

## **Практическая работа №16 Обработка событий мыши и клавиатуры в программах на Джава с графическим интерфейсом пользователя**

**Цель:** научиться обрабатывать различные события мыши и клавиатуры для разных компонентов.

### **Теоретические сведения**

#### **Механизм обработки событий библиотеки Swing**

В контексте графического интерфейса пользователя наблюдаемыми объектами являются элементы управления: кнопки, флажки, меню и т.д. Они могут сообщить своим наблюдателям об определенных событиях, как элементарных (наведение мышкой, нажатие клавиши на клавиатуре), так и о высокоуровневых (изменение текста в текстовом поле, выбор нового элемента в выпадающем списке и т.д.).

Наблюдателями должны являться объекты классов, поддерживающих специальные интерфейсы (в классе наблюдателя должны быть определенные методы, о которых «знает» наблюдаемый и вызывает их при наступлении события). Такие классы в терминологии Swing называются слушателями.

#### **Интерфейс `MouseListener` и обработка событий от мыши**

События от мыши — один из самых популярных типов событий.

Практически любой элемент управления способен сообщить о том, что на него навели мышью, щелкнули по нему и т.д. Об этом будут оповещены все зарегистрированные слушатели событий от мыши.

Слушатель событий от мыши должен реализовать интерфейс `MouseListener`. В этом интерфейсе перечислены следующие методы:

- `public void mouseClicked(MouseEvent event)` — выполнен щелчок мышкой на наблюдаемом объекте
- `public void mouseEntered(MouseEvent event)` — курсор мыши вошел в область наблюдаемого объекта
- `public void mouseExited(MouseEvent event)` — курсор мыши вышел из области наблюдаемого объекта
- `public void mousePressed(MouseEvent event)` — кнопка мыши нажата в момент, когда курсор находится над наблюдаемым объектом
- `public void mouseReleased(MouseEvent event)` — кнопка мыши

отпущена в момент, когда курсор находится наднаблюдаемым объектом

Чтобы обработать нажатие на кнопку, требуется описать класс, реализующий интерфейс `MouseListener`. Далее необходимо создать объект этого класса и зарегистрировать его как слушателя интересующей нас кнопки. Для регистрации слушателя используется метод `addMouseListener()`.

Опишем класс слушателя в пределах класса окна `SimpleWindow`, после конструктора. Обработчик события будет проверять, ввел ли пользователь логин «Иван» и выводить сообщение об успехе или неуспехе входа в систему:

```
class MouseL implements MouseListener {
    public void mouseClicked(MouseEvent event) { if
(loginField.getText().equals("Иван"))
        JOptionPane.showMessageDialog(null, "Вход
выполнен"); else
        JOptionPane.showMessageDialog(null, "Вход НЕ
выполнен");
    }
    public void mouseEntered(MouseEvent event) {}
    public void mouseExited(MouseEvent event) {}
    public void mousePressed(MouseEvent event) {}
    public void mouseReleased(MouseEvent event) {}
}
```

Мы сделали слушателя вложенным классом класса `SimpleWindow`, чтобы он мог легко получить доступ к его внутренним полям `loginField` и `passwordField`. Кроме того, хотя реально мы обрабатываем только одно из пяти возможных событий мыши, описывать пришлось все пять методов (четыре имеют пустую реализацию). Дело в том, что в противном случае класс пришлось бы объявить абстрактным (ведь он унаследовал от интерфейса пустые заголовки методов) и мы не смогли бы создать объект этого класса. А мы должны создать объект слушателя и прикрепить его к кнопке. Для этого в код конструктора `SimpleWindow()` необходимо добавить команду:

```
ok.addMouseListener(new MouseL());
```

Это можно сделать сразу после команды:

```
JButton ok = new JButton("OK");
```

Создание слушателей с помощью анонимных классов. Чтобы кнопка `ok` обрела слушателя, который будет обрабатывать нажатие на нее, нам понадобилось описать новый (вложенный) класс. Иногда вместо вложенного класса можно обойтись анонимным. Анонимный класс не имеет имени и в программе, может быть, создан только один объект этого класса (создание которого совмещено с определением класса). Но очень часто слушатель пишется для того, чтобы обрабатывать события единственного объекта — в нашем случае кнопки `ok`, а значит, используется в программе только однажды: во время привязки к этому объекту. Таким образом, мы можем заменить вложенный класс анонимным. Для этого описание класса `MouseListener` можно просто удалить, а команду

```
ok.addMouseListener(new MouseL()); заменить на:
ok.addMouseListener(new MouseListener() {
    public void mouseClicked(MouseEvent event) { if
(loginField.getText().equals("Иван"))
    JOptionPane.showMessageDialog(null,          "Вход
выполнен");
    else JOptionPane.showMessageDialog(null, "Вход НЕ
выполнен");
}
    public void mouseEntered(MouseEvent event) {}
    public void mouseExited(MouseEvent event) {} public
void mousePressed(MouseEvent event) {} public void
mouseReleased(MouseEvent event) {}
});
```

Новый вариант выглядит более громоздко, чем первый.

Злоупотребление анонимными классами может сделать программу плохо читаемой. Однако в результате все действия с кнопкой (создание, настройка ее внешнего вида и команды обработки щелчка по ней) не разнесены, как в случае вложенных классов, а находятся рядом, что облегчает сопровождение (внесение изменений) программы. В случае простых (в несколько строк) обработчиков разумно делать выбор в пользу анонимных классов.

### **Класс `MouseAdapter`**

Программа стала выглядеть загроможденной главным образом из-за того, что помимо полезного для нас метода `mouseClicked()` пришлось

определять пустые реализации всех остальных, не нужных методов. Но этого можно избежать.

Класс `MouseAdapter` реализует интерфейс `MouseListener`, определяя пустые реализации для каждого из его методов. Можно унаследовать своего слушателя от этого класса и переопределить те методы, которые нам нужны.

В результате предыдущее описание слушателя будет выглядеть более компактно:

```
ok.addMouseListener(new MouseAdapter() { public
void mouseClicked(MouseEvent event) {
    if (loginField.getText().equals("Иван"))
        JOptionPane.showMessageDialog(null, "Вход выполнен");
    else JOptionPane.showMessageDialog(null, "Вход НЕ
        выполнен");
    }
});
```

### **Общая структура слушателей**

Кроме слушателей `MouseListener` визуальные компоненты `Swing` поддерживают целый ряд других слушателей.

Каждый слушатель должен реализовывать интерфейс

`***Listener`, где `***` — тип слушателя. Практически каждому из этих интерфейсов (за исключением тех, в которых всего один метод) соответствует пустой класс-заглушка `***Adapter`. Каждый метод интерфейса слушателя принимает один параметр типа `***Event`, в котором собрана вся информация, относящаяся к событию.

Чтобы привязать слушателя к объекту (который поддерживает соответствующий тип слушателей) используется метод `add***Listener(***Listener listener)`.

Например, слушатель `MouseListener` должен реализовать интерфейс с таким же именем, которому соответствует класс-заглушка `MouseAdapter`. Методы этого интерфейса обрабатывают параметр типа `MouseEvent`, а регистрируется слушатель методом `addMouseListener(MouseListener listener)`.

### **Слушатель фокуса `FocusListener`**

Слушатель `FocusListener` отслеживает моменты, когда объект

получает фокус (то есть становится активным) или теряет его.

Концепция фокуса очень важна для оконных приложений. В каждый момент времени в окне может быть только один активный (находящийся в фокусе) объект, который получает информацию о нажатых на клавиатуре клавишах (т.е. реагирует на события клавиатуры), о прокрутке колесика мышки и т.д. Пользователь активирует один из элементов управления нажатием мышки или с помощью клавиши Tab (переключаясь между ними).

Интерфейс `FocusListener` имеет два метода:

`public void focusGained(FocusEvent event)` — вызывается, когда наблюдаемый объект получает фокус

`public void focusLost(FocusEvent event)` — вызывается, когда наблюдаемый объект теряет фокус.

### **Слушатель колеса манипулятора мыши `MouseWheelListener`**

Слушатель `MouseWheelListener` оповещается при вращении колесика мыши в тот момент, когда данный компонент находится в фокусе. Этот интерфейс содержит всего один метод:

`public void mouseWheelMoved(MouseWheelEvent event)`.

### **Слушатель клавиатуры `KeyListener`**

Слушатель `KeyListener` оповещается, когда пользователь работает с клавиатурой в тот момент, когда данный компонент находится в фокусе. В интерфейсе определены методы:

`public void mouseKeyTyped(KeyEvent event)` — вызывается, когда с клавиатуры вводится символ

`public void mouseKeyPressed(KeyEvent event)` — вызывается, когда нажата клавиша клавиатуры

`public void mouseKeyReleased(KeyEvent event)` — вызывается, когда отпущена клавиша клавиатуры.

Аргумент `event` этих методов способен дать весьма ценные сведения. В частности, команда `event.getKeyChar()` возвращает символ типа `char`, связанный с нажатой клавишей. Если с нажатой клавишей не связан никакой символ, возвращается константа `CHAR_UNDEFINED`. Команда `event.getKeyCode()` возвратит код нажатой клавиши в виде целого числа типа `int`. Его можно сравнить с одной из многочисленных констант, определенных в классе `KeyEvent`: `VK_F1`, `VK_SHIFT`, `VK_D`, `VK_MINUS` и

т.д. Методы `isAltDown()`, `isControlDown()`, `isShiftDown()` позволяют узнать, не была ли одновременно нажата одна из клавиш-модификаторов `Alt`, `Ctrl` или `Shift`.

Рассмотрим пример программы, где отрабатывается выбор меню с помощью нажатия пункта меню и программа закрывается по нажатию сочетания клавиш клавиатуры. Полный листинг с кодом программы приведен в приложении А.

На рис. 16.1 представлен свернутый код программы.

```
ShortCut.java x
1  import javax.swing.AbstractAction;
2  import javax.swing.ImageIcon;
3  import javax.swing.JFrame;
4  import javax.swing.JMenu;
5  import javax.swing.JMenuBar;
6  import javax.swing.JMenuItem;
7  import javax.swing.KeyStroke;
8  import java.awt.EventQueue;
9  import java.awt.event.ActionEvent;
10 import java.awt.event.InputEvent;
11 import java.awt.event.KeyEvent;
12
13  public class ShortCut extends JFrame {
14
15      public ShortCut() {...}
16
17
18
19
20      private void initUI() {...}
21
22
23
24
25
26
27
28
29
30      private void createMenuBar() {...}
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70      private class MenuItemAction extends AbstractAction {...}
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87      public static void main(String[] args) {
88          /* java.awt.EventQueue.invokeLater(new Runnable() {
89              public void run() {
90                  new ShortCut().setVisible(true);
91              }
92          });
93          */
94          EventQueue.invokeLater(() -> {
95
96              var ex = new ShortCut();
97              ex.setVisible(true);
98          });
99
100
101      }
102  }
103
104
```

### *Рисунок 16.1 Окно программы Меню выбора*

Для обеспечения функциональности необходимо добавить выражения импорта для всех классов, которые мы будем использовать.

Соответственно это:

```
import javax.swing.AbstractAction;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.KeyStroke;
import java.awt.EventQueue;
import java.awt.event.ActionEvent;
import java.awt.event.InputEvent;
import java.awt.event.KeyEvent;
```

Класс ShortCut наследуется от базового класса JFrame. В нашем классе реализован конструктор ShortCut(){ } класса, который состоит из одного метода объявленного ниже:

```
public ShortCut () {

    initUI();
}
```

Метод initUI() объявлен с модификатором private, это вспомогательный метод класса, так называемый утилитный. Код метода ниже:

```
private void initUI() {
    createMenuBar();
    setTitle("Выбор меню");
    setSize(360, 250);
    setLocationRelativeTo(null);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
}
```

Задача метода создать базовое окно с заголовком “Выбор меню” и вызвать функцию создания меню createMenuBar(). Также наш метод initUI() делает следующее - устанавливает размеры окна – метод setSize(360, 250), реакцию окна по умолчанию на закрытие, а метод setLocationRelativeTo(null) используется, чтобы центрировать окно на экране.



Метод `createMenuBar()` также является приватным, он используется только внутри класса. И выполняет очень важные функции по обработке событий мыши. Код метода представлен на листинге 16.1.

Обратите внимание, что при объявлении типов переменных для создания объектов графического интерфейса используется служебное слово `var`. На первой лекции по языку Джава, когда мы обсуждали типы данных говорилось об этом типе. Такой тип означает `local variable` или локальная переменная. Ключевое слово `var` можно использовать только с локальными переменными, то есть переменными, которые объявлены, например внутри конструкторов или внутри блоков инициализации, или внутри методов.

Для создания меню используется класс `JMenu`. Мы создаем этот объект с помощью вызова конструктора `JMenu("Файл")`. Далее с помощью вызова конструктора класса `MenuItemAction()` создаются объекты `newMenuItem`, `openMenuItem`, `saveMenuItem` `exitMenuItem`.

Листинг 16.1 Метод `createMenuBar()`

```
private void createMenuBar() {

    var menuBar = new JMenuBar();

    var iconNew = new
ImageIcon("src/resources/new.png");
    var iconOpen = new
ImageIcon("src/resources/open.png");
    var iconSave = new
ImageIcon("src/resources/save.png");
    var iconExit = new
ImageIcon("src/resources/exit.png");

    var fileMenu = new JMenu("Файл");
    fileMenu.setMnemonic(KeyEvent.VK_F);

    var newMenuItem = new JMenuItem(new
MenuItemAction("Создать файл", iconNew,
                KeyEvent.VK_N));

    var openMenuItem = new JMenuItem(new
```

```

MenuItemAction("Открыть файл", iconOpen,
                KeyEvent.VK_O));

    var saveMenuItem = new JMenuItem(new
MenuItemAction("Сохранить файл", iconSave,
                KeyEvent.VK_S));

    var exitMenuItem = new JMenuItem("Выход",
iconExit);
    exitMenuItem.setMnemonic(KeyEvent.VK_E);
    exitMenuItem.setToolTipText("Выход из
приложения");

exitMenuItem.setAccelerator(KeyStroke.getKeyStroke(Ke
yEvent.VK_E, InputEvent.CTRL_DOWN_MASK));

    exitMenuItem.addActionListener((event) ->
System.exit(0));

    fileMenu.add(newMenuItem);
    fileMenu.add(openMenuItem);
    fileMenu.add(saveMenuItem);
    fileMenu.addSeparator();
    fileMenu.add(exitMenuItem);

    menuBar.add(fileMenu);

    setJMenuBar(menuBar);
}

```

Строчки кода ниже

```

var exitMenuItem = new JMenuItem("Выход", iconExit);
exitMenuItem.setMnemonic(KeyEvent.VK_E);

```

Переменная `exitMenuItem` представляет собой пункт меню выбрать который модно с помощью нажатия мыши и также по сочетанию клавиш для `setMnemonic(KeyEvent.VK_E)` означает нажатие <контрол+E>. При выборе меню файл и нажатии

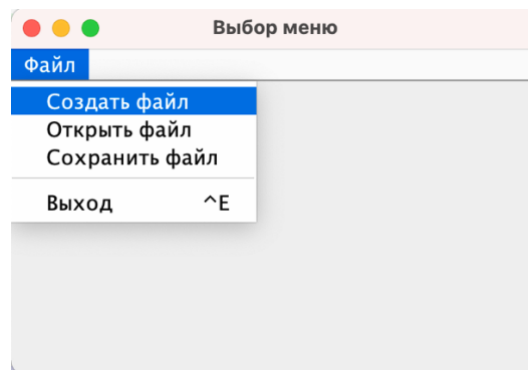


Рисунок 16.1 Окно программы Выбор меню.

Добавляем пункты меню с помощью метода `add(Component comp)`:

```
fileMenu.add(newMenuItem);  
fileMenu.add(openMenuItem);  
fileMenu.add(saveMenuItem);  
fileMenu.addSeparator();  
fileMenu.add(exitMenuItem);
```

Затем добавляем

```
menuBar.add(fileMenu);  
setJMenuBar(menuBar);
```

Также если включить в пункты меню пиктограммы

```
var menuBar = new JMenuBar();  
var iconNew = new  
ImageIcon("/Users/natala/Downloads/create_file.jpeg")  
;  
var iconOpen = new  
ImageIcon("/Users/natala/Downloads/open_file.jpeg" );  
var iconSave = new  
ImageIcon("/Users/natala/Downloads/save_file.jpeg ");  
var iconExit = new  
ImageIcon("/Users/natala/Downloads/exit_prog.jpeg");
```

На рис.16. представлено окно меню с использованием пиктограмм



Рисунок 16.1 Окно программы Выбор меню с пиктограммами

И наконец, для того чтобы запустить программу в классе `MenuItemAction`, который наследуется от класса `AbstractAction` описан конструктор класса с тремя параметрами `MenuItemAction(String text, ImageIcon icon, Integer mnemonic)`, Внутри конструктора происходит вызов конструктора родителя **`super`**(text и производится переопределение метода `actionPerformed(ActionEvent e)` для реализации реакции на событие.

Листинг 16.2 Класс `MenuItemAction`

```
private class MenuItemAction extends AbstractAction {

    public MenuItemAction(String text, ImageIcon
icon,
                                Integer mnemonic) {
        super(text);

        putValue(SMALL_ICON, icon);
    }
}
```

```

        putValue(MNEMONIC_KEY, mnemonic);
    }

    @Override
    public void actionPerformed(ActionEvent e) {

        System.out.println(e.getActionCommand());
    }
}

```

В методе main() точке входа в программу следующий код:

```

java.awt.EventQueue.invokeLater(new Runnable() {
    public void run() {
        new ShortCut().setVisible(true);
    }
});

```

Означает, что этом примере вы видите анонимный класс, который создается как **new Runnable()**. Этот анонимный класс переопределяет метод запуска интерфейса runnable. Затем этот анонимный класс создается и передается методу EventQueue.invokeLater(), который является статическим методом. Этот метод добавляет объект в очередь событий - eventQueue. В EvenQueue много событий, таких как события клавиатуры или события мыши или что-то еще какие-то события, ждущие обработки. Существует поток, который продолжает опрос данных из этой очереди. После того, как этот поток достигнет анонимного класса, который был здесь создан, то он выполнит метод run(), который создаст объект класса ShortCut (пользовательский класс окна) и сделает его видимым.

Вся сложность этого участка кода заключается в том, что новая часть ShortCut.setVisible (true) не выполняется в основном потоке, а в случае диспетчерского потока. В Swing вы должны выполнить весь код, который изменяет пользовательский интерфейс в потоке диспетчеризации<sup>1</sup> событий.

*Замечание. Поток диспетчеризации событий-это поток, который обрабатывает все события GUI и управляет вашим Swing GUI. Он всегда запускается где-то в коде Swing, если в вашей программе есть GUI.*

---

<sup>1</sup> <https://stackoverflow.com/questions/7217013/java-event-dispatching-thread-explanation>

*Причина, по которой это делается не явным образом, заключается в простоте - вам не нужно беспокоиться о запуске и управлении дополнительным потоком самостоятельно.*

Важно помнить, что классы Swing не являются потокобезопасными. Это означает, что вы всегда должны вызывать методы Swing из одного и того же потока, иначе вы рискуете получить странное или неопределенное поведение. GUI может быть изменен только из одного потока, поэтому вы должны обновлять свой GUI с помощью метода `invokeLater()`, это связано с проблемами параллелизма. Когда пользователь нажимает кнопку на GUI, то Runnable, который уведомляет всех слушателей, прикрепленных к кнопке, и попадает в очередь. То же самое происходит когда пользователь меняет размер И когда вы используете `invokeLater()`, то экземпляр ваш Runnable попадает в очередь.

То же самое можно переписать, используя лямбда выражения  
`EventQueue.invokeLater(() -> {`

```
        var ex = new ShortCut();  
        ex.setVisible(true);  
    });
```

### **Слушатель изменения состояния ChangeListener**

Слушатель `ChangeListener` реагирует на изменение состояния объекта. Каждый элемент управления по-своему определяет понятие «изменение состояния». Например, для панели со вкладками `JTabbedPane` это переход на другую вкладку, для ползунка `JSlider` — изменение его положения, кнопка `JButton` рассматривает как смену состояния щелчок на ней. Таким образом, хотя событие — это достаточно общее, необходимо уточнять его специфику для каждого конкретного компонента. В интерфейсе определен всего один метод:

```
public void stateChanged(ChangeEvent event).
```

### **Слушатель событий окна WindowListener**

Слушатель `WindowListener`, может быть, привязан только к окну и оповещается о различных событиях, произошедших с окном:

```
public void windowOpened(WindowEvent event) — окно открылось.
```

```
public void windowClosing(WindowEvent event) — попытка закрытия  
окна (например, пользователя нажал на крестик). Слово «попытка»  
означает, что данный метод вызовется до того, как окно будет закрыто и
```

может воспрепятствовать этому (например, вывести диалог типа «Вы уверены?» и отменить закрытие окна, если пользователь выберет «Нет»).

`public void windowClosed(WindowEvent event)` — окно закрылось.  
`public void windowIconified(WindowEvent event)` — окно свернуто.

`public void windowDeiconified(WindowEvent event)` — окно развернуто.

`public void windowActivated(WindowEvent event)` — окно стало активным.

`public void windowDeactivated(WindowEvent event)` — окно стало неактивным.

### **Слушатель событий компонента `ComponentListener`**

Слушатель `ComponentListener` оповещается, когда наблюдаемый визуальный компонент изменяет свое положение, размеры или видимость. В интерфейсе четыре метода:

`public void componentMoved(ComponentEvent event)` — вызывается, когда наблюдаемый компонент перемещается (в результате вызова команды `setLocation()`, работы менеджера размещения или еще по какой-то причине).

`public void componentResized(ComponentEvent event)` — вызывается, когда изменяются размеры наблюдаемого компонента.

`public void componentHidden(ComponentEvent event)` — вызывается, когда компонент становится невидимым.

`public void componentShown(ComponentEvent event)` — вызывается, когда компонент становится видимым.

### **Слушатель выбора элемента `ItemListener`**

Слушатель `ItemListener` реагирует на изменение состояния одного из элементов, входящих в состав наблюдаемого компонента. Например, выпадающий список `JComboBox` состоит из множества элементов, и слушатель реагирует, когда изменяется выбранный элемент. Также данный слушатель оповещается при выборе либо отмене выбора флажка `JCheckBox` или переключателя `JRadioButton`, изменении состояния кнопки `JToggleButton` и т.д. Слушатель обладает одним методом:

`public void itemStateChanged(ItemEvent event).`

### **Универсальный слушатель `ActionListener`**

Среди многочисленных событий, на которые реагирует каждый элемент управления (и о которых он оповещает соответствующих

слушателей, если они к нему присоединены), есть одно основное, вытекающее из самой сути компонента и обрабатываемое значительно чаще, чем другие. Например, для кнопки это щелчок на ней, а для выпадающего списка — выбор нового элемента.

Для отслеживания и обработки такого события может быть использован особый слушатель `ActionListener`, имеющий один метод:

```
public void actionPerformed(ActionEvent event).
```

У использования `ActionListener` есть небольшое преимущество в эффективности (так, при обработке нажатия на кнопку не надо реагировать на четыре лишних события — ведь даже если методы-обработчики пустые, на вызов этих методов все равно тратятся ресурсы). А кроме того, очень удобно запомнить и постоянно использовать один класс с одним методом и обращаться к остальным лишь в тех относительно редких случаях, когда возникнет такая необходимость.

Обработка нажатия на кнопку `ok` легко переписывается для `ActionListener`:

```
ok.addMouseListener(new ActionListener() {  
    public void actionPerformed(ActionEvent event) {  
        if (loginField.getText().equals("Иван"))  
            JOptionPane.showMessageDialog(null, "Вход  
выполнен");  
        else JOptionPane.showMessageDialog(null, "Вход НЕ  
выполнен");  
    }  
});
```

## **Варианты заданий на практическую работу №16**

1. Реализуйте игру-угадайку, которая имеет одно текстовое поле и одну кнопку. Он предложит пользователю угадать число между 0-20 и дает ему три попытки. Если ему не удастся угадать, то будет выведено сообщение, что пользователь допустил ошибку в угадывании и что число меньше / больше. Если пользователь попытался три раза угадать, то программу завершается с соответствующим сообщением. Если пользователь угадал, то программа должна тоже завершаться с соответствующим сообщением. Реализация приложения Java, который имеет макет границы и надписи для каждой области в макете. Теперь



определим события мыши, чтобы.

а. Когда мышь входит в область CENTER программа показывает диалоговое окно с сообщением “Добро пожаловать в ЦАО”

б. Когда мышь входит в область WEST программа показывает диалоговое окно с сообщением “Добро пожаловать в ЗАО”

с. Когда мышь входит в область SOUTH программа показывает диалоговое окно с сообщением “Добро пожаловать ЮАО”

д. Когда мышь входит в область NORTH программа показывает диалоговое окно с сообщением “Добро пожаловать в САО”

е. Когда мышь входит EAST программа показывает диалоговое окно с сообщением “Добро пожаловать в ВАО”

2. Реализуйте программу на Джава с использованием JTextArea и двумя следующего меню выбора:

а) Цвет: который имеет возможность выбора из три возможных: синий, красный и черный

б) Шрифт: три вида: “Times New Roman”, “MS Sans Serif”, “Courier New”.

*Замечание.* Вы должны написать программу, которая с помощью меню, может изменять шрифт и цвет текста, написанного в JTextArea

3. Реализуйте программу Проверка пароля на Джава с использованием Layout менеджеров компоновки. Окно программы должно иметь вид как на рис. 16.1.

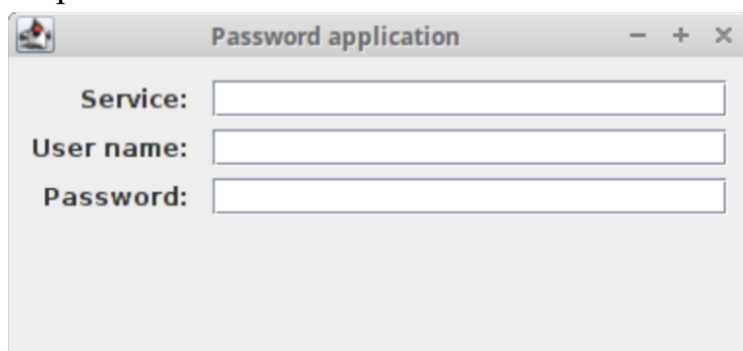
The image shows a standard Java Swing window titled "Password application". It has a title bar with a small icon on the left and standard window control buttons (minimize, maximize, close) on the right. The main content area of the window contains three text input fields arranged vertically. Each field is preceded by a label: "Service:", "User name:", and "Password:". The labels are in a bold, sans-serif font. The input fields are simple rectangular boxes with a thin border.

Рисунок 16.2 Окно программы Проверка пароля

