События в Java

Лекция 6

Цели

- Знакомство с моделью событий
- Описание структуры событий AWT
- Рассмотрение основ управления событиями
- Работа с несколькими слушателями

События

- **Событие (event)** – это объект, описывающий изменение состояния источника события.
- Взаимодействие с элементами GUI приводит к генерации событий.
- Объект, в котором произошло событие, называется **источником** (source) события.
- События генерируются, когда пользователь щёлкает кнопкой мыши, нажимает или отпускает клавишу, перемещает компоненты, меняет размеры окна и др.

События

- Все события в **AWT** классифицированы.
- При возникновении события автоматически создается объект соответствующего событию класса.
- Этот объект не производит никаких действий, он только хранит все сведения о событии.
- Каждый тип событий имеет свой метод для регистрации:

public void **add<ТипСобытия>Listener** (TypeListener e)

ТипСобытия – имя события,

Параметр e – объектная ссылка на слушателя событий,
add<ТипСобытия>Listener () –метод, регистрирующий событие.

Методы

- Все компоненты AWT и Swing включают методы **addXXXListener()** и **removeXXXListener()**, так что для каждого компонента может быть добавлен и удален подходящий тип слушателя.

- Нужно заметить, что "XXX" в каждом случае также представляет аргумент метода, например: **addMyListener(MyListener m).**

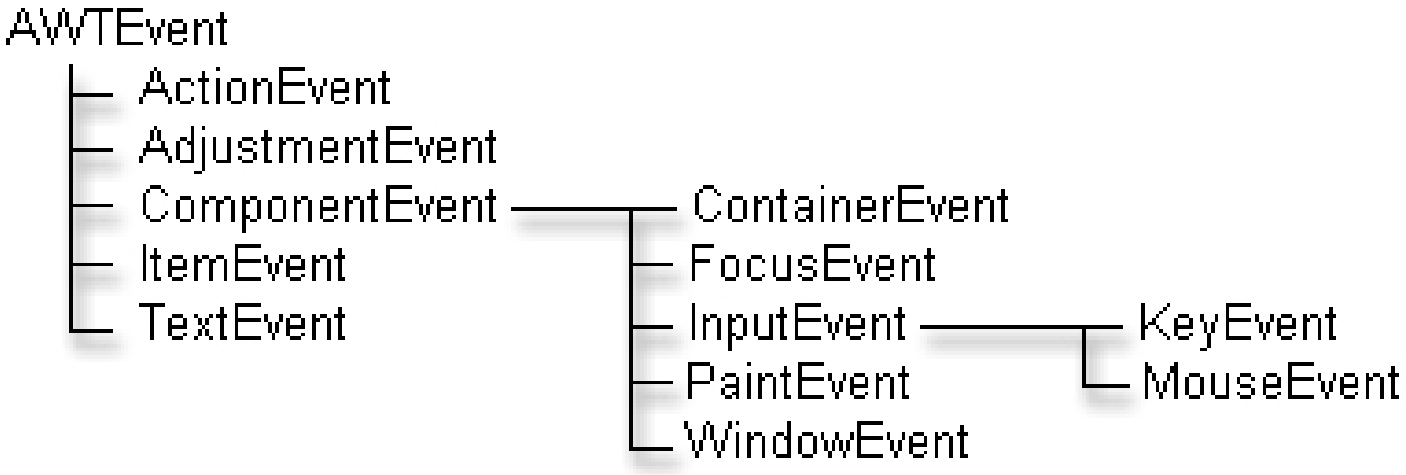
Обработка событий с помощью компонентов

- Модель делегирования событий – **Event Delegation Model** – используется для обработки событий.
- Эта модель позволяет программе регистрировать обработчики, называемые ‘**listeners**’ («слушатели»), для тех объектов, чьи события должны быть перехвачены.
- При возникновении события обработчики вызываются автоматически.
- Действие, которое должно быть совершено после наступления события, выполняется соответствующим listener’ом.

События

- События типа ComponentEvent - FocusEvent, KeyEvent, MouseEvent возникают во всех компонентах.

- А события типа ContainerEvent — только в контейнерах: Container, Dialog, FileDialog, Frame, Panel, ScrollPane, Window.



Типы событий

- События типа **ActionEvent** проявляются в компонентах Button, List, TextField.
- События типа **WindowEvent** возникают ТОЛЬКО В окнах: Frame, Dialog, FileDialog, Window.
- События типа **TextEvent** генерируются только в компонентах TextComponent, TextArea, TextField.
- События типа **ItemEvent** возникают только в компонентах Checkbox, Choice, List.
- События типа **AdjustmentEvent** возникают только в компоненте Scrollbar.

События

- **Источник событий** – это объект, регистрирующий одного или нескольких слушателей.
- Источник событий посылает оповещение всем зарегистрированным слушателям о случившемся событии.
- Слушатели событий используют объект События для получения дополнительной информации и определения действий по реакции на событие.

Обработка событий с помощью компонентов

- Слушатели событий (**Event listeners**) реализуются в Java в виде интерфейсов.
- Шаги, которые необходимо выполнить при использовании модели Event Delegation Model:
  - Связать класс с соответствующим интерфейсом **listener**.
  - Определить все компоненты, которые генерируют события.
  - Определить все события, которые должны быть обработаны.
  - Реализовать методы слушателя (listener) и написать код обработки события в этих методах.

События

- Методы обработки событий описаны в интерфейсах - **слушателях событий** (listener).
- Для каждого типа событий есть свой интерфейс.
- Имена интерфейсов составляются из имени события и слова **Listener**, например, ActionListener, MouseListener.
- Методы интерфейса "слушают", что происходит в потенциальном источнике события.
- При возникновении события эти методы автоматически выполняются, получая в качестве аргумента объект-событие и используя при обработке сведения о событии, которое содержится в этом объекте.
- Чтобы задать обработку события определенного типа, надо реализовать соответствующий интерфейс.
- Классы, реализующие такой интерфейс, классы-обработчики (handlers) события, называются ***слушателями*** (listeners): они "слушают", что происходит в объекте, чтобы отследить возникновение события и обработать его.

Типы событий AWT

Событие	Интерфейс слушателя	Описание	Компоненты, поддерживающие это событие
ActionEvent	ActionListener	Происходит при возникновении события "действие", используется для обработки событий меню, кнопок и т.п.	Button, List, TextField, MenuItem. JButton, JList, JTextField, JMenuItem и наследованные от них, включая JCheckBoxMenuItem, JMenu и JpopupMenu.
AdjustmentEvent	AdjustmentListener	Происходит при возникновении регулируемых событий	Scrollbar, JScrollbar и все, что вы создаете, реализуя Adjustable interface.
ComponentEvent	ComponentListener	Происходит при перемещении, изменении размеров, изменении видимости компонента, также служит корневым классом для ContainerEvent, FocusEvent, и др.	Компоненты GUI.
ContainerEvent	ContainerListener	Происходит при добавлении компонента в контейнер или при удалении из контейнера.	Container и наследованные от него, включая Dialog, FileDialog, Frame, Panel, ScrollPane, Window, JPanel, JApplet, JScrollPane, JDialog, JFileDialog и JFrame.

Типы событий AWT

FocusEvent	FocusListener	Происходит при возникновении событий получения или потери фокуса компонентом.	Компоненты GUI.
ItemEvent	ItemListener	Происходит при возникновении события выделения элемента.	CheckBox, ListItem, Choice, List. JCheckBox, JCheckBoxMenuItem, JComboBox, JList и все, что реализует ItemSelectable interface.
KeyEvent	KeyListener	Происходит при возникновении событий от клавиатуры.	Компоненты GUI при действии с клавиатурой.
MouseEvent	MouseListener (для кликов и перемещений) MouseMotionListener (для перемещений)	Происходит при возникновении событий от мыши.	Компоненты GUI при работе с мышью.

Типы событий AWT

TextEvent	TextListener	Происходит при изменении значения текстовых компонентов, таких, как TextArea, TextField.	TextField, TextArea. Все, что унаследовано от JTextComponent, включая JTextArea и JTextField.
WindowEvent	WindowListener	Происходит при изменении состоянии окна.	Window и унаследованные от него, включая Frame, Dialog, FileDialog, Window, JDialog, FileDialog и JFrame.

Интерфейсы слушателя

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

Интерфейсы слушателя

Интерфейс слушателя	Методы интерфейса
MouseListener	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent)
MouseMotionListener	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
WindowListener	windowOpened(WindowEvent) windowClosing(WindowEvent) windowClosed(WindowEvent) windowActivated(WindowEvent) windowDeactivated(WindowEvent) windowIconified(WindowEvent) windowDeiconified(WindowEvent)
ItemListener	itemStateChanged(ItemEvent)

Событие ActionEvent

- Это простое событие означает, что надо выполнить какое-то действие. При этом неважно, что вызвало событие: щелчок мыши, нажатие клавиши или что-то другое.
- В классе **ActionEvent** есть два полезных метода:
  - **getActionCommand ()** возвращает в виде строки String надпись на кнопке Button , точнее, то, выбранный пункт списка List , или что-то другое, зависящее от компонента;
  - **getModifiers()** возвращает код клавиш <Alt>, <Ctrl>, <Meta> или <Shift> , если какая-нибудь одна или несколько из них были нажаты, в виде числа типа int ; узнать, какие именно клавиши были нажаты, можно сравнением со статическими константами этого класса ALT_MASK , CTRL_MASK, META_MASK, SHIFT_MASK.

Событие MouseEvent

Событие MouseEvent возникает в компоненте по любой из семи причин:

- нажатие кнопки мыши — идентификатор MOUSE_PRESSED;
- отпускание кнопки мыши — идентификатор MOUSE_RELEASED;
- щелчок кнопкой мыши — идентификатор MOUSE_CLICKED (нажатие и отпускание не различаются);
- перемещение мыши — идентификатор MOUSE_MOVED;
- перемещение мыши с нажатой кнопкой — идентификатор MOUSE_DRAGGED;
- появление курсора мыши в компоненте — идентификатор MOUSE_ENTERED;
- выход курсора мыши из компонента — идентификатор MOUSE_EXITED.

Событие MouseEvent

Для их обработки есть семь методов в двух интерфейсах:

```
public interface MouseListener extends EventListener{
    public void mouseClicked(MouseEvent e);
    public void mousePressed(MouseEvent e) ;
    public void mouseReleased(MouseEvent e);
    public void mouseEntered(MouseEvent e);
    public void mouseExited(MouseEvent e);
}
public interface MouseMotionListener extends EventListener{
    public void mouseDragged(MouseEvent e);
    public void mouseMoved(MouseEvent e);
}
```

Событие MouseEvent

- Эти методы могут получить от аргумента **e** координаты курсора мыши в системе координат компонента методами e.getX(), e.getY(), или одним методом e.getPoint(), возвращающим экземпляр класса Point.

- Узнать, какая кнопка была нажата, можно с помощью метода e.getModifiers() класса inputEvent сравнением со следующими статическими константами класса InputEvent:
  - BUTTON1_MASK — нажата первая кнопка, обычно левая;
  - BUTTON2_MASK — нажата вторая кнопка, обычно средняя, или одновременно нажаты обе кнопки на двухкнопочной мыши;
  - BUTTON3_MASK — нажата третья кнопка, обычно правая.

Обработка действий клавиатуры

- Обрабатываются эти события тремя методами, описанными в интерфейсе:

```
public interface KeyListener extends EventListener{
    public void keyTyped(KeyEvent e);
    public void keyPressed(KeyEvent e);
    public void keyReleased(KeyEvent e);
}
```

Обработка действий клавиатуры

- Событие KeyEvent происходит в компоненте по любой из трех причин:
  - нажата клавиша — идентификатор KEY_PRESSED;
  - отпущена клавиша — идентификатор KEY_RELEASED;
  - введен символ — идентификатор KEY_TYPED.

- Последнее событие возникает из-за того, что некоторые символы вводятся нажатием нескольких клавиш, например, заглавные буквы вводятся комбинацией клавиш <Shift>+<буква>.

- Нажатие функциональных клавиш, например <F1>, не вызывает событие KEY_TYPED.

Событие TextEvent

- Событие **TextEvent** происходит только по одной причине — изменению текста — и отмечается идентификатором TEXT_VALUE_CHANGED.
- Соответствующий интерфейс имеет только один метод:

```
public interface TextListener extends EventListener{
    public void textValueChanged(TextEvent e) ;
    }
```

- От аргумента **e** этого метода можно получить ссылку на объект-источник события методом **getSource()**, унаследованным от класса EventObject, например, так:

```
TextComponent tc = (TextComponent)e.getSource();
String s = tc.getText() ;
// Дальнейшая обработка
```

Обработка действий с окном

- Событие **WindowEvent** может произойти по семи причинам:
  - окно открылось — идентификатор WINDOW_OPENED;
  - окно закрылось — идентификатор WINDOW_CLOSED;
  - попытка закрытия окна — WINDOW_CLOSING;
  - окно получило фокус — WINDOW_ACTIVATED;
  - окно потеряло фокус — WINDOW_DEACTIVATED;
  - окно свернулось в ярлык — WINDOW_ICONIFIED;
  - окно развернулось — WINDOW_DEICONIFIED.

Обработка действий с окном

Соответствующий интерфейс содержит семь методов:

```
public interface WindowListener extends EventListener
   {
      public void windowOpened(WindowEvent e);
      public void windowClosing(WindowEvent e);
      public void windowClosed(WindowEvent e);
      public void windowIconified(WindowEvent e);
      public void windowDeiconified(WindowEvent e);
      public void windowActivated(WindowEvent e);
      public void windowDeactivated(WindowEvent e);
   }
```

- Аргумент **e** этих методов дает ссылку типа **window** на окно-источник методом e.getWindow().
- Чаще всего эти события используются для перерисовки окна методом repaint() при изменении его размеров и для остановки приложения при закрытии окна.

Событие ComponentEvent

- Данное событие происходит в компоненте по четырем причинам:
  - компонент перемещается — идентификатор COMPONENT_MOVED;
  - компонент меняет размер — идентификатор COMPONENT_RESIZED;
  - компонент убран с экрана — идентификатор COMPONENT_HIDDEN;
  - компонент появился на экране — идентификатор COMPONENT_SHOWN.

- Соответствующий интерфейс содержит описания четырех методов:

```
public interface ComponentListener extends EventListener{
        public void componentResized(ComponentEvent e);
        public void componentMoved(ComponentEvent e);
        public void componentShown(ComponentEvent e);
        public void componentHidden(ComponentEvent e);
}
```

- Аргумент **e** методов этого интерфейса предоставляет ссылку на компонент-источник события методом e.getComponent().

Событие FocusEvent

- Событие возникает в компоненте, когда он получает фокус ввода — идентификатор FOCUS_GAINED, ИЛИ теряет фокус — идентификатор FOCUS_LOST.

- Соответствующий интерфейс:

```
public interface FocusListener extends EventListener{
    public void focusGained(FocusEvent e) ;
    public void focusLost(FocusEvent e) ;
}
```

- Обычно при потере фокуса компонент перечерчивается бледным цветом, для этого применяется метод brighter () класса Color, при получении фокуса становится ярче, что достигается применением метода darker(). Это приходится делать самостоятельно при создании своего компонента.

Событие ItemEvent

- Это событие возникает при выборе или отказе от выбора элемента в списке List, choice или флажка checkbox и отмечается идентификатором ITEM_STATE_CHANGED.

- Соответствующий интерфейс очень прост:

```
public interface ItemListener extends EventListener{
    void itemStateChanged(ItemEvent e);
}
```

- Аргумент e предоставляет ссылку на источник методом e.getItemSelectable(), ссылку на выбранный пункт методом e.getItem() в виде Object.

Алгоритм обработки события

- Каждый компонент поддерживает только определенные типы событий. И как только вы узнаете, какие события поддерживает данный компонент, вам необходимо:
  - Взять имя класса события и удалить слово "Event". К остатку прибавить слово "Listener". Это интерфейс слушателя, который вы должны реализовать в вашем внутреннем классе.
  - Реализовать вышеупомянутый интерфейс и написать методы для событий, который вы хотите отслеживать. Например, вы можете следить за движением мыши, тогда вы пишите код метода mouseMoved() из интерфейса MouseMotionListener.
  - Создать объект класса слушателя и зарегистрировать его в компоненте с помощью метода, произведенного добавлением "add" к имени вашего слушателя. Например: addMouseMotionListener().

Пример

```
class MyClass implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
}
```

Классы-адаптеры

- Классы-адаптеры представляют собой пустую реализацию интерфейсов-слушателей, имеющих более одного метода.
- Их имена составляются из имени события и слова Adapter.
- Вместо того, чтобы реализовать интерфейс, можно расширять эти классы.

Классы-адаптеры

Интерфейс слушателя	Класс-адаптер	Методы интерфейса
ComponentListener	ComponentAdapter	componentHidden(ComponentEvent) componentShown(ComponentEvent) componentMoved(ComponentEvent) componentResized(ComponentEvent)
ContainerListener	ContainerAdapter	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)
FocusListener	FocusAdapter	focusGained(FocusEvent) focusLost(FocusEvent)
KeyListener	KeyAdapter	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)
MouseListener	MouseAdapter	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent)
MouseMotionListener	MouseMotionAdapter	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
WindowListener	WindowAdapter	windowOpened(WindowEvent) windowClosing(WindowEvent) windowClosed(WindowEvent) windowActivated(WindowEvent) windowDeactivated(WindowEvent) windowIconified(WindowEvent) windowDeiconified(WindowEvent)

Классы-адаптеры

- Например, для действий с мышью есть два класса-адаптера:

```
public abstract class MouseAdapter implements MouseListener{
    public void mouseClicked(MouseEvent e){}
    public void mousePressed(MouseEvent e){}
    public void mouseReleased(MouseEvent e){}
    public void mouseEntered(MouseEvent e){}
    public void mouseExited(MouseEvent e){}
    }

public abstract class MouseMotionAdapter implements
   MouseMotionListener{
    public void mouseDragged(MouseEvent e){}
    public void mouseMoved(MouseEvent e){}
    }
```