Лабораторная работа №6

Обработка событий в AWT и Swing

Рассмотренные в предыдущей лабораторной работе средства построения графических интерфейсов не позволяли определить реакции на действия пользователя. В настоящей лабораторной работе рассматриваются некоторые компоненты Swing, которые позволяют реагировать на действия пользователя, и способы, позволяющие определить действия пользователя и описать реакцию на них.

1. События

Событие (event) возникает при воздействии на компонент с помощью мыши, при вводе с клавиатуры, при перемещении окна или изменении его размеров. Объект, в котором произошло событие, называется ***источником*** (source) события.

Все события классифицированы. При возникновении события Java автоматически создает объект соответствующего событию класса. Этот объект не производит никаких действий, он только хранит все сведения о событии.

Иерархия классов-событий AWT представлена на рисунке 1. Некоторые компоненты пакета Swing могут генерировать другие объекты событий, которые непосредственно расширяют класс EventObject.

Во главе иерархии стоит класс EventObject из пакета java.util – непосредственное расширение класса Object. Его расширяет абстрактный класс AWTEvent из пакета java.awt – прародитель всех классов, описывающих события библиотеки AWT.

События ComponentEvent, FocusEvent, KeyEvent, MouseEvent возникают во всех компонентах. События типа ContainerEvent – в контейнерах класса Container. События типа WindowEvent возникают к окнах класса Window и в его наследниках. События типа TextEvent генерируются только в компонента, предназначенных для работы с текстом, таких как TextComponent, TextArea, TextField. События типа ActionEvent возникают в компонентах Button, List, TextField, JcomboBox, JtextField, кнопках класса AbstractButton и его наследниках. События типа ItemEvent возникают в компонентах Checkbox, Jcheckbox, Choice, JcomboBox и в нопках класса AbstractButton и его наследниках. События типа AdjustmentEvent возникают только в полосах прокрутки Scrollbar и JscrollBar.

Определить в каком объекте произошло событие (какой объект является источником события), можно с помощью метода getSource() класса EventObject. Этот метод возвращает ссылку на объект класса Object.

В каждом из классов-событий определен метод paramString(),

возвращающий содержимое объекта-события в виде строки. В каждом классе-событии есть собственные методы, предоставляющие сведения о событии.

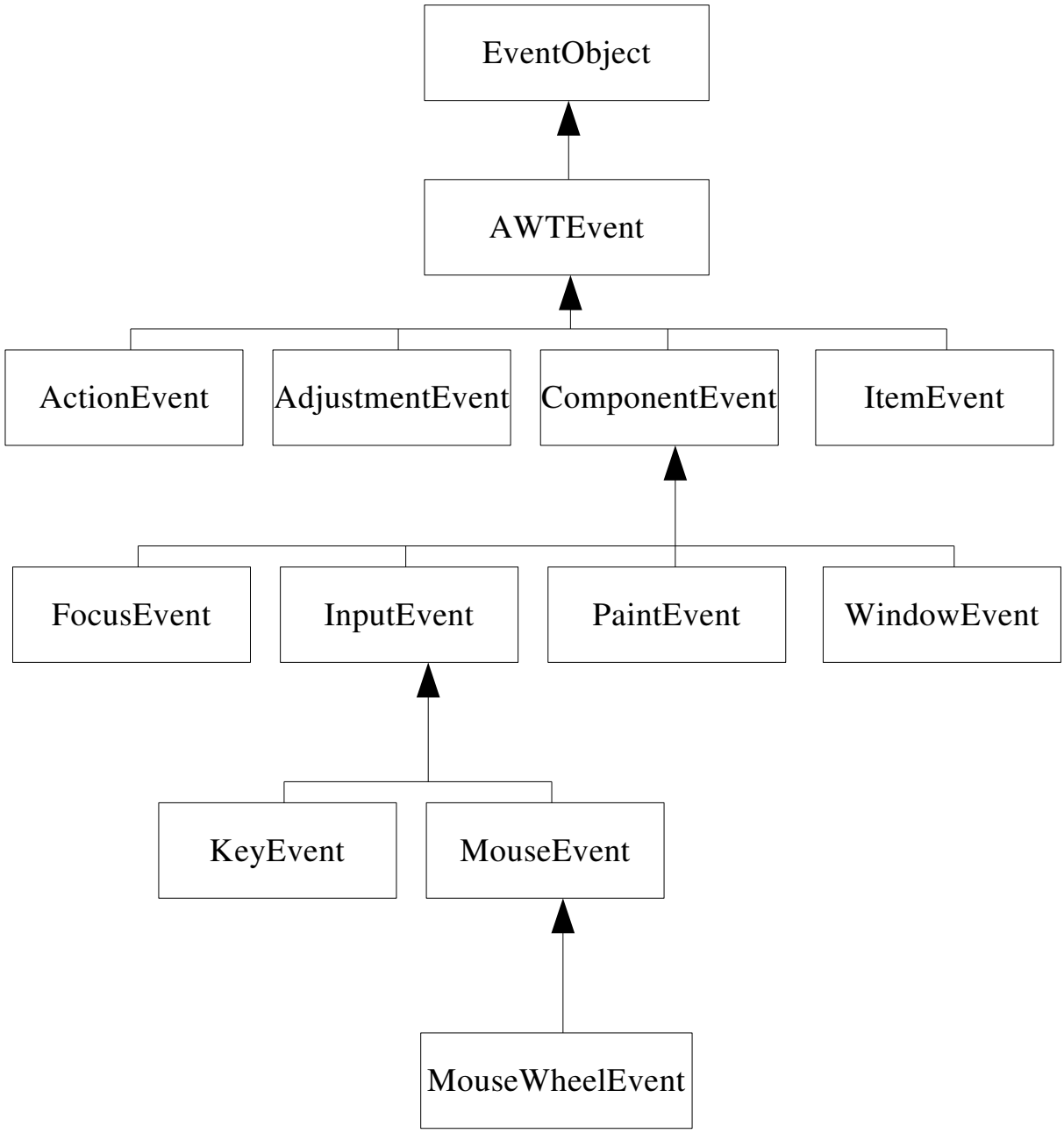

Рисунок 1. Иерархия событий библиотеки AWT

Событие нельзя обработать произвольно написанным методом. У каждого события есть свои методы, к которым обращается Java при его возникновении. Они описаны в интерфейсах-***слушателях*** (listener). Для каждого класса событий (за исключением InputEvent) существует собственный интерфейс. Имена интерфейсов образуются из имени класса события и слова Listener (слушатель), например ActionListener. Методы интерфейса слушают, что производит в потенциальном источнике событий. При возникновении событий они автоматически выполняются, получая в качестве аргумента объект-событие

и используя при обработке сведения о событии, содержащиеся в этом объекте.

Чтобы задать обработку событий определенного типа, надо реализовать соответствующий интерфейс. Классы, реализующие такой интерфейс, классы-***обработчики*** (handlers) события называются слушателями, поскольку они постоянно отслеживают («слушают»), что происходит в объекте, чтобы отследить возникновение события и обработать его.

В AWT события разделены на ***низкоуровневые*** и ***семантические***. Семантические события описывают действия пользователя, например щелчок на кнопке. Низкоуровневые события обеспечивают возможность таких действий. Если пользователь щелкнул по кнопке, значит, он нажал кнопку мыши, возможно, переместил курсор по экрану и отпустил кнопку мыши (курсор при этом находился в пределах кнопки).

Из классов пакета java.awt.event, описывающих семантические события, наиболее часто используются следующие.

ActionEvent – возникает в следующих случаях: щелчок по кнопке, выбор пункта меню, выбор пункта в списке, нажатие клавиши «Enter» при работе с полем редактирования.

AdjustmentEvent – возникает при перемещении ползунка на полосе прокрутки.

ItemEvent – возникает при выборе положения переключателя опций или пункта в списке.

Из низкоуровневых событий наиболее часто обрабатываются следующие.

KeyEvent – возникает при нажатии или отпускании клавиши.

MouseEvent – возникает при нажатии или отпускании кнопки мыши, перемещении курсора мыши, перетаскивании (перемещении курсора при нажатой клавище).

MouseWheelEvent – возникает при вращении колесика мыши.

FocusEvent – возникает при получении или потери фокуса ввода.

WindowEvent – возникает при изменении состояния окна.

В таблице 1 приведены некоторые классы событий из пакетов java.awt.event и javax.swing.event и их более подробные описания.

Таблица 1.некоторые классы событий из пакетов java.awt.event и javax.swing.event

Класс события	Описание	Интерфейс, который должен реализовывать обработчик
java.awt.event		
ActionEvent	Генерируется при выполнении действий с интерфейсным элементом, например по щелчку по кнопке	ActionListener
AdjustmentEvent	Генерируется при выполнении действий с полосой прокрутки	AdjustmentListener
ItemEvent	Генерируется при выборе или отмене выбора элемента, например, по щелчку на флажке опций	ItemListener
FocusEvent	Генерируется когда компонент получает или теряет фокус ввода	FocusListener
KeyEvent	Генерируется при вводе данных с клавиатуры	KeyListener
MouseEvent	Генерируется при перемещении или перетаскивании мыши, нажатии или отпускании клавиши, а также при перемещении курсора мыши за компонент или выводе курсора за пределы компонента	MouseListener
MouseWheelEvent	Генерируется при движении колесика мыши	MouseWheelListener
WindowEvent	Генерируется при активации, деактивации, закрытии окна, сворачивании его в пиктограмму и разворачиванию из пиктограммы	WindowListener
javax.swing.event		
AncestorEvent	Генерируется при добавлении, перемещении или удалении предка компонента	AncestorListener
CaretEvent	Генерируется при изменении позиции курсора в текстовом компоненте	CaretListener
ChangeEvent	Генерируется при изменении состояния компонента	ChangeListener
HyperlinkEvent	Генерируется при действиях, связанных с гиперссылкой	HyperlinkListener
ListDataEvent	Генерируется при изменении содержимого списка	ListDataListener
MenuEvent	Генерируется при выборе или отмене выбора	MenuListener

Класс события	Описание	Интерфейс, который должен реализовывать обработчик
	пунктов меню	
TreeModelEvent	Генерируется при изменении модели дерева	TreeModelListener

2. Обработка событий

Механизм обработки событий в AWT можно описать следующим образом.

1. Обработчик события представляет собой экземпляр класса, реализующего специальный интерфейс.

2. Источник события – это объект, который может зарегистрировать обработчики и посылать им объекты событий.

3. При наступлении события источник посылает объекты события всем зарегистрированным обработчикам.

4. Обработчики используют информацию, инкапсулированную в объекте события и некоторым образом реагируют на событие.

То есть, источник генерирует событие, которое передается одному или нескольким обработчикам. В рамках этой схемы обработчики лишь ожидают возникновения события. При возникновении события они обрабатывают его и возвращают управление. Преимущество такого подхода заключается в том, что логика обработки событий отделена от логики пользовательского интерфейса, генерирующего эти события. Элемент интерфейса делегирует обработку события отдельному фрагменту кода. В такой модели (она называется моделью делегирования) обработчик, чтобы получать оповещения о событиях, должен быть ***зарегистрирован*** в источнике.

Регистрация осуществляется путем вызова метода add*Tun*Listener, принадлежащего источнику. Заголовок этого метода имеет вид, подобный приведенному ниже.

public void add*Tun*Listener(*Tun*Listener el),

где *Tun* – это имя события, а параметр *el* представляет собой ссылку на обработчик события. Например, метод, регистрирующий обработчик события клавиатуры, называется addKeyListener(). Для обработки событий, связанных с перемещением мыши, используется addMouseMotionListener(). О возникшем событии оповещаются все обработчики.

Источник события представляет метод, предоставляющий возможность

отменить регистрацию обработчика событий определенного типа. Этот метод имеет следующий заголовок:

public void remove*Tun*Listener(*Tun*Listener el),

где *Tun* – имя события, а параметр el – ссылка на обработчик. Например, для того, чтобы удалить некоторый обработчик событий клавиатуры, надо вызвать метод removeKeyListener().

Напомним, что методы, добавляющие или обрабатывающие события, принадлежат объектам-источникам событий. Например, класс JButton (кнопка) содержит методы для регистрации и отмены регистрации обработчика ActionListener.

Обработчик – это объект, оповещаемый о возникновении события. К нему предъявляются два основных требования. Во-первых, он должен получать оповещение о конкретном типе событий. Он должен быть зарегистрирован в одном или нескольких источниках. Во-вторых, он должен реализовывать метод, предназначенный для обработки события.

Методы, позволяющие получать и обрабатывать события, определены в интерфейсах, содержащихся в пакетах java.awt.event, javax.swing.event и java.beans. Например, в интерфейсе ActionListener объявлен метод, который вызывается тогда, когда пользователь щелкнет на кнопке, или выполняет другое действие, затрагивающее компонент. Это событие может быть обработано любым объектом, при условии, что он реализует интерфейс ActionListener и зарегистрирован как получатель события.

В таблице 2 приводится описание объектов и интерфейсов, участвующих в обработке событий.

Таблица 1.Объекты и интерфейсы, участвующие в обработке событий

Интерфейс	**Методы**	**Параметр и методы доступа**	**Источник события**
ActionListener	actionPerformed	ActionEvent • getActionCommand • getModifiers	AbstractButton JcomboBox JtextField Timer
AdjustmentListener	adjustmentValueChanged	AdjustmentEvent • getAdjustable • getAdjustmentType • getValue	JScrollbar
ItemListener	itemStateChange	ItemEvent • getItem • getItemSelectable • getStateChange	AbstractButton JComboBox
FocusListener	focusGained	FocusEvent	Component

Интерфейс	Методы	Параметр и методы доступа	Источник события
	focusLost	• isTemporary	
KeyListener	keyPressed keyReleased keyTyped	KeyEvent • getKeyChar • getKeyCode • getKeyModifiersText • getKeyText • isActionKey	Component
MouseListener	mousePressed mouseReleased mouseEntered mouseExited mouseClicked	MouseEvent • getClickCount • getX • getY • getPoint • translatePoint	
MouseMotionListener	mouseDragged mouseMoved	MouseEvent	Component
MouseWheelListener	mouseWheelMoved	MouseWheelEvent • getWheelRotation • getScrollAmount	Component
WindowListener	windowClosing windowOpened windowIconified windowDeiconified windowClosed windowActivated windowDefctivated	WindowEvent • getWindow	Window
WindowFocusListener	windowGainedFocus windowLostFocus	WindowEvent • getOppositeWindow	Window
WindowStateListener	windowStateEvent	WindowEvent • getOldState • getNewState	Window

3. Классы адаптеров

Несмотря на то, что большинство интерфейсов обработчиков событий реализовать не сложно, Java предоставляет набор классов адаптеров, в которых уже определены методы, объявленные в интерфейсах обработчиков. Классы адаптеры удобны тогда, когда необходимо использовать лишь некоторые из событий, объявленных в интерфейсе. Новый класс, выполняющий функции обработчика, можно создать как подкласс класса адаптера, переопределив в нем только интересующие методы. Поскольку при использовании адаптера отпадает

необходимость определять все методы, объявленные в интерфейсе обработчика, уменьшается объем кода и время его написания.

Для некоторых интерфейсов обработчиков адаптеры отсутствуют. Например, для ActionEvent адаптера не существует, поскольку в интерфейсе, который должен реализовывать обработчик этого события, объявлен только один метод.

Адаптеры создаются только для интерфейсов, в которых объявлено более одного метода. Например, в интерфейсе MouseMotionListener описаны два метода: mouseDragged() и mouseMoved(). Реализации этих методов, не выполняющие никаких действий, определены в классе MouseMotionAdapter. Если при разработке интересуют только события, связанные с перемещением, то достаточно создать обработчик как подкласс MouseMotionAdapter и реализовать в нем метод mouseMoved(). Пустой (то есть не выполняющий никаких действий) метод mouseDragged() будет унаследован от MouseMotionAdapter.

Ниже перечислено несколько классов адаптеров (большинство из них определено в пакете java.awt.event).

Таблица 3. Классы адаптеров

Класс адаптера	**Реализуемый интерфейс**
FocusAdapter	FocusListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
MouseInputAdapter	MouseInputListener
WindowAdapter	WindowListener

4. Кнопки и обработка событий, связанных с кнопками

Кнопки являются одними из самых простых управляющих элементов Swing. Кнопка представляет собой экземпляр класса JButton. Этот класс является потомком абстрактного класса AbstractButton, в котором определены функции, общие для всех кнопок. На кнопке может отображаться текст, изображение или информация обеих типов.

Класс JButton содержит три конструктора.

JButton(String label) – создает кнопку с надписью, передаваемую строкой label.

JButton(Icon icon) – создает кнопку с изображением.

JButton(String label, Icon icon) – создает кнопку с надписью и изображением.

По щелчку на кнопке генерируется событие ActionEvent. Для регистрации и отключения обработчиков этого события JButton предоставляет методы

void addActionListener(ActionListener al)

void removeActionListener(ActionListener al).

В интерфейсе ActionListener определен только один метод actionPerformed(). Определение этого метода выглядит следующим образом:

void actionPerformed(ActionEvent ae). Этот метод вызывается по щелчку на кнопке, то есть он занимается обработкой событий, связанных с действиями пользователя с кнопкой. Реализуя метод actionPerformed() необходимо позаботиться о том, чтобы он быстро выполнял свою задачу и возвращал управление.

Объект ActionEvent, передаваемый методу actionPerformed() позволяет получить важную информацию, связанную с событием данного типа.

Ниже приводится пример программы, демонстрирующей обработку событий, связанных с нажатием кнопки, а на рисунке 2 показано окно, созданное этой программой (была нажата первая кнопка).

```
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;


public class ButtonDemo implements ActionListener {
     JLabel jlab;
     ButtonDemo(){
          // Создаем фрейм
          JFrame frame = new JFrame("Обработка событий кнопок");

          // Устанавливаем размеры созданного фрейма
          frame.setSize(220, 90);

          // Устанавливаем менеджер компоновки
          // для созданного фрейма
          frame.setLayout(new FlowLayout());

          // Устанавливаем поведение при закрытии окна
          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

          // Создаем две кнопки
          JButton button1 = new JButton("Первая");
```

```
        JButton button2 = new Jbutton("Вторая");

        // К каждой кнопке добаляем слушателя события
        // ActionEvent. В нашем случае обрабатывать
        // событие будет наш класс ButtonDemo. Он может
        // делать это, поскольку реализует интерфейс
        // ActionListener
        button1.addActionListener(this);
        button2.addActionListener(this);

        // Добавляем кнопки к фрейму
        frame.getContentPane().add(button1);
        frame.getContentPane().add(button2);

        // Создаем метку и добавляем ее к фрейму
        jlab = new JLabel("Нажмите кнопку");
        frame.getContentPane().add(jlab);

        // Делаем фрейм видимым
        frame.setVisible(true);
    }

    public void actionPerformed(ActionEvent e) {
        // Этот метод будет вызываться, когда в любой
        // из двух кнопок возникнет событие класса ActionEvent
        if (e.getActionCommand().equals("Первая"))
            jlab.setText("Нажата первая кнопка");
        else
            //
            jlab.setText("Нажата вторая кнопка");

    }


    public static void main(String[] args) {
        ButtonDemo bd = new ButtonDemo();

    }

}
```

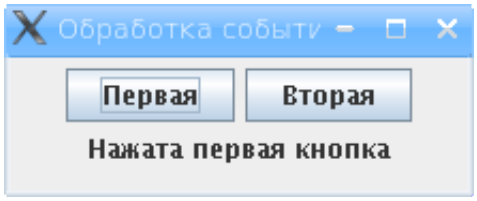

Рисунок 2. Окно программы после того, как была нажата первая кнопка

5. Поля ввода и обработка событий, связанных с ними

Поля ввода используются очень часто. Они позволяют пользователю вводить строку текста.

Поля ввода реализуются как объекты класса JTextField, являющимся потомком абстрактного класса JTextComponent.

В классе JTextField определены следующие конструкторы.

JTextField(int cols) – создает пустое поле ввода шириной в cols символов.

JTextField(String text, int cols) -создает текстовое поле шириной в cols символов, в котором отображается строка text.

После окончания ввода в текстовое поле пользователь нажимает клавишу «Enter», в результате чего генерируется событие ActionEvent. Класс JTextField предоставляет методы addActionListener() и removeActionListener(). Для обработки событий необходимо реализовать метод actionPerformed(). Обработка этого класса событий поля редактирования осуществляется аналогично обработке событий кнопки.

Чтобы получить строку, отображаемую в поле редактирования, необходимо обратиться к экземпляру класса JTextField и вызвать у него метод getText().

Задать текст позволяет метод setText(String text).

Ниже приводится текст программы, которая позволяет вводить два числа и складывает их при нажатии соответствующей кнопки. Результат выполнения программы приведен на рисунке 3.

```
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;


public class TextFieldDemo implements ActionListener{
     JTextField field1, field2;
     JLabel itog;
     TextFieldDemo(){

          // Создаем фрейм
          JFrame frame = new JFrame("Обработка событий кнопок");
```

```
        // Устанавливаем размеры созданного фрейма
        frame.setSize(300, 120);

        // Устанавливаем менеджер компоновки
        // для созданного фрейма
        frame.setLayout(new GridLayout(4,1));

        // Устанавливаем поведение при закрытии окна
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Создаем два поля для ввода
        field1 = new JTextField(10);
        field2 = new JTextField(10);

        // Создаем метку, в которую будем выводить
        // результат умножения
        itog = new JLabel("Введите числа для умножения");

        // Создаем кнопку, по нажатии на которую
        // будет производиться умножение
        JButton go = new Jbutton("Умножить");

        // добавляем к кнопке обработчик событий
        go.addActionListener(this);

        // добавляем к фрейму два поля для ввода,
        // кнопку и метку
        frame.getContentPane().add(field1);
        frame.getContentPane().add(field2);
        frame.getContentPane().add(go);
        frame.getContentPane().add(itog);

        // делаем фрейм видимым
        frame.setVisible(true);
}
        // Реализуем метод для обработки нажатия кнопки.
        // Метод описан в интерфейсе ActionListener,
        // который реализуется нашим классом TextFieldDemo
        public void actionPerformed(ActionEvent e) {
             // Получаем строку, которая введена в первое
             // текстовое поле
             String f1 = field1.getText();

             // Получаем строку, которая введена во второе
             // текстовое поле
             String f2 = field2.getText();

             // Преобразуем полученные из текстовых полей строки
             // в целые числа. Для этого используем статический
             // метод parseInt класса Integer
             int num1 = Integer.parseInt(f1);
             int num2 = Integer.parseInt(f2);
```

```
                // выполняем умножение
                int num = num1 * num2;

                // Помещаем в метку результат. Для этого сначала
                // создаем объект класса-оболочки Integer,
                // передавая в его конструктор величину типа int.
                // Затем с помощью метода toString преобразуем
                // величину в строку
                itog.setText(new Integer(num).toString());
        }
        public static void main(String[] args) {
                TextFieldDemo tfd = new TextFieldDemo();
        }

}
```

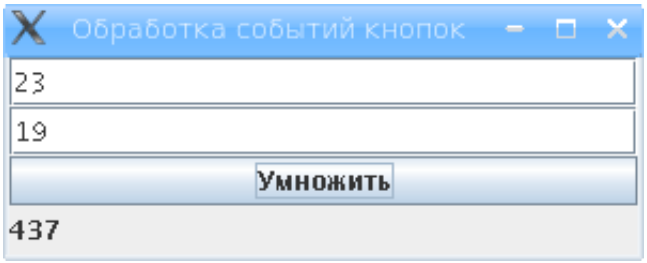

Рисунок 3. Результат выполнения программы

Ниже приведена измененная версия этой программы, которая позволяет обрабатывать события в поле ввода, связанные с изменением фокуса.

```
import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.FocusEvent;
import java.awt.event.FocusListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;


public class TextFieldDemo implements ActionListener,
FocusListener{
        JTextField field1, field2;
        JLabel itog, status;
        String stroka = new String("Фокус там, где ");
```

```
TextFieldDemo(){
     // Создаем фрейм
     JFrame frame = new JFrame("Обработка событий кнопок");
     // Устанавливаем размеры созданного фрейма
     frame.setSize(300, 120);
     // Устанавливаем менеджер компоновки
     // для созданного фрейма
     frame.setLayout(new GridLayout(5,1));
     // Устанавливаем поведение при закрытии окна
     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     // Создаем два поля для ввода
     field1 = new JTextField("15",10);
     field1.addFocusListener(this);
     field2 = new JTextField("21",10);
     field2.addFocusListener(this);
     // Создаем метку, в которую будем выводить
     // результат умножения
     itog = new JLabel("Введите числа для умножения");
     // Создаем метку, в которую будем выводить
     // компонент, который имеет фокус
     status = new JLabel("Отображает, где фокус ввода");
     status.setForeground(new Color(200,0,0));
     status.setHorizontalAlignment(JLabel.CENTER);
     status.addFocusListener(this);
     // Создаем кнопку, по нажатии на которую
     // будет производиться умножение
     JButton go = new JButton("Умножить");
     // добавляем к кнопке обработчик событий
     go.addActionListener(this);
     // добавляем к фрейму два поля для ввода,
     // кнопку и метку
     frame.getContentPane().add(field1);
     frame.getContentPane().add(field2);
     frame.getContentPane().add(go);
     frame.getContentPane().add(itog);
     frame.getContentPane().add(status);
     // делаем фрейм видимым
     frame.setVisible(true);
}
     // Реализуем метод для обработки нажатия кнопки.
     // Метод описан в интерфейсе ActionListener,
     // который реализуется нашим классом TextFieldDemo
     public void actionPerformed(ActionEvent e) {
          // Получаем строку, которая введена в первое
          // текстовое поле
          String f1 = field1.getText();
          // Получаем строку, которая введена во второе
          // текстовое поле
          String f2 = field2.getText();
          // Преобразуем полученные из текстовых полей строки
          // в целые числа. Для этого используем статический
          // метод parseInt класса Integer
```

```
                int num1 = Integer.parseInt(f1);
                int num2 = Integer.parseInt(f2);
                // выполняем умножение
                int num = num1 * num2;
                // Помещаем в метку результат. Для этого сначала
                // создаем объект класса-оболочки Integer,
                // передавая
                // в его конструктор величину типа int.
                // Затем с помощью метода toString преобразуем
                // величину в строку
                itog.setText(new Integer(num).toString());
        }
            public void focusGained(FocusEvent e) {
                status.setText(stroka+((JTextField)
e.getComponent()).getText());


            }
            public void focusLost(FocusEvent e) {


            }
        public static void main(String[] args) {
            TextFieldDemo tfd = new TextFieldDemo();
        }


}
```

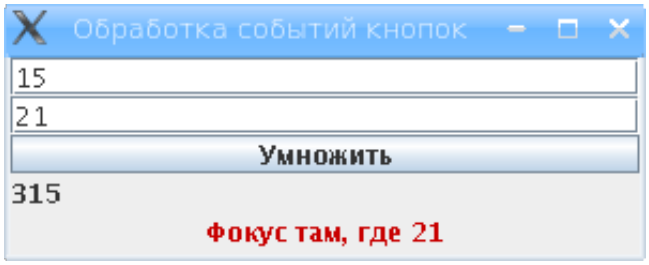

Рисунок 4. Результат выполнения программы, обрабатывающей изменения фокуса ввода