *событие* — это объект, который описывает изменение состояния источника. Он может быть сгенерирован как последовательность взаимодействий оператора ПК с элементами в графическом интерфейсе пользователя. Генерацию событий могут вызвать такие действия оператора, как нажатие кнопки, ввод символа через клавиатуру, выбор элемента в списке, щелчок мыши и множество других операций.

Могут происходить также события, которые непосредственно не вызываются взаимодействиями с интерфейсом пользователя. Событие, например, может сгенерировать таймер, завершающий работу, счетчик, превышающий предустановленное значение, программа в момент завершения операций или возникновения особых программных ситуаций, аппаратный отказ и т. д. Программист может сам определять события, которые будут обнаруживать и обрабатывать его приложение.

Источники событий

Источник — это объект, который генерирует событие. Генерация события происходит тогда, когда каким-то образом изменяется внутреннее состояние этого объекта. Источники могут генерировать несколько типов событий.

Чтобы блоки прослушивания могли принимать уведомление об определенном типе событий, источник должен *регистрировать* эти блоки. Каждый тип событий имеет собственный метод регистрации. Общая форма таких методов:

```
public void addTypeListener(TypeListener el)
```

Здесь Type — это имя события, а el — ссылка на блок прослушивания события. Например, метод, который регистрирует блок прослушивания события клавиатуры, называется addKeyListener(). Метод, регистрирующий блок прослушивания движения мыши, называется addMouseMotionListener().

Когда событие происходит, все зарегистрированные блоки прослушивания уведомляются (о происшедшем событии) и принимают копию объекта события. Это известно как *мультивещание* (multicasting) событий. Во всех случаях уведомления посылаются только блокам прослушивания, которые зарегистрировались для их приема.

Некоторые источники могут позволять регистрироваться только одному блоку прослушивания. Общая форма такого метода:

```
public void addTypedListener(TypeListener el) throws Ja-
va.util.TooManyListenersException
```

где Type – имя события, а el — ссылка на блок прослушивания события. При наступлении события зарегистрированный блок прослушивания уведомляется. Это известно как *унивещание* (unicasting) событий.

Источник должен также обеспечить метод, который позволяет блоку прослушивания не регистрировать заинтересованность в определенном типе событий. Общая форма такого метода:

```
public void removeTypeListener (TypeListener el)
```

Здесь Type — имя события, а el — ссылка на блок прослушивания события. Например, чтобы удалить блок прослушивания клавиатуры, следует вызвать метод removeKeyListener().

Методы, которые добавляют или удаляют блоки прослушивания, обеспечиваются генерирующим событие источником. Например, класс Component обеспечивает методы для добавления и удаления блоков прослушивания событий клавиатуры и мыши.

Блок прослушивания событий

Блок прослушивания — это объект, который получает уведомление, когда происходит событие. К нему предъявляются два главных требования. Во-первых, чтобы принимать уведомления относительно определенных типов событий, он должен быть *зарегистрирован* одним или несколькими источниками. Во-вторых, он должен *реализовать* методы для приема и обработки этих уведомлений.

Методы, которые принимают и обрабатывают события, определены в наборе интерфейсов, находящихся в пакете java.awt.event. Например, интерфейс MouseMotionListener определяет два метода для приема уведомлений о событиях перетаскивания или передвижения мыши. Любой объект может принимать и обрабатывать одно или оба этих события, если он обеспечивает реализацию этого интерфейса.

Классы событий

В основе механизма обработки событий находятся классы, которые представляют события.

В корне иерархии классов событий Java находится класс EventObject, который размещен в пакете java.util. Это — суперкласс для всех событий. Один из его конструкторов:

```
EventObject(Object src)
```

где src — объект, который генерирует это событие.

EventObject содержит два метода: getSource() и toStringt(). Метод getSource() возвращает источник события. Его общая форма:

```
Object getSource()
```

Метод toString() возвращает строчный эквивалент события.

Класс AWTEvent, определенный в пакете java.awt, является подклассом класса EventObject. Это суперкласс (прямо или косвенно) всех AWT-событий, используемых моделью делегирования событий. Для определения типа события можно использовать его метод getID(). Сигнатура этого метода:

```
int getID()
```

Итак:

□ Eventobject — суперкласс всех событий.

□ AWTEvent — суперкласс всех AWT-событий, которые обрабатываются моделью делегирования событий.

Пакет java.awt.event определяет несколько типов событий, которые генерируются различными элементами интерфейса пользователя. Табл. 12.1 перечисляет наиболее важные из этих классов событий и кратко

описывает, когда они генерируются. Наиболее часто используемые конструкторы и методы каждого класса рассматриваются в следующих разделах.

Таблица 12.1. Основные классы событий java.awt.event

Класс событий	Описание
ActionEvent	Генерируется, когда нажата кнопка, дважды щелкнут элемент списка или выбран пункт меню
AdjustmentEvent	Генерируется при манипуляциях с полосой прокрутки
ComponentEvent	Генерируется, когда компонент скрыт, перемещен, изменен в размере или становится видимым
ContainerEvent	Генерируется, когда компонент добавляется или удаляется из контейнера
FocusEvent	Генерируется, когда компонент получает или теряет фокус
InputEvent	Абстрактный суперкласс для всех классов событий ввода компонентов
ItemEvent	Генерируется, когда: • помечен флажок или элемент списка • сделан выбор элемента в списке выбора • выбран/отменен элемент меню с меткой
KeyEvent	Генерируется, когда получен ввод от клавиатуры
MouseEvent	Генерируется, когда объект перетащен мышью (dragged) или перемещен (moved), произошел щелчок (clicked), нажата (pressed) или отпущена (released) кнопка мыши; также генерируется, когда указатель мыши входит или выходит в/за границы компонента
TextEvent	Генерируется, когда изменено значение текстовой области или текстового поля
WindowEvent	Генерируется, когда окно активизировано, закрыто, дезактивировано, развернуто или свернуто в значок, открыто или организован выход (exit) из него

Класс ActionEvent

Событие ActionEvent генерируется, когда нажата кнопка, произошел двойной щелчок по элементу списка или выбран пункт меню. Класс ActionEvent определяет четыре целочисленные константы, которые можно использовать для идентификации события: alt_mask, ctrl_mask, meta_mask и shift_mask. Кроме того, существует целочисленная константа, action_performed, которую можно применять для идентификации action-события.

ActionEvent имеет два конструктора:

```
ActionEvent (Object src, int type, String cmd)
ActionEvent (Object src, int type, String cmd, int modifiers)
```

где src — ссылка на объект, который генерировал это событие; type — тип события; cmd — его командная строка; modifiers — указывает, какие клавиши-модификаторы (<Alt>, <Ctrl>, <META>, и/или <Shift>) были нажаты при генерации события.

Можно получить имя команды для вызова объекта ActionEvent, используя метод getActionCommand() с форматом:

```
String getActionCommand()
```

Например, когда кнопка нажата, генерируется action-событие, которое имеет имя команды, равное метке (надписи) на этой кнопке.

Метод getModifiers() возвращает значение, которое указывает, какие клавиши-модификаторы (<Alt>, <Ctrl>, <META>, и/или <Shift>) были нажаты при генерации события. Его формат:

```
int getModifiers()
```

Класс AdjustmentEvent

События AdjustmentEvent генерируются полосой прокрутки. Существует пять типов adjustment-событий. Класс AdjustmentEvent определяет целочисленные константы, которые можно использовать лля идентификации этих типов:

- BLOCK_DECREMENT. Пользователь щелкнул внутри полосы прокрутки, чтобы уменьшить ее значение.
- BLOCK_INCREMENT. Пользователь щелкнул внутри полосы прокрутки, чтобы увеличить ее значение.
- TRACK. Ползунок был перемещен.
- UNIT_DECREMENT. Кнопка-стрелка в начале полосы прокрутки была нажата, чтобы уменьшить ее значение.
- UNIT__INCREMENT. Кнопка-стрелка в конце полосы прокрутки была нажата, чтобы увеличить ее значение.

Кроме того, имеется целочисленная константа, ADJUSTMENT_VALUE_CHANGED, которая указывает на изменение установки полосы.

AdjustmentEvent имеет следующий конструктор:

```
AdjustmentEvent (Adjustable src, int id, int type, int data)
```

где src — ссылка на объект, который генерировал это событие; id — числовой идентификатор события установки, равный ADJUSTMENT_VALUE_CHANGED; type — определяет тип события; data — связанные с событием данные.

Метод getAdjustable() возвращает объект, который генерирован этим событием. Его формат:

```
Adjustable getAdjustable()
```

Тип события установки можно получить методом getAdjustmentType(). Он возвращает одну из констант, определенных в AdjustmentEvent. Общая форма этого метода:

```
int getAdjustmentType()
```

Значение установки полосы прокрутки можно получить методом getValue() с форматом:

```
int getValue()
```

Например, когда выполняются манипуляции с полосой прокрутки, этот метод возвращает значение, представляющее измененное значение полосы.

Класс ComponentEvent

События ComponentEvent генерируются, когда размер, позиция или видимость компонента изменяются. Существует четыре типа компонентных событий. Класс ComponentEvent определяет четыре целочисленные константы, которые можно использовать для идентификации этих типов:

- □ COMPONENT_HIDDEN. Компонент был скрыт.
- □ COMPONENT_MOVED. Компонент был перемещен.
- □ COMPONENT_RESIZED. Размер компонента был изменен.
- □ COMPONENT_SHOWN. Компонент стал видимым.

Класс ComponentEvent имеет такой конструктор:

```
ComponentEvent(Component src, int type)
```

где src — ссылка на объект, который сгенерировал данное событие; type — определяет тип события.
ComponentEvent — это суперкласс (прямой или косвенный) классов ContainerEvent, FocusEvent, KeyEvent, MouseEvent и WindowEvent.

Метод getComponent() возвращает компонент, который сгенерировал событие. Формат этого метода:

```
Component getComponent()
```

Класс ContainerEvent

События класса ContainerEvent генерируются при добавлении или удалении компонента из контейнера. Существует два типа container-

событий. Для их идентификации класс ContainerEvent определяет целочисленные константы: COMPONENT_ADDED и COMPONENT_REMOVED. Они указывают, что компонент был добавлен или удален из контейнера.

Класс ContainerEvent — это подкласс ComponentEvent с конструктором:

```
ContainerEvent(Component src, int type, Component comp)
```

где src — ссылка на контейнер, который сгенерировал это событие; type — тип события; comp — компонент, который был добавлен или удален из контейнера.

Можно получить ссылку на контейнер, который сгенерировал событие, используя метод getContainer():

```
Container getContainer()
```

Метод getChild() возвращает ссылку на компонент, который был добавлен или удален из контейнера. Его общая форма:

```
Component getChild()
```

Класс FocusEvent

Событие FocusEvent генерируется, когда компонент получает или теряет фокус ввода. Эти события идентифицируются целочисленными константами FOCUS_GAINED и FOCUS_LOST. FocusEvent — подкласс ComponentEvent и имеет конструкторы:

```
FocusEvent(Component src, int type)
FocusEvent(Component src, int type, boolean temporaryFlag)
```

где src — ссылка на компонент, который генерировал это событие; type — тип события. Параметр temporaryFlag устанавливается как true, если событие фокуса временное. Иначе — устанавливается как false.

Временное событие фокуса происходит в результате другой операции интерфейса пользователя. Например, предположим, что фокус находится в текстовом поле. Если пользователь перемещает мышь, чтобы откорректировать полосу прокрутки, фокус временно теряется.

Метод isTemporary() указывает, является ли это изменение фокуса временным. Его форма:

```
boolean isTemporary()
```

Метод возвращает true, если изменение временно. Иначе, он возвращает false.

Класс InputEvent

Абстрактный класс InputEvent есть подкласс ComponentEvent и суперкласс для событий ввода компонентов. Его подклассы— KeyEvent и MouseEvent. Класс InputEvent определяет следующие восемь целочисленных констант, которые можно использовать для получения информации относительно любых модификаторов, связанных с этим событием:

- ALT_MASK
- ALT_GRAPH_MASK
- BUTT0N1_MASK
- BUTTON2_MASK
- BUTTON3_MASK
- CTRL_MASK
- META_MASK
- SHIFT_MASK

Методы isAltDown(), isAltGraphDown(), isControlDown(), isMetaDown() и isShiftDownt() проверяют, были ли нажаты эти модификаторы в то время, когда событие было сгенерировано. Форматы этих методов:

```
boolean isAltDown()
boolean isAltGraphDown()
boolean isControlDown()
boolean isMetaDown()
boolean isShiftDown()
```

Метод getModifiers() возвращает значение, которое содержит все флажки модификаторов для этого события. Его сигнатура имеет вид:

```
int getModifiers()
```

Класс ItemEvent

События ItemEvent генерируются при установке/сбросе флажка или щелчке по элементу списка, или когда пользователь выполняет (или отменяет) выбор пункта меню с меткой. (Флажки и списковые панели описаны дальше.) Существует два типа item-событий, которые идентифицированы целочисленными константами:

- DESELECTED. Пользователь отменил выбор элемента.
- SELECTED. Пользователь выбрал элемент.

Кроме того, ItemEvent определяет целочисленную константу ITEM_STATE_CHANGED, которая обозначает изменение состояния.

Класс itemEvent имеет конструктор:

```
ItemEvent(ItemSelectable src, int type, Object entry, int state)
```

где src - ссылка на компонент, который генерировал это событие, например, им мог бы быть элемент типа "выбор" или "список"; type — определяет тип события; entry — передает конструктору определенный

элемент, который генерировал item-событие; state — текущее состояние этого элемента.

Метод getItem() можно использовать для получения ссылки на элемент, который генерировал событие. Его сигнатура:

```
Object getItem()
```

Метод getItemSelectable() можно использовать для получения ссылки на объект ItemSelectable, который сгенерировал событие. Его общая форма:

```
ItemSelectable getItemSelectable()
```

Примерами элементов интерфейса пользователя, которые реализует интерфейс ItemSelectable, являются (обычные) списки и списки с выбором.

Метод getStateChange() возвращает изменение состояния события (SELECTED или DESELECTED). Формат метода:

```
int getStateChange()
```

Класс KeyEvent

Событие KeyEvent генерируется, когда происходит ввод с клавиатуры. Имеются три типа key-событий, которые идентифицируются целочисленными константами: KEY_PRESSED, KEY_RELEASED и KEY_TYPED. Первые два события генерируются, когда любая клавиша нажимается или отпускается. Последнее событие происходит только при нажатии символьной клавиши. Напоминаем, что не все нажатия клавиш приводят к вводу символа. Например, нажатие клавиши <Shift> не генерирует символ.

Существуют и другие целочисленные константы, которые определены в классе KeyEvent. Например, VK_0 - VK_9 и VK_A - VK_Z определяют эквиваленты ASCII-цифр и букв. Вот некоторые из них:

- VK_ALT
- VK_CANCEL
- VK_CONTROL
- VK_DOWN
- VK_ENTER
- VK_ESCAPE
- VK_LEFT
- VK_PAGE_DOWN
- VK__PAGE_UP
- VK_RIGHT
- VK_SHIFT
- VK_UP

Константы VK определяют *коды виртуальных клавиш* (virtual key codes) и независимы от любых модификаторов, таких как <Ctrl>, <Shift> или <Alt>.

Класс KeyEvent является подклассом InputEvent и имеет два конструктора:

```
KeyEvent (Component src, int type, long when, int modifiers, int code)
KeyEvent (Component src, int type, long when, int modifiers, int code, char
ch)
```

где src — ссылка на компонент, который генерировал это событие; type — тип события; when — параметр, передающий конструктору системное время, когда была нажата клавиша; modifiers — параметр, указывающий, какие модификаторы были нажаты, когда данное клавишное событие произошло; code — параметр, передающий конструктору код виртуальной клавиши VK_UP, VK_A и т. д. Параметр ch передает эквивалент символа (если он существует). Если никакой допустимый символ не существует, то ch содержит CHAR_UNDEFINED. Для событий KEY_TYPED параметр code будет содержать VK_UDEFINED.

Класс KeyEvent определяет несколько методов, но обычно используются getKeyChar(), возвращающий символ, который был введен, и getKeyCode(), который возвращает код клавиши. Их общие формы:

```
char getKeyChar()
int getKeyCode()
```

Если никакой допустимый символ не доступен, то getKeyChar() возвращает CHAR_UNDEFINED. Когда происходит событие KEY_TYPED, getKeyCode() возвращает VK_UNDEFINED.

Класс MouseEvent

Существует семь типов mouse-событий. Класс MouseEvent определяет целочисленные константы, которые могут использоваться для их идентификации:

- □ MOUSE_CLICKED. Пользователь щелкнул кнопкой мыши.
- □ MOUSE_DRAGGED. Пользователь перетащил мышь.
- □ MOUSE_ENTERED. Указатель мыши веден в компонент.
- □ MOUSE_EXITED. Указатель мыши выведен из компонента.
- □ MOUSE_MOVED. Мышь передвинута.
- □ MOUSE_PRESSED. Кнопка мыши нажата.
- □ MOUSE_RELEASED. Кнопка мыши освобождена.

MouseEvent — это подкласс InputEvent. Он имеет конструктор:

```
MouseEvent(Component src, int type, long when, int modifiers, int x, int y,
int clicks, boolean triggersPopup)
```

где src — ссылка на компонент, который генерировал это событие; type — тип события; when — параметр, передающий конструктору системное время, когда клавиша была нажата; modifiers — параметр, указывающий, какие модификаторы были нажаты, когда произошло mouse-событие. Координаты мыши передаются параметрами x и y. Счет щелчков передается в clicks. Флажок triggersPopup указывает, приводит

ли это событие к появлению раскрывающегося меню на данной платформе.

Обычно используемые методы в этом классе — getX() и getY(). Они возвращают (x, y)-координаты мыши, когда событие произошло. Их форматы:

```
int getX()
int getY()
```

Альтернативно, чтобы получить координаты мыши, можно использовать метод getPoint() с форматом:

```
Point getPoint()
```

Он возвращает объект Point, который содержит (x, y)-координаты в его целых членах x и y. Метод translatePoint() изменяет положение события. Его формат:

```
void translatePoint (int x, int y)
```

Здесь аргументы x и y добавляются к координатам события.

Метод getClickCount() получает число щелчков мыши для этого события. Его сигнатура:

```
int getClickCount()
```

Метод isPopupTrigger() проверяет, приводит ли событие к появлению всплывающего меню на данной платформе. Его формат:

```
boolean isPopupTrigger()
```

Класс TextEvent

Экземпляры этого класса описывают события, генерируемые текстовыми полями и текстовыми областями, когда пользователь или программа вводят в них символы. В TextEvent определена целая константа TEXT_VALUE_CHANGED.

Один из конструкторов указанного класса:

```
TextEvent (Object src, int type)
```

где src — ссылка на компонент, который генерировал это событие; type — тип события.

Объект TextEvent не включает немедленно символы в текстовый компонент, который сгенерировал событие. Чтобы извлечь эту информацию, программа должна использовать другие методы. Данная операция отличается от других событийных объектов, рассмотренных в текущем разделе. По этой причине здесь не обсуждаются никакие методы класса TextEvent. Уведомление text-события нужно представлять себе как сигнал, извещающий блок прослушивания о том,

что он должен сам извлечь информацию из определенного текстового компонента.

Класс WindowEvent

Существует семь типов window-событий. Чтобы идентифицировать их, класс WindowEvent определяет следующие целые константы:

- □ WINDOW_ACTWATED. Окно активизировано.
- □ WINDOW_CLOSED. Окно закрыто.
- □ WINDOW_CLOSING. Пользователь потребовал, чтобы окно было закрыто.
- □ WINDOW_DEACTWATED. Окно деактивизировано.
- □ WINDOW_DEICONIFIED. Окно развернуто из пиктограммы.
- □ WINDOW_ICONIFIED. Окно свернуто в пиктограмму.
- □ WINDOW_OPENED. Окно открыто.

WindowEvent — подкласс ComponentEvent и имеет следующий конструктор:

```
WindowEvent (Window src, int type)
```

где src — ссылка на компонент, который генерировал это событие; type указывает тип события.

В этом классе чаще всего используется метод getWindow(). Он возвращает window-объект, который сгенерировал событие. Его общая форма:

```
Window getWindow()
```

1.2. Элементы-источники событий

Некоторые из компонентов интерфейса пользователя, которые могут генерировать события, описанные в предыдущем разделе, перечислены в табл. 12.2. В дополнение к этим элементам графического интерфейса пользователя, события могут генерировать другие компоненты, такие как апплеты. Например, от апплета принимаются key- и mouse-события. (Можно также строить свои собственные компоненты, которые генерируют события.) Пока мы будем обрабатывать только события мыши и клавиатуры, но затем будет рассмотрена обработка событий из источников, показанных в табл. 12.2.

Таблица 12.2. Примеры источников событий

Источник события	Описание
Кнопка (Button)	Генерирует action-события, когда нажимается кнопка

Флажок (checkbox)	Генерирует item-события, когда флажок устанавливается или сбрасывается
Список с выбором (choice)	Генерирует item-события, когда изменяется выбор элемента в списке с выбором
Список (List)	Генерирует action-события, когда на элементе списка выполнен двойной щелчок (мышью). Генерирует item-события, когда элемент выделяется или выделение снимается
Пункт меню (Menu Item)	Генерирует action-события, когда пункт меню выделен; генерирует события элемента, когда пункт меню с меткой выделен или выделение отменяется
Полоса прокрутки (Scrollbar)	Генерирует adjustment-события при манипуляциях с полосой прокрутки
Текстовые компоненты	Генерирует text-события, когда пользователь вводит символ
Окно (window)	Генерирует window-события, когда окно активизируется, закрывается, деактивизируется, сворачивается в пиктограмму, разворачивается из пиктограммы, открывается или выполняется выход из него (quit)

1.3. Интерфейсы прослушивания событий

Модель делегирования событий содержит две части: источники событий и блоки прослушивания событий. Блоки прослушивания событий создаются путем реализации одного или нескольких *интерфейсов прослушивания событий*, определяемых пакетом java.awt.event. Когда событие происходит, источник события вызывает соответствующий метод, определенный блоком прослушивания, и передает ему объект события в качестве параметра. Табл. 12.3 перечисляет используемые чаще всего интерфейсы прослушивания и приводит краткое описание методов, которые эти блоки прослушивания определяют.

Таблица 12.3. Интерфейсы прослушивания событий

Интерфейс	Описание
ActionListener	Определяет (один) метод для приема action-события
AdjustmentListener	Определяет (один) метод для приема adjustment-события
ComponentListener	Определяет четыре метода, распознающих события, связанные со скрытием, перемещением, изменением и показом компонента

ContainerListener	Определяет два метода, распознающих события добавления или удаления компонента из контейнера
FocusListener	Определяет два метода, распознающих события приобретения или потери компонентом фокуса клавиатуры
ItemListener	Определяет (один) метод, распознающий события изменения состояний элемента
KeyListener	Определяет три метода, распознающих события нажатия, отпускания и ввода символа клавиши
MouseListener	Определяет пять методов, распознающих события щелчка, входа в границы компонента, выхода из границ компонента, нажатия и отпускания клавиши мыши
MouseMotionListener	Определяет два метода, распознающих события перетаскивания или перемещения мыши
TextListener	Определяет (один) метод, распознающий события изменения текстового значения
WindowListener	Определяет семь методов, распознающих события активизации, деактивизации, открытия, закрытия, сворачивания/разворачивания (в значок) и выхода из окна

Следующие разделы рассматривают специфические методы, которые содержатся в каждом интерфейсе.

Интерфейс ActionListener

Этот интерфейс определяет метод actionPerformed(), который вызывается при наступлении action-события. Его общая форма:

```
void actionPerformed(ActionEvent ae)
```

Интерфейс AdjustmentListener

Этот интерфейс определяет метод adjustmentValueChanged(), который вызывается при наступлении adjustment-события. Его общая форма:

```
void adjustmentValueChanged(AdjustmentEvent ae)
```

Интерфейс ComponentListener

Этот интерфейс определяет четыре метода, которые вызываются, когда компонент изменяется в размерах, перемещается, отображается на экране или скрывается. Их общие формы:

```
void componentResized(ComponentEvent ce)
void componentMoved(ComponentEvent ce)
void componentShown()(ComponentEvent ce)
void componentHidden(ComponentEvent ce)
```

AWT обрабатывает события изменения размеров и перемещения. Методы componentResized() и componentMoved() обеспечены только для целей уведомления.

Интерфейс ContainerListener

Этот интерфейс содержит два метода. Когда компонент добавляется к контейнеру, вызывается метод componentAdded(). Когда компонент удаляется из контейнера, вызывается метод componentRemoved(). Их общие формы:

```
void componentAdded(ContainerEvent ce)
void componentRemoved(ContainerEvent ce)
```

Интерфейс FocusListener

Этот интерфейс определяет два метода. Когда компонент получает фокус клавиатуры, вызывается метод focusGained(). Когда компонент теряет фокус клавиатуры, вызывается метод focusLost(). Их общие формы:

```
void focusGained(FocusEvent fe)
void focusLost(FocusEvent fe)
```

Интерфейс ItemListener

Этот интерфейс определяет метод itemStateChanged(), который вызывается при изменении состояния элемента. Его общая форма:

```
void itemStateChanged( ItemEvent ie)
```

Интерфейс KeyListener

Этот интерфейс определяет три метода. Методы keyPressed() и keyReleased() вызываются, когда клавиша нажата и отпущена, соответственно. Метод keyTyped() вызывается при вводе символа.

Например, если пользователь нажимает и отпускает буквенную клавишу <A>, последовательно генерируется три события — "клавиша нажата", "символ введен" и "клавиша отпущена". Если пользователь нажимает и отпускает клавишу <Home>, последовательно генерируется два события — "клавиша нажата" и "клавиша отпущена".

Общие формы этих методов:

```
void keyPressed(KeyEvent ке)
```

```
void keyReleased (KeyEvent Are)
void keyTyped (KeyEvent ke)
```

Интерфейс MouseListener

Этот интерфейс определяет пять методов. Если кнопка мыши нажата и сразу же отпущена, вызывается метод mouseClicked(). Когда указатель мыши входит в границы компонента, вызывается метод mouseEntered(), а когда выходит — вызывается метод mouseExited(). Методы mousePressed() и mouseReleased() вызываются, когда кнопка мыши нажимается и отпускается, соответственно.

Общие формы этих методов:

```
void mouseClicked (MouseEvent me)
void mouseEntered (MouseEvent me)
void mouseExited (MouseEvent me)
void mousePressed (MouseEvent me)
void mouseReleased (MouseEvent me)
```

Интерфейс MouseMotionListener

Этот интерфейс определяет два метода. Метод mouseDragged() вызывается много раз, когда мышь перетаскивается (dragged). Метод mouseMoved() вызывается много раз, когда мышь перемещается (moved). Их общие формы:

```
void mouseDragged (MouseEvent me)
void mouseMoved (MouseEvent me)
```

Интерфейс TextListener

Этот интерфейс определяет метод textChanged(), который вызывается при изменении содержимого текстовой области или текстового поля. Его общая форма:

```
void textChanged (TextEvent te)
```

Интерфейс WindowListener

Этот интерфейс определяет семь методов. Методы windowActivated() и windowDeactivated() вызываются, когда окно активизируется или деактивируется, соответственно. Если окно сворачивается в пиктограмму, вызывается метод windowIconified(). Когда окно разворачивается из пиктограммы, вызывается метод windowDeIconified(). Когда окно открывается или закрывается, вызываются методы windowOpened() или windowClosed(), соответственно. Метод windowClosing() вызывается, когда обнаруживается, что окно закрыто. Общие формы этих методов:

```
void windowActivated(WindowEvent we)
void windowClosed (WindowEvent we)
void windowClosing (WindowEvent we)
void windowDeactivated (WindowEvent we)
void windowDeiconified (WindowEvent we)
void windowIconified (WindowEvent we)
void windowOpened (WindowEvent we)
```

1.4. Использование модели делегирования событий

Программирование апплетов, использующее модель делегирования событий, включает два шага:

1. Следует реализовать соответствующий интерфейс в блоке прослушивания, чтобы он принимал события желаемого типа.

2. Следует реализовать код для регистрации (или не регистрации, если это необходимо) блока прослушивания в качестве получателя уведомлений о событии.

Источник может генерировать несколько типов событий. Каждое событие должно быть зарегистрировано отдельно. Объект тоже может регистрироваться, чтобы принимать несколько типов событий, но он должен реализовать все интерфейсы, которые требуются для приема этих событий.

Рассмотрим примеры, демонстрирующие обработку событий мыши и клавиатуры.

Обработка событий мыши

Для обработки событий мыши (mouse-события) нужно реализовать интерфейсы MouseListener и MouseMotionListener. Следующий апплет представляет этот процесс. Он отображает текущие координаты мыши в окне состояния апплета. Каждый раз, когда кнопка нажимается, в месте расположения указателя отображается слово "Down". Каждый раз, когда кнопка отпускается, показывается слово "Up". Если произведен щелчок кнопкой мыши, в левом верхнем углу области показа апплета выводится сообщение "Mouse clicked".

Когда указатель мыши входит или выходит из окна апплета, в левом верхнем углу области апплета выводится сообщение "Mouse entered". При перетаскивании мыши показывается символ *, который отслеживает указатель мыши. Две переменные, mouseX и mouseY, хранят положение курсора мыши, когда происходят события нажатия, освобождения кнопки или перетаскивания мыши. Эти координаты затем используются методом paint(), чтобы отобразить вывод в той точке, где эти событие происходят.

Программа 90. Обработка событий мыши

```
// Файл MouseEvents.java
// Демонстрирует обработчики событий мыши.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="MouseEvents" width=300 height=100>
</applet>
*/
public class MouseEvents extends Applet
      implements MouseListener, MouseMotionListener {   // Шаг 1
   String msg = "";
   int mouseX = 0, mouseY = 0;        // Координаты мыши
   public void init() {
      addMouseListener(this);         // Шаг 2. Регистрация (себя) как блока
                                      // прослушивания mouse-событий
      addMouseMotionListener(this);   // Регистрация (себя) как блока
   }                                  // прослушивания MouseMotion-событий
   // Обработка щелчка мыши.             Шаг 3
   public void mouseClicked(MouseEvent me) {   // Реализуем метод
      mouseX = 0;                  // mouseClicked интерфейса MouseListener
      mouseY = 10;
      msg = "Mouse clicked.";
      repaint();
   }
   public void paint(Graphics g) {
      g.drawString(msg, mouseX, mouseY);
   }

   // Обработка ввода мыши в область окна.
   public void mouseEntered(MouseEvent me) {
   // Сохранить координаты
      mouseX = 0;
      mouseY = 10;
      msg = "Mouse entered.";
      repaint();
   }
   // Обработка вывода мыши из области окна.
   public void mouseExited(MouseEvent me) {
   // Сохранить координаты
      mouseX = 0;
      mouseY = 10;
      msg = "Mouse exited.";
      repaint();
   }
   // Обработка нажатия кнопки.
   public void mousePressed(MouseEvent me) {
   // Сохранить координаты
      mouseX = me.getX();
      mouseY = me.getY();
      msg = "Down";
      repaint();
   }
```

```
    // Обработка освобождения кнопки.
    public void mouseReleased(MouseEvent me) {
    // Сохранить координаты
       mouseX = me.getX();
       mouseY = me.getY();
       msg = "Up";
       repaint();
    }
    // Обработка перетаскивания мыши.
    public void mouseDragged(MouseEvent me) {
    // Сохранить координаты
       mouseX = me.getX();
       mouseY = me.getY();
       msg = "*";
       showStatus("Dragging mouse at " + mouseX + ",  " + mouseY);
       repaint();
    }
    // Обработка перемещения мыши.
    public void mouseMoved(MouseEvent me) { // Отобразить состояние
       showStatus ("Moving mouse at " + me.getX() + ",  " + me.getY());
    }
}
```

Программу можно запустить непосредственно из среды Eclipse командой **Run, Run** или комбинацией клавиш **Ctrl+F11**. Окно апплета показано на рис. 1

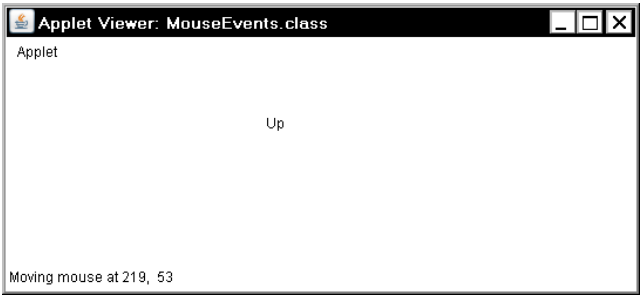

Рис. 1.Окно апплета для демонстрации событий от мыши

Рассмотрим этот пример подробнее. Класс MouseEvents расширяет Applet и реализует интерфейсы MouseMotionListener и MouseListener. Оба интерфейса содержат методы, которые принимают и обрабатывают различные типы событий мыши. Обратите внимание, что апплет является как источником, так и блоком прослушивания для этих событий (это общая ситуация для любых апплетов), т.к. Component, который обеспечивает методы addMouseListener() и addMouseMotion-Listener(), является суперклассом для Applet.

Внутри init() апплет регистрирует сам себя как блок прослушивания событий мыши. Это организуется с помощью методов

addMouseListener() и addMouseMotionListener(), которые являются членами класса Component:

```
synchronized void addMouseListener(MouseListener ml)
synchronized void addMouseMotionListener(MouseMotionListener mml)
```

где ml — ссылка на объект, принимающий событие мыши; mml - ссылка на объект, принимающий события движения мыши. В этой программе для обеих ссылок используется один и тот же объект.

Затем апплет реализует все методы, определенные интерфейсами MouseMotionListener и MouseListener. Они являются обработчиками для различных событий мыши. Каждый метод обрабатывает свое событие и затем возвращает управление.

1.5. Обработка событий клавиатуры

Чтобы обрабатывать события клавиатуры нужно реализовать интерфейс KeyListener.

Когда клавиша нажимается, генерируется событие KEY_PRESSED. Это приводит к запросу обработчика события keyPressed(). Когда клавиша отпускается, генерируется событие KEY_RELEASED и выполняется обработчик keyReleased().

Если нажатием клавиши сгенерирован символ, то посылается уведомление о событии KEY_TYPED и вызывается обработчик keyTyped(). Таким образом, каждый раз, когда пользователь нажимает клавишу, генерируется по крайней мере два, а часто — три события. Если в программе нужны только вводимые символы, то можно игнорировать информацию, которую посылает нажатие клавиш. Однако если ваша программа должна обработать специальные клавиши, типа клавиш управления курсором или функциональных клавиш, то она должна наблюдать за ними через обработчик keyPressed().

Прежде чем программа сможет обработать события клавиатуры: она должна запросить фокус ввода. Для этого вызывается requestFocus(), который определен в классе Component. Если этого не сделаеть, программа не будет принимать никаких событий клавиатуры.

Следующий пример демонстрирует ввод с клавиатуры. Он отображает в окно апплета символы клавиш и показывает состояние нажатия/освобождения каждой клавиши в окне состояния.

Программа 91. Обработка событий клавиатуры

```
// Файл SimpleKey.java
// Демонстрирует обработчики клавишных событий.
import java.awt.*;
import java.awt.event.*;
```

```
import java.applet.*;
/*
   <applet code="SimpleKey" width-300 height=100> </applet>
*/
public class SimpleKey extends Applet implements KeyListener {
   String msg = "";
   int X = 10, Y = 20;          // Координаты вывода
   public void init() {
      addKeyListener(this);     // Регистрация (себя) как блока
                                // прослушивания key-событий
      requestFocus();           // Запрос фокуса ввода
   }
   public void keyPressed(KeyEvent ke) {
      showStatus("Key Down");  // Вывод сообщения в окне состояния браузера
   }
   public void keyReleased(KeyEvent ke) {
      showStatus("Key Up");
   }
   public void keyTyped(KeyEvent ke) {
      msg += ke.getKeyChar();
      repaint ();               // Вызывает paint()
   }
   // Показывает нажатую клавишу в позиции указателя мыши.
   public void paint(Graphics g) {    //
      g.drawString(msg, X, Y);
   }
}
```

Вывод примера представлен на рис. 2.

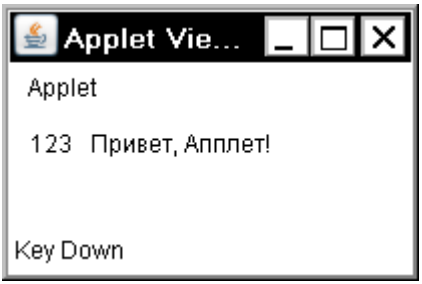

Рис. 2.Окно апплета SimpleKey

Если нужно обработать специальные клавиши типа клавиш перемещения курсора или функциональных клавиш, следует откликнуться на них в обработчике keyPressed(). Они недоступны через keyTyped(). Чтобы идентифицировать данные клавиши, используйте код их виртуальной клавиши. Например, следующий апплет выводит имя нескольких специальных клавиш:

Программа 92. Функциональные и управляющие клавиши

```
// Файл KeyEvents.java
// Демонстрирует некоторые коды виртуальных клавиш.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="KeyEvents" width = 300 height = 100> </applet>
*/
public class KeyEvents extends Applet implements KeyListener {
   String msg = "";
   int X = 10, Y = 20;          // Координаты вывода
   public void init() {
      addKeyListener(this);     // Регистрация блока прослушивания
      requestFocus();           // Запрос фокуса ввода
   }
   public void keyPressed(KeyEvent ke) {
      showStatus("Key Down");
      int key = ke.getKeyCode();  // Код, связанный с клавишей
      switch(key) {
         case KeyEvent.VK_F1:
            msg += "<F1>";
         break;
         case KeyEvent.VK_F2:
            msg += "<F2>";
         break;
         case KeyEvent.VK_F3:
            msg += "<F3>";
         break;
         case KeyEvent.VK_PAGE_DOWN:
            msg += "<PgDn>";
         break;
         case KeyEvent.VK_PAGE_UP:
            msg += "<PgUp>";
         break;
         case KeyEvent.VK_LEFT:
            msg += "<Left Arrow>";
         break;
         case KeyEvent.VK_RIGHT:
            msg += "<Right Arrow>";
         break;
      }
      repaint();
   }
   public void keyReleased(KeyEvent ke) {
      showStatus("Key Up");
   }
   public void keyTyped(KeyEvent ke) {
      msg += ke.getKeyChar();
      repaint();
   }
   // Показывает нажатую клавишу в позиции указателя мыши.
```

```
    public void paint(Graphics g) {
        g.drawString(msg, X, Y);
    }
}
```

Вывод примера представлен на рис.3.

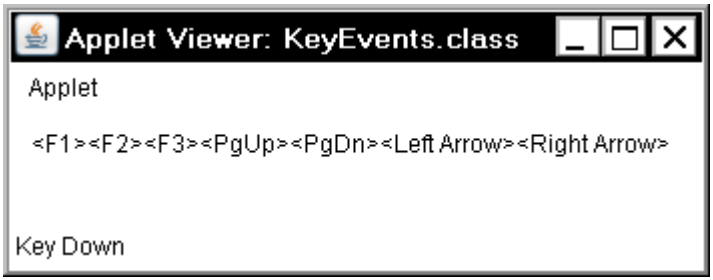

Рис. 3.Нажатие функциональных и управляющих клавиш

Процедуры, показанные в предыдущих примерах с событиями клавиатуры и мыши, могут быть обобщены на любой тип обработки событий, включая события, сгенерированные элементами управления.

1.6. Классы-адаптеры

Java обеспечивает специальное средство, называемое *классом адаптера* (adapter class), которое может упростить в некоторых ситуациях создание обработчиков событий. Класс адаптера (или просто "адаптер") обеспечивает пустую реализацию всех методов в интерфейсе прослушивания событий. Класс полезен, когда нужно принимать и обрабатывать только некоторые из событий, предоставляемые конкретным интерфейсом прослушивания. Для этого определяется новый класс, действующий как блок прослушивания событий, расширяющий один из имеющихся (в пакете java.awt.event) адаптеров и реализующий только те события, которые требуется обрабатывать.

Например, класс MouseMotionAdapter имеет два метода mouseDragged() и mouseMoved(). Сигнатуры этих пустых методов точно такие же, как в интерфейсе MouseMotionListener. Если нужны только события перетаскивания мыши, то можно просто расширить адаптер MouseMotionAdapter и реализовать метод mouseDragged(). События же перемещения мыши будет обрабатывать пустая реализация метода mouseMoved().

Табл. 12.4 перечисляет различные классы-адаптеры, которые определены в пакете java.awt.event, и указывает те интерфейсы прослушивания, которые каждый из них реализует.

Таблица 12.4. Интерфейсы прослушивание событий, реализованные с помощью классов адаптеров

Класс-адаптер	Интерфейс прослушивания событий
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
FocusAdapter	FocusListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
WindowAdapter	WindowListener

Следующий пример демонстрирует использование класса-адаптера. Он отображает сообщение в строке состояния программы просмотра апплета или браузера, когда выполняется перетаскивание мыши или щелчок. Однако все другие события мыши по умолчанию игнорируются. Программа содержит три класса. Класс AdapterDemo расширяет Applet. Его метод init() создает экземпляр MyMouseAdapter и регистрирует этот объект для приема уведомлений от mouse-событий. Он также создает экземпляр MyMouseMotionAdapter и регистрирует этот объект для приема уведомлений событий MouseMotion (движения мыши). В качестве аргумента оба конструктора получают ссылку на данный апплет.

Класс MyMouseAdapter реализует метод mouseClicked(). Другие события мыши "молча" игнорируются кодом, унаследованным от класса MouseAdapter.

Класс MyMouseMotionAdapter реализует метод mouseDragged(). Другое событие движения мыши игнорируется кодом, унаследованным от класса MouseMotionAdapter.

Обратите внимание, что оба класса прослушивания событий хранят ссылку на данный апплет. С помощью ссылки this эта информация передается в оба конструктора и используется позже для вызова метода showStatus().

Программа 93. Классы-адаптеры

```
// Файл AdapterDemo.java
// Демонстрирует адаптер.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code "AdapterDemo" width=300 height l00> </applet>
```

```
*/
public class AdapterDemo extends Applet {
   public void init() {
      addMouseListener(new MyMouseAdapter(this) );
      addMouseMotionListener(new MyMouseMotionAdapter(this));
   }
}
class MyMouseAdapter extends MouseAdapter {
   AdapterDemo aD;
   public MyMouseAdapter(AdapterDemo _aD) {
      this.aD = _aD;
   }
   // Обработка щелчка мыши.
   public void mouseClicked(MouseEvent me) {
      aD.showStatus("Mouse clicked");
   }
}
class MyMouseMotionAdapter extends MouseMotionAdapter {
   AdapterDemo aD;
   public MyMouseMotionAdapter(AdapterDemo _aD) {
      this.aD = _aD;
   }
   // Обработка перетаскивания мыши.
   public void mouseDragged(MouseEvent me) {
      aD.showStatus("Mouse dragged");
   }
}
```

Реакция на перетаскивание мыши и на щелчек мыши приведены на рис. 4, 5

Рис. 4. Окно апплета с сообщением о перетаскивании мыши

Рис. 5. Окно апплета с сообщением о щелчке мыши

Отказ от необходимости реализовывать все методы, определенные интерфейсами MouseMotionListener и MouseListener значительно сокращает объем программы, освобождая код от загромождения пустыми методами.

1.7. Внутренние классы

Напомним, что *внутренний класс* — это класс, определенный внутри другого класса, или даже внутри выражения. Этот раздел иллюстрирует, как можно использовать внутренние классы, чтобы упростить код применяющий адаптеры событий.

Чтобы понять выгоду, обеспечиваемую внутренними классами, рассмотрим апплет, показанный в следующем листинге. Он *не использует* внутренний класс. Его цель состоит в том, чтобы отобразить строку "Mouse Pressed" в строке состояния программы просмотра апплета или браузера, когда нажимается клавиша мыши. В этой программе имеются два класса верхнего уровня. Класс MousePressedDemo расширяет класс Applet, а MyMouseAdapter расширяет MouseAdapter. Метод init() класса MousePressedDemo создает экземпляр (объект) класса MyMouseAdapter и передает его (как аргумент) методу addMouseListener().

Обратите внимание, что в качестве аргумента конструктора MyMouseAdapter указана *ссылка* на данный апплет. Она сохраняется в переменной экземпляра для более позднего использования методом mousePressed(). Когда нажимается кнопка мыши, через эту ссылку вызывается метод апплета showStatus(). Другими словами, showStatus() вызывается относительно ссылки на апплет, сохраненной адаптером MyMouseAdapter.

Программа 94. Программа без внутренних классов

```
// Файл MousePressedDemo.java
// Этот апплет не использует внутренний класс.
import java.applet.*;
import java.awt.event.*;
/*
<applet code="MousePressedDemo" width=200 height=100> </applet>
*/
public class MousePressedDemo extends Applet {
   public void init() {
      addMouseListener(new MyMouseAdapter(this));
   }
}

class MyMouseAdapter extends MouseAdapter {
   MousePressedDemo mPD;    // mPD - апплет MousePressedDemo
   public MyMouseAdapter(MousePressedDemo m) {
      mPD = m;
   }
   public void mousePressed(MouseEvent me) {
      mPD.showStatus("Mouse Pressed.");
   }
}
```

Следующий листинг демонстрирует, как предшествующая программа может быть улучшена с использованием внутреннего класса. Здесь InnerClassDemo — класс верхнего уровня, который расширяет класс Applet. Класс MyMouseAdapter — внутренний класс, который расширяет MouseAdapter. Поскольку MyMouseAdapter определен в области видимости InnerClassDemo, он имеет доступ ко всем переменным и методам в пределах этого класса. Поэтому метод mousePressed() может прямо вызвать метод showStatus(). Больше не требуется делать это через сохраненную ссылку на апплет. Таким образом, нет необходимости передавать конструктору MyMouseAdapter() ссылку на вызывающий объект.

Программа 95. Использование внутреннего класса

```
// Файл InnerClassDemo.java
// Демонстрация внутреннего класса.
import java.applet.*;
import java.awt.event.*;
/*
<applet code = "InnerCiassDemo" width = 200 height = 100> </applet>
*/
public class InnerClassDemo extends Applet {
   public void init() {
      addMouseListener(new MyMouseAdapter());
   }

   class MyMouseAdapter extends MouseAdapter {
      public void mousePressed(MouseEvent me) {
         showStatus("Mouse Pressed");
      }
      public void mouseReleased(MouseEvent me) {
         showStatus("");
      }
   }
}
```

Анонимные внутренние классы

Анонимный внутренний класс — это класс, которому не назначено имя. Этот раздел иллюстрирует, как анонимный внутренний класс может облегчать написание обработчиков событий. Рассмотрим апплет, показанный в следующем листинге. Как и прежде, его цель состоит в том, чтобы отобразить строку "Mouse Pressed" в строке состояния программы просмотра апплета или браузера, когда нажимается кнопка мыши.

Программа 96. Анонимный класс

```
// Файл AnonymousInnerClassDemo.java
```

```
// Демонстрация анонимного внутреннего класса.
import java.applet.*;
import java.awt.event.*;
/*
<applet code = "AnonymousInnerClassDemo" width = 200 height = 100> </applet>
*/
public class AnonymousInnerClassDemo extends Applet {
   public void init() {
      addMouseListener(new MouseAdapter() {
            public void mousePressed(MouseEvent me) {
               showStatus("Mouse Pressed");
            }
            public void mouseReleased(MouseEvent me) {
               showStatus("");
            }
         }
      );
   }
}
```

В этой программе представлен один класс верхнего уровня — AnonymousInnerClassDemo. Метод init() вызывает addMouseListener(). Его аргумент является выражением, которое определяет и строит экземпляр анонимного внутреннего класса. Давайте тщательно проанализируем это выражение.

Синтаксис new MouseAdapter(){...} указывает компилятору, что код между фигурными скобками определяет анонимный внутренний класс. Кроме того, этот класс расширяет класс MouseAdapter. Данный новый класс не именован, но автоматически создается экземпляр этого класса, когда данное выражение выполняется.

Поскольку этот анонимный внутренний класс определен в области видимости AnonymousInnerClassDemo, он имеет доступ ко всем переменным и методам в пределах области видимости этого класса. Поэтому он может прямо вызывать метод showStatus().

Как только что было показано, как именованный, так и анонимный внутренние классы решают некоторые раздражающие проблемы простым и довольно эффективным способом. Мы видим, что они позволяют создавать более результативный код и экономят труд программиста (уменьшая объем исходного кода).

Задачи11-12. Обработка событий в апплетах

11. Перепишите программу 91 ввода с клавиатуры так, чтобы использовался класс KeyAdapter.

12. Перепишите программу 92 ввода с клавиатуры так, чтобы использовался класс KeyAdapter.