ЛАБОРАТОРНАЯ РАБОТА

РАЗРАБОТКА ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА С ИСПОЛЬЗОВАНИЕМ ФРЕЙМОВ, ЭЛЕМЕНТОВ УПРАВЛЕНИЯ

Цель: Научиться создавать простейшие GUI-приложения.

События

В основу *Java*-программирования наряду с другими положен механизм обработки событий.

Событие – объект, который описывает изменение состояния источника (нажатие кнопки, выбор пункта меню, разворачивание, сворачивание окошка, нажатие клавиши и т.д.). Источник – это объект, генерирующий события. Одно и то же событие может быть значимым для одних объектов и не существенным для других.

В *Java* различают два механизма обработки событий:

1) с помощью метода *handleEvent()* (применялся до версии *jdk 1.1* (*Java 1.0*));

2) с помощью модели делегирования событий.

Далее будем рассматривать особенности обработки событий с применением второго механизма. В соответствии с моделью делегирования событий в обработке событий участвуют 2 объекта: источник (*source*) и блок прослушивания (*listener* − интерфейс для перехода конкретного вида события от конкретного компонента). Источник − объект, генерирующий событие. Блок прослушивания – объект, получающий уведомление о возникновении события, зарегистрированного одним или несколькими источниками, путем вызова одного из его методов (методов блока прослушивания) для приема и обработки этих уведомлений.

Методы обработки событий находятся в пакете *java.awt.event*.

Классы событий

В корне иерархии классов событий *Java* находится класс *EventObject*, находящийся в пакете *java.util*. Класс *EventObject* содержит 2 метода:

- *Object getSource( )* – возвращает источник события;
- *toString( )* – возвращает название этого события в виде строки.

В *Java* определены несколько типов событий (табл. 3.1).

Таблица3.1

Классы событий

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 добавлен или удален из контейнера
FocusEvent	Генерируется, когда компонент получает или теряет фокус
ItemEvent	Генерируется, когда помечен флажок или элемент списка, сделан выбор элемента в списке, выбран или отменен элемент меню с меткой
KeyEvent	Генерируется, когда получен ввод с клавиатуры
MouseEvent	Генерируется, когда объект перетащен мышью (*dragged*), перемещен (*moved*), произошел щелчок (*clicked*), нажата (*pressed*) или отпущена (*released*) кнопка мыши, указатель мыши входит или выходит в/за границы компонента
TextEvent	Генерируется, когда изменено значение текстового поля
WindowEvent	Генерируется, когда окно активизировано, закрыто, развернуто, организован выход из него

Класс *ActionEvent*

Определяет четыре целочисленные константы, которые можно использовать для идентификации любых модификаторов, связанных с событием действия: *ALT_MASK*, *CTRL_MASK*, *META_MASK* и *SHIFT_MASK*. Кроме того, существует целочисленная константа *ACTION_PERFORMED*, которую можно применять для идентификации action-события.

Имеет два конструктора:

ActionEvent (*Object src, int type, String cmd*);

ActionEvent (*Object src, int type, String cmd, int modifiers*);

src – ссылка на объект, который генерирует события (для следующих описываемых классов значение аналогичное);

type – тип события (для следующих описываемых классов значение аналогичное);

cmd – командная строка события;

modifiers – указывает, какие клавиши-модификаторы были нажаты при генерации события (*Alt, Ctrl, Shift*).

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

FocusEvent

Событие этого класса идентифицируется константой *FOCUS_GAINED* и *FOCUS_LOST*. Конструкторы класса:

FocusEvent (*Component src, int type*);

FocusEvent (*Component src, int type, boolean temporaryFlag*);

temporaryFlag – устанавливается как *true*, если событие фокуса временное, иначе – *false*.

ItemEvent

Существует два типа *Item*-событий, которые определяются константами:

DESELECTED – пользователь отменил выбор элемента;
SELECTED – выбрал элемент списка.

Конструктор класса:

ItemEvent (*ItemSelectable src, int type, Object entry, int state*);
entry – передает конструкторы элемент, который генерировал *Item*-событие ;
state – состояние этого элемента.

Для того чтобы получить ссылку на объект *ItemSelectable*, используется метод *getItemSelectable*( ).

KeyEvent

Имеется три типа *Key*-событий, которые идентифицируются тремя константами:

KEY_PRESSED – клавиша нажата;
KEY_RELEASED – клавиша отпущена;
KEY_TYPED – генерируется только при нажатии символьной клавиши.

Конструктор класса:

KeyEvent (*Component src, int type, long when, int modifiers, int code*);
when – параметр, передающий конструктору системное время, когда была нажата клавиша;
modifiers – параметр, указывающий, какие модификаторы были нажаты вместе с клавишей;
code – параметр, передающий конструктору код клавиши.

MouseEvent

Существует семь типов *Mouse*-событий, которые идентифицируются семью константами:

MOUSE_CLICKED – пользователь щелкнул кнопкой мыши;
MOUSE_DRAGGED – пользователь перетащил мышь;
MOUSE_ENTERED – указатель мыши введен в компонент;
MOUSE_EXITED – указатель мыши выведен из компонента;
MOUSE_MOVED – мышь передвинута;

MOUSE_PRESSED – кнопка мыши нажата;
MOUSE_RELEASED – кнопка мыши освобождена.

Конструктор класса:

MouseEvent (*Component src, int type, long when, int modifiers, int x, int y, int clicks, boolean triggersPopup*);
x, y – координаты мыши;
clicks – подсчитывается количество щелчков;
triggersPopup – показывает, приводит ли это событие к появлению раскрывающегося меню; если да, то значение параметра соответствует *true*.
int getX( ); *int getY*( ); − методы для получения координат мышки.

TextEvent
TEXT_VALUE_CHANGED – событие, определяющее ввод текста пользователем в текстовое поле.

Конструктор класса:

TextEvent (*Object src, int type*);
WindowEvent
Существует семь типов событий *WindowEvent*:

WINDOW_ACTIVATED – окно активизировано;
WINDOW_CLOSED – окно закрыто;
WINDOW_DEACTIVATED – окно деактивизировано;
WINDOW_DEICONIFIED – окно развернуто из пиктограммы;
WINDOW_ICONIFIED – окно свернуто в пиктограмму;
WINDOW_OPENED – окно открыто;
WINDOW_CLOSING – пользователь потребовал закрытия окна.

Конструктор класса:

WindowEvent (*Window src, int type*);
Метод *Window getWindow*(); − возвращает *Window*-объект, который сгенерировал это событие.

Элементы-источники событий
В таблице 3.2 приведены некоторые элементы-источники событий, применяемые в Java, и их описание.

Таблица 3.2

Элементы-источники событий

Источник событий	Описание

1	2
Button (кнопка)	Генерирует *action*-события, в тот момент когда нажимается кнопка
Checkbox (флажок)	Генерирует *item*-события, когда флажок устанавливается/сбрасывается
Choice (список с выбором)	Генерирует *item*-события, когда изменяется выбор элемента в списке с выбором
List (список)	Генерирует *action*-события, когда на элементе списка выполнен двойной щелчок (мышью). Генерирует item-события, когда элемент выделяется или снимается выделение
MenuItem (пункт мею)	Генерирует *action*-события, когда пункт меню выделен. Генерирует события элемента, когда пункт меню с меткой выделен или выделение отменяется.
Scrollbar (полоса прокрутки)	Генерирует *adjustment*-события при манипуляциях с полосой прокрутки

Окончание табл. 3.2

1	2
TextField и TextArea (текстовое поле и текстовая область)	Генерирует *text*-события, когда пользователь вводит символ
Window (окно)	Генерирует *window*-события, когда окно активизируется, закрывается, деактивизируется, сворачивается в пиктограмму, разворачивается из пиктограммы, открывается или выполняется выход из него (*quit*).

Интерфейсы прослушивания событий

Модель делегирования событий содержит две части: источник событий и блоки прослушивания событий. Блоки прослушивания событий создаются путем реализации одного или нескольких интерфейсов прослушивания событий. Эти интерфейсы определены в пакете *java.awt.event*. Когда событие происходит, источник события вызывает соответствующий метод, определенный блоком прослушивания, и передает ему объект события в качестве параметра.

В табл. 3.3 приведены интерфейсы прослушивания событий и их методы. Когда класс реализует какой-нибудь из этих интерфейсов, то все методы интерфейса должны быть реализованы в этом классе. В случае, если среди методов интерфейса вам необходимы не все, а только некоторые из них, то для остальных методов в качестве реализации можно оставить пустые скобки {}.

Таблица 3.3

Интерфейсы прослушивания событий

Интерфейс	Описание, определяемые методы
1	2
ActionListener	Определяет один метод для приема action-событий: *void actionPerformed(ActionEvent ae)*
AdjustmentListener	Определяет один метод для приема adjustment-событий: *void adjustmentValueChanged(AdjustmentEvent ae)*
FocusListener	Определяет два метода для приема focus-события *void focusGained(FocusEvent fe)* *void focusLost(FocusEvent fe)*
ItemListener	Определяет один метод, распознающий события изменения состояние элемента *void itemStateChanged(ItemEvent ie)*

Окончание табл. 3.3

1	2
KeyListener	Определяет три метода, распознающих события клавиатуры *void keyPressed(KeyEvent ke)* *void keyReleased(KeyEvent ke)* *void keyTyped(KeyEvent ke)*
MouseListener	Определяет пять методов, распознающих события щелчка, входа в границы компонента, выхода из границ, нажатия/ отпускания клавиши мыши *void mouseClicked(MouseEvent me)* *void mouseEntered(MouseEvent me)* *void mouseExited(MouseEvent me)* *void mousePressed(MouseEvent me)* *void mouseReleased(MouseEvent me)*
MouseMotionListener	Определяет два метода, распознающих события перетаскивания/ перемещения мыши *void mouseDragged(MouseEvent me)* *void mouseMoved(MouseEvent me)*
TextListener	Определяет один метод, связанный с событием изменения текстового значения *void textChanged(TextEvent te)*
WindowListener	Определяет семь методов, связанных с окошком – событиями активации и т.д. *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)*

Классы пакета AWT: Component, Window, Frame

Класс Component

Абстрактный класс, инкапсулирующий все элементы визуального интерфейса пользователя. Все управляющие компоненты окна пользователя являются подклассами класса Component. В данном классе определено более 100 методов, которые отвечают за управление событиями, позиционирование, управление размерами, управление цветами, перерисовку.

Класс Window

Создает окно верхнего уровня на рабочем столе. Он расширяется классом Frame, который и представляет интерфейсное окно, окно с меню, обрамлением, необходимое для создания графического приложения с его компонентами.

Класс Frame

Инкапсулирует полноценное окно, имеющее строку заголовка, строку меню, обрамление и углы, изменяющие размеры окна.

Для создания окна *Frame* существуют два конструктора:

Frame ();
Frame (String Zagolovok);

Для установления размера фрейма существуют следующие методы:

void setSize (int Width, int Height);
void setSize (Dimension size);

Dimension – класс, содержащий поля *width* и *height*.
Метод, позволяющий сделать окно видимым:

void setVisible (boolean visibleFlag);

Пример кода для создания фреймового окна показан ниже.

Пример 3.1

```
import java.awt.*;
public class NewFrame extends Frame
{
TextArea ta;
```

```
public NewFrame ( String title )
{
super ( title );
setSize(300,200);
//…
}
public static void main ( String args [ ] )
{
NewFrame nf = new NewFrame (“Мой фрейм”);
nf.show ( );
}
}
```

Некоторые методы класса *Frame*:
String getTitle( ); – получить заголовок окна;
void setTitle (String); – установить заголовок окна;
void setResizable (boolean); - разрешить изменение размеров окна;
boolean isResizable(); - возвратить true, если размер окна можно изменять, иначе false.

Элементы управления Label, Button, Checkbox, Choice, List, Scrollbar

Элемент управления – это компоненты, которые предоставляют пользователю различные способы взаимодействия с приложением (например кнопки, флажки, полосы прокрутки и т.п.)

Элементы управления представлены следующими классами:

Label – с помощью класса *Label* можно создавать текстовые строки в окне *Java*-программ. По умолчанию текст будет выровнен влево, но используя методы

```
setAligment (Label.CENTER );
setAligment (Label.RIGHT );
```

строку можно выровнять по центру и по правому краю. Можно создавать выводимый текст либо при создании объекта класса *Label*, либо создать пустой объект и уже затем определить его текст вызовом метода *setText( )*.

Для этого класса существуют три конструктора:

```
Label first = new Label ( );
Label second = new Label (“some text”);
Label third = new Label (“some text”, Label.CENTER);
```

Button – представляет на экране кнопку. Имеет два конструктора:

```
Button first = new Button ( );
```

Button second = new Button (“some text”);

Сделать кнопку неактивной можно методом *void disable( ).*
Следующий пример демонстрирует обработку кнопки.

Пример 3.2
Листинг ButtonDemo.java

```
import java.awt.*;
import java.awt.event.*;
public class ButtonDemo extends Frame
implements ActionListener,WindowListener{
Button btn;
Label lb;
int count;
public ButtonDemo(){
super("Фреймовое окно с кнопкой");
setLayout(new FlowLayout(FlowLayout.LEFT));
btn=new Button("Нажмите кнопку");
setSize(300,200);
btn.addActionListener(this);
lb=new Label("Здесь текстовое поле");
count=0;
add(btn);
add(lb);
setVisible(true);
addWindowListener(this);
}
public void actionPerformed(ActionEvent ae)       {
count++;
lb.setText("Кнопка нажата  "+count+"  раз");
}
public void windowClosing(WindowEvent we){
this.dispose();
}
public void windowActivated(WindowEvent we){};
public void windowClosed(WindowEvent we){};
public void windowDeactivated(WindowEvent we){};
public void windowDeiconified(WindowEvent we){};
public void windowIconified(WindowEvent we){};
public void windowOpened(WindowEvent we){};

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

```
}
}
```

Checkbox – отвечает за создание и отображение кнопок с независимой фиксацией. Эти кнопки имеют два состояния: включено и выключено. Щелчок по такой кнопке приводит к тому, что ее состояние меняется на противоположное. Если разместить несколько кнопок с независимой фиксаций внутри элемента класса *CheckboxGroup*, то вместо них мы получаем кнопки с зависимой фиксацией. Для такой группы кнопок характерно то, что в один и тот же момент может быть включена только одна кнопка. Если нажать какую-либо кнопку из группы, то ранее нажатая кнопка будет отпущена.

Choice – создает раскрывающийся список.

Пример реализации списка из трех пунктов.

Пример 3.3

```
Choice choice = new Choice ( );
choice.addItem (“First”);
choice.addItem (“Second”);
choice.addItem (“Third”);
```

Методы класса *Choice*:

int countItems( ) – считать количество пунктов в списке;

String getItem(int) – возвратить строку с определенным номером в списке;

void select(int) – выбрать строку с определенным номером.

List – по назначению похож на *Choice*, но предоставляет пользователю не раскрывающийся список, а окно с полосами прокрутки. Такое окно содержит пункты выбора.

Создать объект класса *List* можно двумя способами:

1. Создать пустой список и добавить в него пункты методом *addItem()*. При этом размер списка будет расти при добавлении пунктов.

Пример 3.4

```
List list1 = new List ( );
   list1.addItem (“1”);
   list1.addItem (“2”);
   list1.addItem (“3”);
```

2. Создать пустой список, добавить пункты при помощи *addItem ( )*, при этом можно ограничить количество видимых в окне списка пунктов. Ниже показан пример, демонстрирующий список, в котором видно 2 элемента.

Пример 3.5

```
List list2 = new List (2, true );
```

```
list2.addItem (“1”);
list2.addItem (“2”);
list2.addItem (“3”);
```

Некоторые полезные методы класса *List*:

String getItem(int) – получить текст пункта с номером *int*;

int countItems( ) – посчитать количество пунктов в списке;

void clear( ) – очистить список;

void delItem(int) – удалить из списка пункт с номером *int*;

void delItems(int, int) – удалить из списка элементы с int по *int*;

int getSelectedIndex( ) – получить порядковый номер выделенного элемента списка;

void select(int) – выделить элемент списка с определенным номером.

Следующий пример демонстрирует обработку списка, раскрывающегося списка и кнопки с независимой фиксацией.

Пример 3.6

Листинг ListDemo.java

```
import java.awt.*;
import java.awt.event.*;
public class ListDemo extends Frame implements ItemListener{
List lst;Checkbox chb;Choice ch;
public ListDemo(){
super("Фреймовое окно");
setLayout(new FlowLayout(FlowLayout.CENTER));
setSize(300,200);
lst = new List (2, false );
lst.addItem ("1 BSUIR");
lst.addItem ("2 BSEU");
lst.addItem ("3 BSU");
chb=new Checkbox("Кнопка с независимой фиксацией");
ch=new Choice();
ch.add("Сюда переносятся строки со списка");
add(lst);
add(ch);
add(chb);
setVisible(true);
lst.addItemListener(this);
}
public void itemStateChanged(ItemEvent ie){
ch.addItem(lst.getSelectedItem());
}
public static void main(String args[]){
```

```
ListDemo list=new ListDemo();
}
}
```

Scrollbar – определяет полосу прокрутки. Создать полосу прокрутки можно следующим образом:

new Scrollbar ( );

new Scrollbar (Scrollbar.VERTICAL);

new Scrollbar (<ориентация>, <текущее значение>, <видно>, <минимальное значение>, <максимальное значение>);

<ориентация> – ориентация полосы, которая задается константами *Scrollbar.HORIZONTAL, Scrollbar.VERTICAL*.

<текущее значение> – начальное значение, в которое помещается бегунок полосы прокрутки.

<видно> – сколько пикселов прокручиваемой области видно и на сколько пикселов эта область будет прокручена при щелчке мышью на полосе прокрутки.

<минимальное значение> – минимальное значение полосы прокрутки.

<максимальное значение> – максимальное значение полосы прокрутки.

Элементы управления TextField и TextArea

Эти два класса позволяют отображать текст с возможностью его выделения и редактирования. Это по сути маленькие текстовые редакторы – однострочный (*TextField*) и многострочный (*TextArea*).

Создать текстовое поле и текстовую область можно следующими способами:

TextField tf = new TextField (50);

TextArea ta = new TextArea (5, 30);

Чтобы запретить или разрешить редактирование текста в окне, можно воспользоваться методом *void setEditable(boolean).*

tf.setEditable (false);

ta.setEditable (false);

Некоторые методы классов *TextField* и *TextArea*:

String getText( ) – читать текст;

void setText(String) – отобразить текст;

void selectAll( ) – выделить весь текст;

int getColumns( ) – возвратить количество символов строки.

Специфические методы *TextField*:

void setEchoChar(char) – установить символ маски (при вводе паролей);

char getEchoChar( ) − узнать символ маски.

Специфические методы для *TextArea*:

int getRows( ) – считать количество строк в окне;

void insertText(String, int) – вставить текст в определенной позиции int;

void replaceText(String, int, int) – заменить текст между заданными начальной и конечной позицией.

Следующий пример демонстрирует приложение с элементами управления: кнопкой (*Button*), списком (*List*), раскрывающимся списком(*Choice*), текстовой строкой (*Label*), текстовым полем (*TextField*). Введенное в текстовом поле слово при нажатии кнопки добавляется как в список, так и в раскрывающийся список. Также реализован механизм закрытия фрейма.

Пример 3.7

Листинг GUISample.java

```
import java.io.*; //импортирование пакета ввода-вывода
import java.awt.*; //импортирование пакета awt
import java.awt.event.*; //импортирование пакета поддержки событий
public class GUISample extends Frame{ //объявление класса GUISample
Button b1 = new Button("Add"); //создание кнопки с надписью "Add"
Choice ch1=new Choice(); //создание раскрывающегося списка
TextField tf1 = new TextField(); //создание текстового поля (строки
//ввода)
Label label1 = new Label("Enter your text here:"); //создание текстовой
//строки
List l1 = new List(); //создание списка
public GUISample(){ //объявление конструктора класса
setLayout(null); //отключение менеджера компоновки
setSize(600,400); //установка размеров фрейма
setTitle("This is my Frame"); //установка заголовка фрейма
setBackground(Color.cyan); //установка цвета заднего фона фрейма
add(b1); //добавление кнопки к окну
b1.setBounds(220,200,84,24); //установка размеров кнопки
b1.setForeground(Color.black); //установка цвета переднего фона кнопки
b1.setBackground(Color.magenta); //установка цвета заднего фона кнопки
add(ch1); //добавление раскрывающегося списка к окну
ch1.setBounds(50,120,120,20); //установка размеров раскрывающегося
//списка
add(tf1); //добавление текстового поля к окну
tf1.setBounds(200,80,120,20); //установка размеров текстового поля
add(label1); //добавление текстовой строки к окну
label1.setBounds(200,55,120,20); //установка размеров текстовой строки
add(l1); //добавление списка к окну
l1.setBackground(Color.white); //установка цвета заднего фона списка
l1.setBounds(350,120,200,216); //установка размеров списка
/*регистрация блока прослушивания событий типа WindowEvent*/
addWindowListener(new WindowClose());
```

```
/*регистрация блока прослушивания событий типа ActionEvent*/
b1.addActionListener(new ButtonAdd());
}
/*объявление класса-адаптера для обработки Window-событий*/
class WindowClose extends WindowAdapter {
/*метод, который вызывается при закрытии окна*/
public void windowClosing(WindowEvent we) {
setVisible(false); //фрейм-окно становится невидимым
}
}
/*объявление класса для обработки Action-событий (класс ButtonAdd реализует
интерфейс ActionListener)*/
class ButtonAdd implements ActionListener {
/*реализация метода,  который вызывается при наступлении action-события*/
public void actionPerformed(ActionEvent event) {
/*добавление текста из текстового поля в раскрывающийся список*/
ch1.add(tf1.getText());
/*добавление текста из текстового поля в список*/
l1.add(tf1.getText(),2);
}
}
static public void main(String args[]){ //объявление метода main()
GUISample MyFrame=new GUISample(); //создание экземпляра класса
GUISample
 MyFrame.setVisible(true); //выведение окна на экран дисплея
} }
```

Результаты работы программы представлены на рис. 3.1.

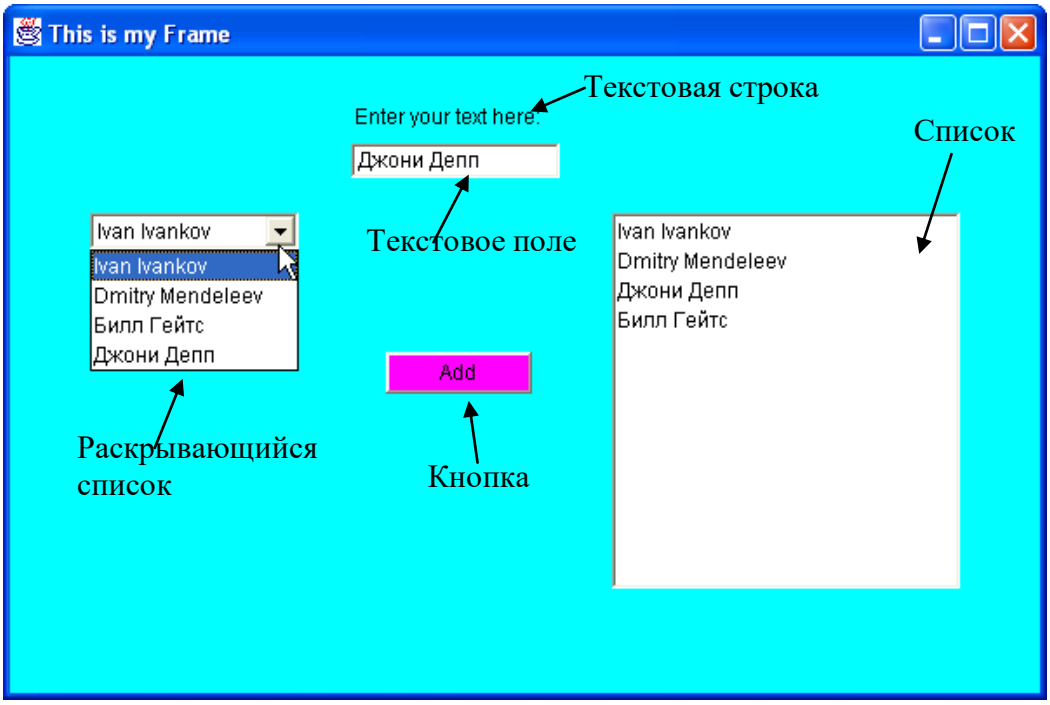

Рис. 3.1. Результат работы программы

Диалоговые окна

Диалоговые окна подобны фрейм-окнам за исключением того, что они – всегда дочерние окна для окна верхнего уровня. Кроме того, диалоговые окна не имеют строки меню. В других отношениях они функционируют подобно фреймовым окнам. Можно, например, добавить к ним элементы управления тем же способом, каким добавляются элементы управления к фреймовому окну. Диалоговые окна могут быть модальными или немодальными. Когда модальное диалоговое окно активно, весь ввод направляется к нему, пока оно не будет закрыто. Это означает, что невозможно обратиться к другим частям программы до тех пор, пока не закрыто диалоговое окно. Когда немодальное диалоговое окно активно, фокус ввода может быть направлен другому окну программы. Таким образом, другие части программы остаются активными и доступными. Диалоговые окна обслуживает класс *Dialog*. Обычно используются следующие конструкторы класса:

Dialog(Frame parentWindow, boolean mode);
Dialog(Frame parentWindow, String title, boolean mode);

parentWindow – владелец диалогового окна. Если *mode* имеет значение *true*, диалоговое окно является модальным. Иначе оно немодальное. Заголовок диалогового окна можно передать через параметр *title*.

Следующий пример демонстрирует фреймовое окно с меню, из которого (выбором пунтов меню *File→DemoDialog*) вызывается модальное диалоговое окно.

Пример 3.8
Листинг Frame1.java

```
import java.awt.*;
import java.awt.event.*;
public class Frame1 extends Frame
implements ActionListener,WindowListener{
Menu file;MenuItem item1;
public Frame1(){
super("Фреймовое окно с меню");
setSize(500,300);
//создать строку главного меню и добавить его во фрейм
MenuBar mbar=new MenuBar();
setMenuBar(mbar);
//создать элемент меню
file=new Menu("File");
mbar.add(file);
file.add(item1=new MenuItem("DemoDialog"));
item1.addActionListener(this);
setVisible(true);
addWindowListener(this);
}
public void actionPerformed(ActionEvent ae)       {
DemoDialog d=new DemoDialog(this,"Диалоговое окно",true);
}
public void windowClosing(WindowEvent we){
this.dispose();
}
public void windowActivated(WindowEvent we){};
public void windowClosed(WindowEvent we){};
public void windowDeactivated(WindowEvent we){};
public void windowDeiconified(WindowEvent we){};
public void windowIconified(WindowEvent we){};
public void windowOpened(WindowEvent we){};
public static void main(String args[])
{Frame1 f=new Frame1();
}
class DemoDialog extends Dialog implements ActionListener{
        Button btn;
        public DemoDialog(Frame1 ff, String title,boolean b){
        super(ff,title,b);
        setLayout(new FlowLayout(FlowLayout.LEFT));
        btn=new Button("Закрыть");
        setSize(300,200);
        add(btn);
        btn.addActionListener(this);
```

```
        setVisible(true);
        }
        public void actionPerformed(ActionEvent ae){
                this.dispose();
        }
}
}
```

На рисунке 3.2 показан результат работы программы:

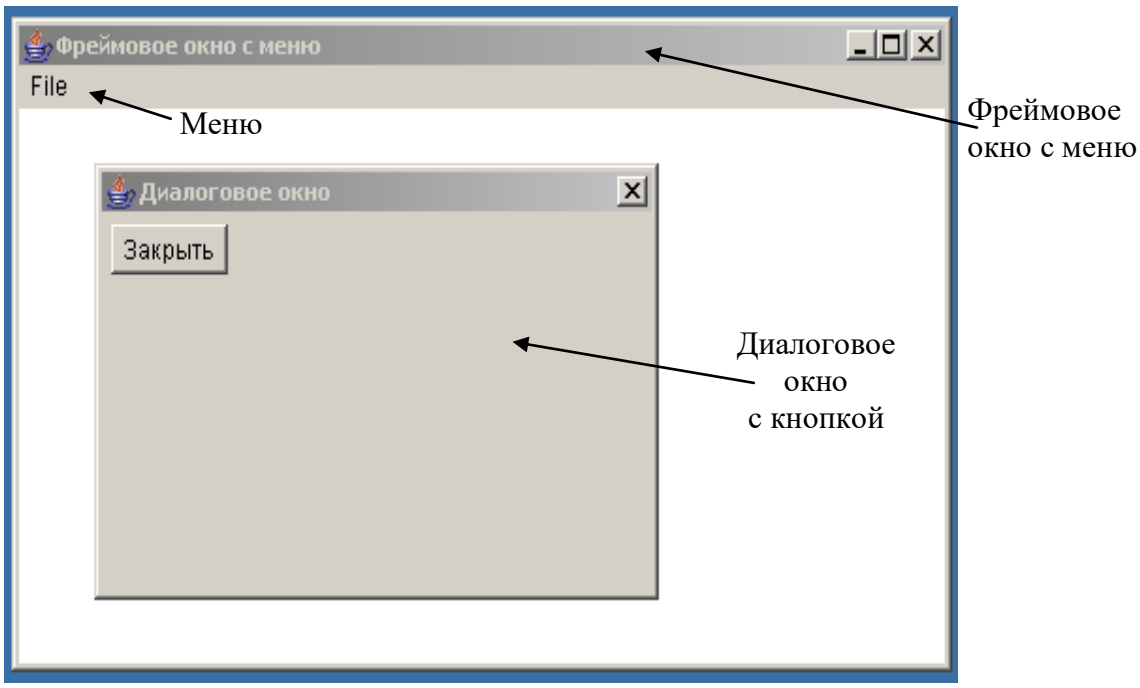

Рис. 3.2. Результат работы программы, приведенной в листинге Frame1.java

Задания для самостоятельного выполнения

1. Разработать приложение управления тремя списками, расположенными горизонтально. Приложение должно обеспечивать перемещение любого выбранного элемента или содержимого всего списка в следующий список по часовой стрелке: как из первого – во второй, из второго в третий, из третьего в первый. Элемент при перемещении должен исчезать из одного списка и появляться в другом. Помимо того приложение должно обеспечивать управление вторым списком – добавление нового элемента, редактирование, удаление.

2. Разработать приложение управления тремя списками, расположенными горизонтально. Приложение должно обеспечивать перемещение выбранного элемента из первого во второй, из второго в третий, из третьего в первый список и наоборот. Направление перемещения элемента из списка в список должно определяться выбором из набора флажков (*CheckboxGroup*). Элемент при перемещении должен исчезать из одного списка и появляться в другом. Помимо того приложение должно обеспечивать управление всеми списками – добавление нового элемента, редактирование, удаление.

3. Разработать приложение, обеспечивающее возможность множественного выбора элементов из списка. Выбранные элементы должны образовывать строку текста и помещаться в текстовое поле. Предусмотреть возможность вывода сообщения в диалоговое окно (*Dialog*) в случае, если суммарное количество символов будет превышать 100.

4. Разработать приложение, реализующее калькулятор. Приложение должно иметь строку редактирования (*TextField*), набор кнопок 0…9, кнопки арифметических действий – суммирование, вычитание, деление, умножение, память.

5. Разработать приложение, реализующее калькулятор. Приложение должно иметь две строки редактирования (*TextField*). Набор флажков (*CheckboxGroup*) определяет, какое арифметическое действие необходимо выполнить: суммирование, вычитание, деление, умножение, память.

6. Разработать приложение, обеспечивающее поиск в двух списках несовпадающих фрагментов текста. Строки, в которых будут найдены искомые фрагменты, должны быть выведены в диалогове окно (*Dialog*) (предполагается, что несколько строк может иметь такие фрагменты). Помимо этого приложение должно обеспечивать управление содержимым списков – добавление нового элемента, редактирование, удаление.

7. Разработать приложение управления тремя списками, расположенными на диалоге горизонтально. Приложение должно обеспечивать перемещение некоторого (указанного в наборе флажков (*CheckboxGroup*)) количества выбранных элементов из списка в список. Перемещение элементов осуществлять слева направо. Элемент при перемещении не исчезает, а выделяется. Помимо этого приложение должно обеспечивать заполнение помеченного флажком списка 10 строками. Предусмотреть очистку помеченного списка.

8. Разработать приложение управления списком. Вывести два флажка (*Checkbox*). При первом включенном флажке осуществляется выбор всех нечетных строк, при втором включенном флажке осуществляется выбор всех четных строк и перенос их в раскрывающийся список (*Choice*).

9. Разработать приложение управления списком. Вывести два флажка (*Checkbox*). При первом включенном флажке осуществляется выбор всех нечетных строк и их удаление, при втором включенном флажке осуществляется выбор всех четных строк и перенос их во второй список. Предусмотреть обновление элементов списка и очистку второго списка.

10. Разработать приложение, реализующее калькулятор. Приложение должно иметь три строки редактирования (*TextField*) – для двух операндов и результата. Набор флажков (*CheckboxGroup*) определяет, какое арифметическое действие необходимо выполнить: суммирование, вычитание, деление, умножение, очистку окон редактирования.

11. Разработать приложение управления тремя списками («Фамилия», «Имя», «Отчество») и строки редактирования (*TextField*). В строку редактирования вводится информация в формате «Фамилия Имя Отчество». По завершении ввода фамилия должна появиться в списке «Фамилия», имя в списке «Имя», отчество в списке

«Отчество». Предусмотреть вывод сообщения в диалогое окно (*Dialog*), если количество введенных в списки ФИО будет превышать 10.

12. Разработать приложение управления тремя списками («Фамилия», «Имя», «Отчество») и строки редактирования (*TextField*). В строку редактирования вводится информация в формате «Фамилия Имя Отчество». По завершении ввода фамилия должна появиться в списке «Фамилия», имя в списке «Имя», отчество в списке «Отчество». Предусмотреть возможность множественного выбора фамилий или отчеств в зависимости от выбора в наборе флажков (*CheckboxGroup*) и вывода всех их в отсортированном порядке в диалоговое окно (*Dialog*).

13. Разработать приложение, обеспечивающее поиск в двух раскрывающихся списках (*Choice*) фрагмента текста. Набором флажков (*CheckboxGroup*) указывать, в каком списке будет осуществляться поиск. Строки, в которых будет найден искомый фрагмент, должны быть выделены (предполагается, что несколько строк может иметь искомый фрагмент). Помимо этого приложение должно обеспечивать управление содержимым списков – добавление нового элемента, редактирование, удаление.

14. Разработать приложение, обеспечивающее возможность множественного выбора элементов из списка. Выбранные элементы должны образовывать строку текста и выводиться в соседний список. Предусмотреть возможность вывода сообщения в диалоговое окно (*Dialog*) в случае, если суммарное количество символов будет превышать 100.

15. Разработать приложение управления тремя списками («Фамилия», «Имя», «Отчество») и строкой редактирования (*TextField*). Для отображения строки редактирования вызывается диалоговое окно (*Dialog*). В строку редактирования вводится информация в формате «Фамилия Имя Отчество». По завершении ввода диалоговое окно закрывается, фамилия должна появиться в списке «Фамилия», имя в списке «Имя», отчество в списке «Отчество». Предусмотреть возможность множественного выбора фамилий и записи их в отсортированном порядке в четвертый список.