# Лабораторная работа №7

# **ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС**

Цель: проектирование простого графического интерфейса пользователя c добавлением основных элементов управления библиотеки Swing, реализация действий к соответствующим элементам.

Современные программы нуждаются в графическом интерфейсе пользователя (GUI). Пользователи отвыкли работать через консоль: они управляют программой и вводят входные данные посредством так называемых элементов управления (в программировании их также называют визуальными компонентами), к которым относятся кнопки(JButton), текстовые поля(JTextField), выпадающие списки(JComboBox) и т.д.

В Java есть три библиотеки визуальных компонентов для создания графического интерфейса пользователя. Самая ранняя из них называется AWT. Считается, что при ее проектировании был допущен ряд недочетов, вследствие которых с ней довольно сложно работать. Библиотека Swing разработана на базе AWT и заменяет большинство ее компонентов своими, спроектированными более тщательно и удобно. Третья, самая новая библиотека, называется SWT.

Каждая библиотека предоставляет набор классов для работы с кнопками, списками, окнами, меню и т.д., но эти классы спроектированы по-разному: они имеют различный набор методов с разными параметрами, поэтому «перевести» программу с одной библиотеки на другую (например, с целью увеличения быстродействия) не так-то просто.

Каждая GUI-программа запускается в окне и по ходу работы может открывать несколько дополнительных окон.

В библиотеке Swing описан класс JFrame, представляющий собой окно с рамкой и строкой заголовка (с кнопками «Свернуть», «Во весь экран» и «Закрыть»). Оно может изменять размеры и перемещаться по экрану.

Конструктор JFrame() без параметров создает пустое окно. Конструктор JFrame(String title) создает пустое окно с заголовком title.

Чтобы написать простейшую программу, выводящую на экран пустое окно, нам потребуется еще три метода:

* *setSize(int width, int height)* — устанавливает размеры окна. Если не задать размеры, окно будет иметь нулевую высоту независимо от того, что в нем находится и пользователю после запуска придется растягивать окно вручную. Размеры окна включают не только «рабочую» область, но и границы и строку заголовка.
* *setDefaultCloseOperation(int operation)* — позволяет указать действие, которое необходимо выполнить, когда пользователь закрывает окно нажатием на крестик. Обычно в программе есть одно или несколько окон при закрытии которых программа прекращает работу. Для того, чтобы запрограммировать это поведение, следует в качестве параметра operation передать константу EXIT\_ON\_CLOSE, описанную в классе JFrame.
* *setVisible(boolean visible)* — когда окно создается, оно по умолчанию невидимо. Чтобы отобразить окно на экране, вызывается данный метод с параметром true. Если вызвать его с параметром false, окно снова станет невидимым.

Пример 8.1. Программа, создающая окно, выводящая его на экран и завершающая работу, после нажатия на кнопку, закрывающую окно.

import javax.swing.JFrame;

public class Panel{

public static void main(String[] args) {

JFrame testWindow = new JFrame("Пробное окно");

testWindow.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

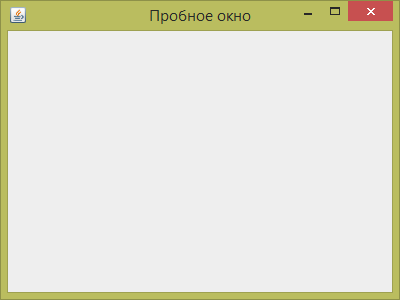
testWindow.setSize(400, 300);

testWindow.setVisible(true);

}

}

В результате мы получим:

  
Рис. 8.1. Пробное окно

Как правило, перед отображением окна, необходимо совершить гораздо больше действий, чем в этой простой программке. Необходимо создать множество элементов управления, настроить их внешний вид, разместить в нужных местах окна. Кроме того, в программе может быть много окон и настраивать их все в методе main() неудобно и неправильно, поскольку нарушает принцип инкапсуляции: держать вместе данные и команды, которые их обрабатывают. Логичнее было бы, чтобы каждое окно занималось своими размерами и содержимым самостоятельно. Поэтому наш пример было бы записать правильнее так:

public class Panel extends JFrame {

public Panel(){

super("Пробное окно");

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setSize(250, 100);

}

}

public class Main {

public static void main (String [] args) {

JFrame testWindow = new Panel();

testWindow.setVisible(true);

}

}

Из примера видно, что окно описывается в отдельном классе, являющемся наследником JFrame и настраивающее свой внешний вид и поведение в конструкторе (первой командой вызывается конструктор суперкласса). Метод main() содержится в другом классе, ответственном за управление ходом программы. Каждый из этих классов очень прост, каждый занимается своим делом, поэтому в них легко разбираться и легко сопровождать (т.е. совершенствовать при необходимости).

Обратите внимание, что метод setVisible() не вызывается в классе Panel, что вполне логично: за тем, где какая кнопка расположена и какие размеры оно должно иметь, следит само окно, а вот принимать решение о том, какое окно в какой момент выводится на экран — прерогатива управляющего класса программы.

Напрямую в окне элементы управления не размещаются. Для этого служит панель содержимого, занимающая все пространство окна. Обратиться к этой панели можно методом getContentPane() класса JFrame. С помощью метода add(Component component) можно добавить на нее любой элемент управления.

Пример 8.2. Добавление кнопки на панель.

JButton myButton = new JButton("Кнопка");

getContentPane().add(myButton);

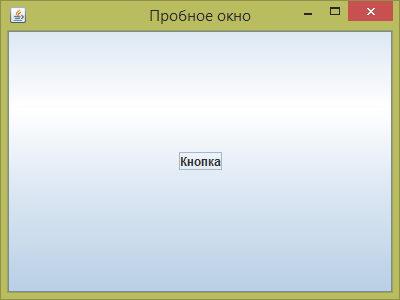
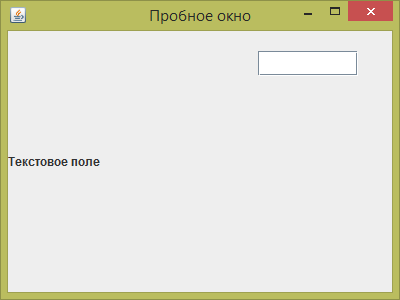
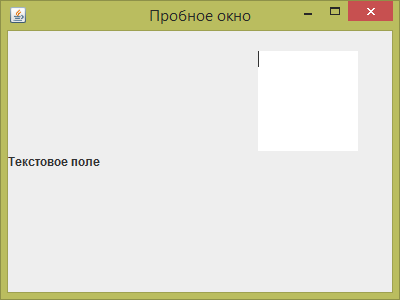
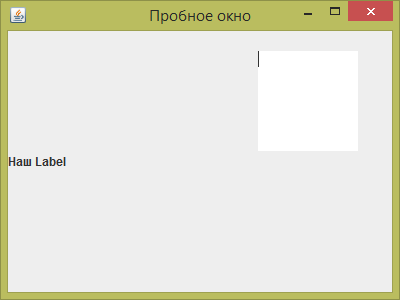
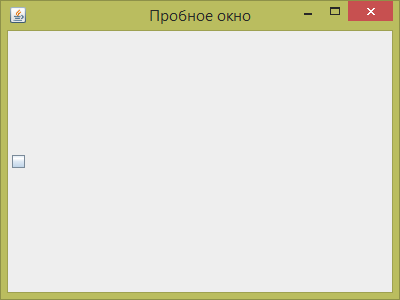
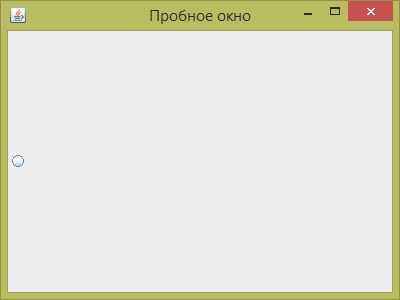
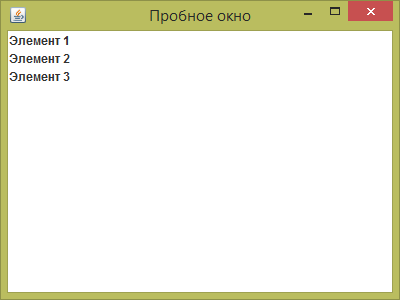
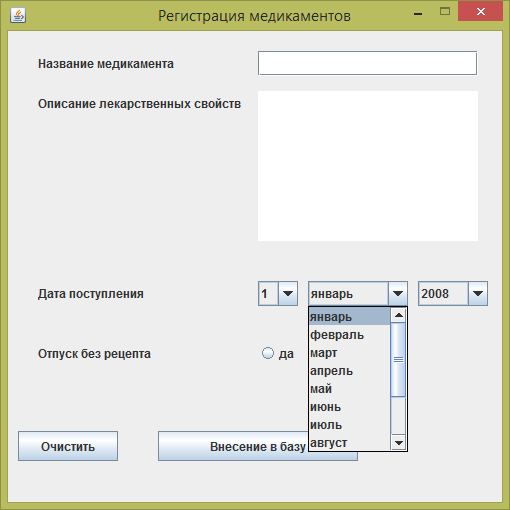


Рис. 8.2. Кнопка

Существует много различных элементов управления библиотеки Swing. Наиболее распространены из них такие, как:

* Текстовое поле(*JTextField*) 
* Текстовая область(*JTextArea*) 
* Лэйбл(*JLabel*) 
* Флажок(*JCheckBox*) 
* Радиокнопка(*JRadioButton*) 
* Список(*JList*) 
* Раскрывающиеся списки(*JComboBox*) 
* и так далее.

Панель JPanel — это элемент управления, представляющий собой прямоугольное пространство, на котором можно размещать другие элементы. JPanel является потомком класса Container. Элементы добавляются и удаляются методами, унаследованными от класса Container.

В примере с кнопкой мы наблюдали, как добавленная на панель содержимого кнопка заняла все ее пространство. Это происходит не всегда. На самом деле у каждой панели есть так называемый менеджер размещения, который определяет стратегию взаимного расположения элементов, добавляемых на панель. Его можно изменить методом setLayout(LayoutManager manager). Но чтобы передать в этот метод нужный параметр, необходимо знать, какими бывают менеджеры.

## **Ручное размещение элементов**

Если в качестве менеджера размещения панели установить null, элементы не будут расставляться автоматически. Координаты каждого элемента необходимо в этом случае указать явно, при этом они никак не зависят от размеров панели и от координат других элементов. По умолчанию координаты равны нулю (т.е. элемент расположен в левом верхнем углу панели). Размер элемента также необходимо задавать явно (в противном случае его ширина и высота будут равны нулю и элемент отображаться не будет).

Координаты элемента можно задать следующим методом:

*setLocation(int x, int y)*

Размер элемента задается следующим методом:

*setSize(int width, int height)*

Создадим панель, с которой не будет связано никакого менеджера размещения и вручную разместим на ней две кнопки.

Пример 8.3. Размещение на панели двух кнопок

public class Panel extends JFrame{

Panel(){

super("Пробное окно");

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

JPanel panel = new JPanel();

panel.setLayout(null);

JButton button = new JButton("Кнопка1");

button.setSize(100, 50);

button.setLocation(40,20);

panel.add(button);

button = new JButton("Кнопка2");

button.setSize(120, 40);

button.setLocation(150,50);

panel.add(button);

setContentPane(panel);

setSize(250, 150);

}

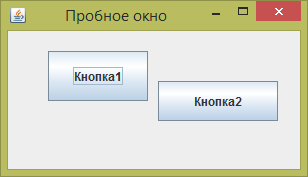


Рис. 8.7. Произвольное размещение.

Таблица 1.1. Основные методы классов библиотеки Swing

|  |  |
| --- | --- |
| Метод | Что делает |
| ***Component*** | |
| От своего предка Component все визуальные элементы перенимают общее для них всех поведение, связанное с их визуальной и функциональной сторонами. Вот список основных, выполняемых компонентами, функций и методов для их реализации: | |
| [Font](http://spec-zone.ru/RU/Java/Docs/7/api/java/awt/Font.html) getFont()  void setFont([Font](http://spec-zone.ru/RU/Java/Docs/7/api/java/awt/Font.html) f) | Получение и установка шрифта надписей на компоненте |
| void  [setForeground](http://spec-zone.ru/RU/Java/Docs/7/api/javax/accessibility/AccessibleComponent.html#setForeground(java.awt.Color))([Color](http://spec-zone.ru/RU/Java/Docs/7/api/java/awt/Color.html) c)  [Color](http://spec-zone.ru/RU/Java/Docs/7/api/java/awt/Color.html) getForeground() | Получение и установка цвета самого шрифта |
| Color getBackground()  void  setBackground(Color с) | Получение и установка цвета фона, на котором отображается текст; |
| getMaximumSize()  void  setMaximumSize(Dimension d) | Получение и установка максимального размера; |
| getMinimumSize()  void  setMinimumSize(Dimension d) | Получение и установка минимального размера; |
| getPreferredSize()  void  setPreferredSize(Dimension d) | Получение и установка желательного размера; |
| void  setLocation(int x, int y)  void  setLocation(Point p) | Задание координат и размера компонента (x, y – координаты левой верхней точки; р – объект класса Point) |
| void  setSize(int w, int h)  void  setSize(Dimension d) | Задание размера компонента  (w – ширина, h – высота; d – объект класса Dimension) |
| void  setBounds(int x,int y, int w, int h)  void  setBounds(Rectangle r) | Задание координат и размера компонента  (x, y – координаты левой верхней точки; w – ширина, h – высота; r – объект класса Rectangle) |
| paint()  update()  repaint() | Отображение компонента |
| ***JButton*** | |
| void setText(String) | Установить надпись на компоненте |
| String getText() | Считать надпись на компоненте |
| Void setActionCommand(String) | Установить имя команды, выполняемой при нажатии кнопки |
| String getActionCommand(void) | Считать имя команды, выполняемой при нажатии кнопки |
| ***JLabel*** | |
| void setText(String) | Установить текст в *лэйбл* |
| String getText() | Получить *лэйбл* |
| Vjid setAlignment(int align)  Int getAlignment() | Выравнивание текста |
| void setIcon(Icon) | Графическое изображение в *лэйбле* |
| Icon getIcon() | Получение графического изображения с *лэйбла* |
| ***JCheckBox*** | |
| boolean getSelected()  void setSelected(boolean ) | Управление состоянием флажков(то есть стоит там галочка или нет) |
| ***JComboBox*** | |
| addItem(Object) | Добавление новый пунктов в список |
| void removeItem(String)  void remove ItemAt(int)  void RemoveAllItems() | Удаление строк из списка |
| String getSelectedItem()  int getSelectedIndex() | Определение текущего выбранного элемента в списке |
| int getItemCount() | Получение количества элементов в списке |
| void setSelectedIndex (int index)  void setSelectedItem (Object obj) | Выбор пункта из списка в ходе программы |
| void setEditable(boolean) | Сделать список редактируемым |
| ***JList*** | |
| Object getSelectedValue()  int getSelectedIndex() | Определение выбранного элемента списка |
| ***JTextField*** | |
| String getText() | Получение строки из поля |
| void setText(String) | Установление в поле текста |
| boolean isEditable()  void setEditable(boolean) | Сделать поле редактируемым |
| ***JTextArea*** | |
| void append(String) | Добавление строки в конец текста |
| void insert(String. int) | Вставки строки в область |
| Void replaceRange(String str, int, int) | Замена символов от s1 до s2 -1 строкой str |

## **События**

Ну вот мы установили на панели все нам необходимые элементы, разместили их так, как нам необходимо, и следующим шагом будет прикрепление к этим элементам соответствующего действия.

Обработка любого события (нажатие кнопки, щелчок мышью и др.) состоит в связывании события с методом, его обрабатывающим. Принцип обработки событий базируется на модели делегирования событий. В этой модели имеется блок прослушивания события *(EventListener)*, который ждет поступления события определенного типа от источника, после чего обрабатывает его и возвращает управление. Источник – это объект, который генерирует событие. Источником событий могут являться элементы управления: кнопки *(JButton,JCheckbox, JRadioButton),* списки, кнопки-меню. События могут генерироваться фреймами и апплетами, как mouse- и key-события. События генерируются окнами при развертке, сворачивании, выходе из окна.

Если изменяется его внутреннее состояние, например, изменился размер, изменилось значение поля, произведен щелчок мыши по форме или выбор значения из списка, после генерации объекта-события пересылается для обработки зарегистрированному в источнике блоку прослушивания как параметр его методов – обработчиков событий.

Блоки прослушивания *Listener* представляют собой объекты классов, реализующих интерфейсы прослушивания событий, определенных в пакетах *java.awt.event* и *javax.swing.event*. Соответствующие методы, объявленные в используемых интерфейсах, необходимо явно реализовать при создании собственных классов прослушивания. Эти методы и являются обработчиками события. Передаваемый источником блоку прослушивания объект-событие является аргументом обработчика события. Объект класса – блока прослушивания события необходимо зарегистрировать в источнике методом

источник.*add*Событие*Listener***(**объект\_прослушиватель**);**

После этого объект-слушатель (*Listener*)будет реагировать именно на данное событие и вызывать метод «обработчик события». Такая логика обработки событий позволяет легко отделить интерфейсную часть приложения от функциональной, что считается необходимым при проектировании современных приложений. Удалить слушателя определенного события можно с помощью метода *remove*Событие*Listener()*.

Когда событие происходит, все зарегистрированные блоки прослушивания уведомляются и принимают копию объекта события. Таким образом источник вызывает метод-обработчик события, определенный в классе, являющемся блоком прослушивания, и передает методу объект события в качестве параметра. В качестве блоков прослушивания на практике используются внутренние классы. В этом случае в методе, регистрирующем блок прослушивания в качестве параметра, используется объект этого внутреннего класса.

Каждый интерфейс, включаемый в блок прослушивания, наследуется от интерфейса *EventListener* и предназначен для обработки определенного типа событий. При этом он содержит один или несколько методов, которые всегда принимают объект события в качестве единственного параметра и вызываются в определенных ситуациях.

Таблица 1.2. События и слушатели, которые с ними работают

|  |  |
| --- | --- |
| Событие | Слушатель |
| **Библиотека java.awt.event** | |
| **ActionEvent**– генерируется: при нажатии кнопки; двойном щелчке клавишей мыши по элементам списка; при выборе пункта меню; | ActionListener |
| **AdjustmentEvent** – генерируется при изменении полосы прокрутки; | AdjustmentListener |
| **ComponentEvent** – генерируется, если компонент скрыт, перемещен, изменен в размере или становится видимым; | ComponentListener |
| **FocusEvent** – генерируется, если компонент получает или теряет фокус ввода; | FocusListener |
| **TextEvent**– генерируется при изменении текстового поля; | TextListener |
| **ItemEvent**– генерируется при выборе элемента из списка. | ItemListener |
| Класс **InputEvent**является абстрактным суперклассом событий ввода (для клавиатуры или мыши). | interface InputListener |
| **KeyEvent -** генерируется при вводе с клавиатуры | KeyListener |
| **MouseEvent -** генерируется, если произошло событие мыши | MouseListener |
| **Библиотека javax.swing.event** | |
| [**ChangeEvent**](http://docs.oracle.com/javase/7/docs/api/javax/swing/event/ChangeEvent.html) – генерируется при изменении состояния в источники события. | ChangeListener |
| **HyperlinkEvent** – генерируется, когда что-то изменилось относительно гипертекстовой ссылки. | HyperlinkListener |
| [**ListDataEvent**](http://docs.oracle.com/javase/7/docs/api/javax/swing/event/ListDataEvent.html) - определяет событие, которое инкапсулирует изменения в списке. | ListDataListener |
| **MenuEvent** – генерируется, когда меню, которое является источником события, было размещено, выбрано или отменено. | MenuListener |
| **MenuKeyEvent –** генерируется, когда элемент меню получил KeyEvent. | MenuKeyListener |
| **TableModelEvent –** генерируется при изменении модели таблицы. | TableModelListener |
| [**TreeModelEvent**](http://docs.oracle.com/javase/7/docs/api/javax/swing/event/TreeModelEvent.html) – генерируется при изменений TreeModel, уведомляет слушателей об изменении. | TreeModelListener |
| [**TreeSelectionEvent**](http://docs.oracle.com/javase/7/docs/api/javax/swing/event/TreeSelectionEvent.html) – событие, которое характеризует изменение текущего выбора. | TreeSelectionListener |

Как только пользователь нажимает кнопку, создается ActionEvent событие, которое передается слушателям кнопки. Для того, чтобы организовать слушателя, Swing предоставляет интерфейс ActionListener, который необходимо реализовать. Интерфейс ActionListener требует только реализации одного метода — actionPerformed. Пример класса, реализующего интерфейс ActionListener представлен ниже.

Пример. 8.4. Класс, реализующий интерфейс ActionListener

public class OurActionListener implements ActionListener {

public void actionPerformed(ActionEvent e) {

//Код, который нужно выполнить при нажатии

}

}

После того, как обработчик создан, его необходимо добавить к кнопке. Делается это при помощи метода addActionListener. В качестве параметра методу передается обработчик. Например, это можно сделать вот так:

JButton button = new JButton("Кнопка");

ActionListener actionListener = new OurActionListener();

button.addActionListener(actionListener);

Здесь мы создаем сначала кнопку. Потом создаем экземпляр нашего слушателя OurActionListener, а затем добавляем его в качестве слушателя к кнопке с помощью вызова addActionListener и передаем ему экземпляр обработчика.

Или же таким образом:

JButton button = new JButton("Кнопка");

button.addActionListener(new OurActionListener());

Вообще слушателей может быть неопределенное количество. Если нам будет необходимо два или более слушателей, которые должны будут по-разному реагировать на нажатие кнопки, то для каждого из них вызовем addActionListener.

Пример 8.5. Создать панель, на которой будут находиться 2 текстовых поля и одна кнопка. При нажатии на кнопку текст из первого текстового поля переместится во второе, а второе в первое, то есть поменяются местами.

public class User // класс, который содержит метод main()

{

public static void main(String[] args) {

Panel testPanel = new Panel();

testPanel.setVisible(true);

}

}

public class Panel extends JFrame{

JButton button;

JTextField text1, text2;

Panel(){

super("Пробное окно");

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setSize(500, 200);

button = new JButton("Поменять местами");

text1 = new JTextField();

text2 = new JTextField();

setLayout(null);

button.setSize(300, 50);

button.setLocation(90,100);

text1.setSize(200, 50);

text1.setLocation(20,20);

text2.setSize(200, 50);

text2.setLocation(250,20);

add(button);

add(text1);

add(text2);

button.addActionListener(new ButtonActionLisener());

}

public class ButtonActionLisener implements ActionListener{

public void actionPerformed(ActionEvent e) {

if(e.getSource() == button ){

String str1 = text1.getText();

String str2 = text2.getText();

text1.setText(str2);

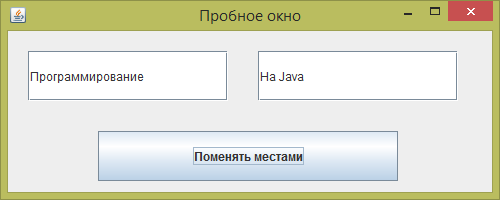
text2.setText(str1);

}

}

}

}



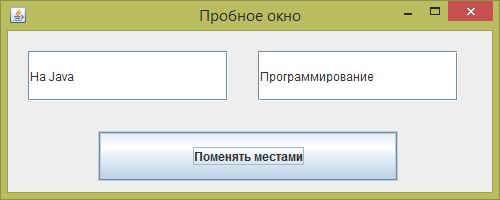


Рис. 8.10.

Таким же образом действие реализуются и для других элементов управления.

Пример 8.6. Создать панель, на которой будут находиться 2 радиокнопки и текстовое поле. При выборе первой из радиокнопок в текстовом поле высвечивается текст “Павел”, при выборе второй “Ольга”.

public class User // класс, который содержит метод main()

{

public static void main(String[] args) {

Panel testPanel = new Panel();

testPanel.setVisible(true);

}

}

public class Panel extends JFrame{

JRadioButton button1, button2;

JTextField text1;

ButtonGroup bg;

Panel(){

super("Пробное окно");

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setSize(500, 200);

button1 = new JRadioButton("Мужской");

button2 = new JRadioButton("Женский");

text1 = new JTextField();

bg = new ButtonGroup();

bg.add(button1);

bg.add(button2);

setLayout(null);

button1.setSize(100, 50);

button1.setLocation(50,20);

button2.setSize(100, 50);

button2.setLocation(200,20);

text1.setSize(200, 50);

text1.setLocation(50,80);

add(text1);

add(button1);

add(button2);

button1.addActionListener(new ButtonActionLisener());

button2.addActionListener(new ButtonActionLisener());

}

public class ButtonActionLisener implements ActionListener{

public void actionPerformed(ActionEvent e) {

{

if(e.getSource() == button1)

text1.setText("Павел");

if(e.getSource() == button2)

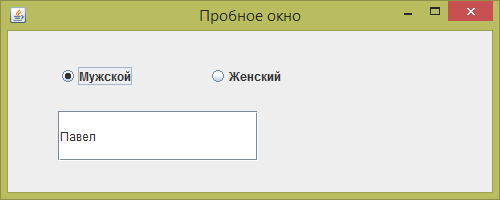
text1.setText("Ольга");

}

}

}

}



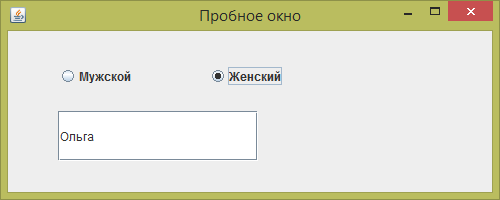


Рис. 8.11

Пример 8.7. Создать панель, на которой будут список и текстовое поле. Пример демонстрирует использование MouseListener. При выборе элемента из списка он появляется в текстовом поле, при вождении мышки по области демонстрируется использование методов MouseEvent.

public class User // класс, который содержит метод main()

{

public static void main(String[] args) {

Panel testPanel = new Panel();

testPanel.setVisible(true);

}

}

public class Panel extends JFrame{

Object box;

JList list;

JTextField text1;

String[] myList = {"Некрасова Ирина", "Лепель Андрей", "Иванов Николай", "Руденко Екатерина", "Некрасов Павел", "Мирницкая Анастасия"};

Panel(){

super("Пробное окно");

setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

setSize(500, 200);

list = new JList(myList);

text1 = new JTextField();

setLayout(null);

list.setSize(150, 110);

list.setLocation(10,10);

text1.setSize(200, 50);

text1.setLocation(200,50);

add(text1);

add(list);

list.addMouseListener(new ButtonActionLisener());

}

public class ButtonActionLisener implements MouseListener{

//  *Если пользователь нажал и отпустил одну из кнопок, вызывается метод mouseClicked.*

public void mouseClicked(MouseEvent e) {

list = (JList) e.getSource();

Object text = list.getSelectedValue();

text1.setText((String)text);

}

// *mouseEntered - данный метод будет вызываться системой у слушателя каждый раз, когда курсор мыши будет оказываться над компонентом.*

public void mouseEntered(MouseEvent e) {

list = (JList) e.getSource();

text1.setText("метод mouseEntered()");

}

// *mouseExited – данный метод срабатывает, когда убираем курсор мыши с компонента.*

public void mouseExited(MouseEvent e) {

list = (JList) e.getSource();

text1.setText("метод mouseExited()");

}

// *Навели на компонент, зажали кнопку — система вызвала mousePressed.*

public void mousePressed(MouseEvent e) {

list = (JList) e.getSource();

text1.setText("метод mousePressed()");

}

// *Отпускаем кнопку — система вызвала mouseReleased.*

public void mouseReleased(MouseEvent e) {

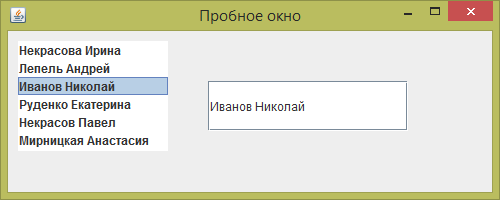
list = (JList) e.getSource();

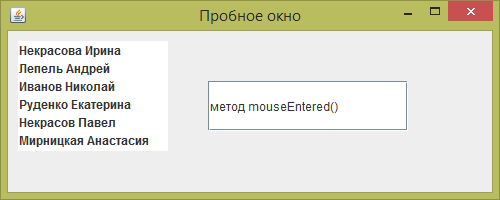
text1.setText("метод mouseReleased()");

}

}

}





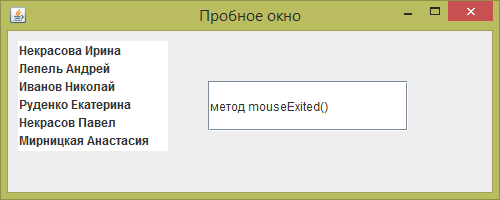


Рис. 8.12.

# **Внутренние классы**

В литературе по Java встречаются такие термины, как "внутренние классы" (inner classes) и "вложенные классы" (nested classes). Итак, вложенный класс - это класс, который объявлен внутри объявления другого класса. Например:

class OutClass{

…

class NewClass{

…

}

…

}

**Для чего используются внутренние классы**

* Это хороший способ группировки классов, которые используются только в одном месте: если класс полезен только для одного другого класса, то логично будет держать их вместе. Вложение таких вспомогательных классов делает код более удобным.
* Инкапсуляция: допустим, есть два класса A и B, классу B требуется доступ к свойству класса A, которое может быть приватным. Вложение класса B в класс A решит эту проблему, более того сам класс B можно скрыть от внешнего использования.
* Улучшение читаемости и обслуживаемости кода: вложение малых классов в более высокоуровневые классы позволяет хранить код там, где он используется.

## **Типы внутренних классов**

*1.Статические внутренние классы.*

Декларируются внутри основного класса и обозначаются ключевым словом static. Не имеют доступа к членам внешнего класса за исключением статических. Может содержать статические поля, методы и классы, в отличие от других типов внутренних классов.

class OutClass{

…

static class NewClass{

…

}

…

}

*2.Внутренние классы.*

Декларируются внутри основного класса. В отличие от статических внутренних классов, имеют доступ к членам внешнего класса. Не могут содержать (но могут наследовать) определение статических полей, методов и классов (кроме констант).

class OutClass{

…

class NewClass{

…

}

…

}

*3.Локальные классы.*

Декларируются внутри методов основного класса. Могут быть использованы только внутри этих методов. Имеют доступ к членам внешнего класса. Имеют доступ как к локальным переменным, так и к параметрам метода при одном условии - переменные и параметры используемые локальным классом должны быть задекларированы final. Не могут содержать определение (но могут наследовать) статических полей, методов и классов (кроме констант).

class OutClass{

void methodWithLocalClass(){

class NewClass{

…

}

}

…

}

*4. Анонимные классы.*

Декларируются внутри методов основного класса. Могут быть использованы только внутри этих методов. В отличие от локальных классов, анонимные классы не имеют названия. Главное требование к анонимному классу - он должен наследовать существующий класс или реализовывать существующий интерфейс. Не могут содержать определение (но могут наследовать) статических полей, методов и классов (кроме констант).

class OutClass{

void methodWithLocalClass()

{

*/\* При определении анонимного класса применен полиморфизм – переменная listener содержит экземпляр анонимного класса, реализующего существующий интерфейс ActionListener \*/*

ActionListener listener = new ActionListener()

{

public void actionPerformed (ActionEvent event)

{

}

};

}

# **Пример выполнения индивидуального задания**

Пример 8.8. Пример выполнения индивидуального задания

Реализовать приложение, в котором будет осуществляться регистрация, поступающих в аптеку новых медикаментов. Введённые данные сохранять в файл. Реализовать механизм закрытия окна.

import java.awt.event.\*;

import java.io.\*;

import javax.swing.\*;

public class User {

public static void main(String[] args) {

Reader med = new Reader("Регистрация медикаментов");

med.setVisible(true);

med.setResizable(false);

med.setLocationRelativeTo(null);

}

}

public class Reader extends JFrame {

String text1;

int k;

Object boxA, boxB, boxC;

File file = new File("D://GUI.txt");

static JLabel *l1*, *l2*, *l3*, *l4*;

JComboBox box\_1, box\_2, box\_3;

JRadioButton flag1, flag2;

ButtonGroup bg;

static JButton *b*, *del*;

static JTextField *text*;

static JTextArea *area*;

static String[] *box1* = {"1","2","3","4","5","6","7","8","9","10","11","12","13","14","15","16","17","18","19","20","21","22","23","24","25","26","27","28","29","30","31"};

static String[] *box2* = {"январь","февраль","март","апрель","май","июнь","июль","август","сентябрь","октябрь","ноябрь","декабрь"};

static String[] *box3* = {"2008","2009","2010","2011","2012","2013","2014","2015","2016","2017","2018","2019","2020"};

public Reader(String str){

super(str);

setSize(500, 500);

setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

*b* = new JButton("Внесение в базу");

*del* = new JButton("Очистить");

*text* = new JTextField(9);

*area* = new JTextArea(9, 9);

*l1* = new JLabel("Название медикамента");

*l2* = new JLabel("Описание лекарственных свойств");

*l3* = new JLabel("Дата поступления");

*l4* = new JLabel("Отпуск без рецепта");

box\_1 = new JComboBox(*box1*);

box\_2 = new JComboBox(*box2*);

box\_3 = new JComboBox<Object>(*box3*);

flag1 = new JRadioButton("да");

flag2 = new JRadioButton("нет");

bg = new ButtonGroup();

bg.add(flag1);

bg.add(flag2);

setLayout(null);

*b*.setSize(200,30);

*b*.setLocation(150, 400);

*del*.setSize(100, 30);

*del*.setLocation(10, 400);

*text*.setSize(220,25);

*text*.setLocation(250, 20);

*area*.setSize(220,150);

*area*.setLocation(250,60);

*l1*.setSize(200,25);

*l1*.setLocation(30, 20);

*l2*.setSize(220, 25);

*l2*.setLocation(30, 60);

*l3*.setSize(200, 25);

*l3*.setLocation(30, 250);

*l4*.setSize(200, 25);

*l4*.setLocation(30, 310);

box\_1.setSize(40,25);

box\_1.setLocation(250,250);

box\_2.setSize(100, 25);

box\_2.setLocation(300, 250);

box\_3.setSize(70, 25);

box\_3.setLocation(410, 250);

flag1.setSize(40,25);

flag1.setLocation(250, 310);

flag2.setSize(50, 25);

flag2.setLocation(300, 310);

add(*b*);

add(*del*);

add(*text*);

add(*area*);

add(*l1*);

add(*l2*);

add(*l3*);

add(*l4*);

add(box\_1);

add(box\_2);

add(box\_3);

add(flag1);

add(flag2);

*b*.addActionListener(new ButtonActionListener());

flag1.addActionListener(new FlagActionListener());

flag2.addActionListener(new FlagActionListener());

*del*.addActionListener(new DelActionListener());

box\_1.addActionListener(new BoxActionListener());

box\_2.addActionListener(new BoxActionListener());

box\_3.addActionListener(new BoxActionListener());

}

public class ButtonActionListener implements ActionListener {

public void actionPerformed(ActionEvent e) {

try{

if(!file.exists()){

file.createNewFile();

}

FileWriter out = new FileWriter(file, true);

try{

String text1 = *text*.getText();

String area2 = *area*.getText();

out.write(text1 + " - ");

out.write(area2 + "\n");

if(k == 1)

out.write("Отпуск медикамента осуществляется без рецепта. " + "\n");

else if(k == -1)

out.write("Отпуск медикамента осуществляется при наличии рецепта. " + "\n");

out.write(" (Дата поступления:" + boxA + " " + boxB + " " + boxC + ")");

} finally{

out.close();

}

}catch(IOException e1){

throw new RuntimeException(e1);

}

}

}

public class FlagActionListener implements ActionListener {

public void actionPerformed(ActionEvent e) {

k = 0;

if(e.getSource() == flag1){

k++;

}

else if(e.getSource() == flag2){

k--;

}

}

}

public class DelActionListener implements ActionListener{

public void actionPerformed(ActionEvent e) {

if(e.getSource() == *del*){

*text*.setText(null);

*area*.setText(null);

}

}

}

public class BoxActionListener implements ActionListener{

public void actionPerformed(ActionEvent e) {

if(e.getSource() == box\_1){

boxA = box\_1.getSelectedItem();

}

if(e.getSource() == box\_2){

boxB = box\_2.getSelectedItem();

}

if(e.getSource() == box\_3){

boxC = box\_3.getSelectedItem();

}

}

}

}

# **Задания для самостоятельного выполнения**

Разработать приложение с 5 (минимум) элементами управления библиотеки Swing (кнопка (*JButton*), список (*JList*), текстовое поле (*JTextField*), раскрывающий список (*JComboBox*), текстовая область (*JTextArea*), флажок (*JCheckBox*), радиокнопки (*JRadioButton*), лэйбл (*JLabel*)) исходя из задания вариантов. Подгрузку данных осуществлять из файла (при необходимости), сохранять данные также в файл. Обязательна реализация механизма закрытия окна. Также можно использовать систему подсветки данных, при желании.

1. Регистрация пластиковой карточки.
2. Регистрация поступлений в магазин игрушек (учесть дату поступления).
3. Регистрация пользователя в сети.
4. Заполнение анкеты.
5. Оформление заказа в интернет магазине.
6. Бронирование тура.
7. Оформление книги в электронной библиотеке.
8. Создание мини библиотеки автобиографий.
9. Опрос на конкретную тематику (Машины, мебели………).
10. Регистрация поступлений в продуктовый магазин (учесть дату поступления).
11. Регистрация посетителей салона по оказанию каких-то услуг.
12. Реализовать электронное меню для ресторана.
13. Создание архива бумаг\книг, занесение в базу данных о них.
14. Система бронирование авиабилета.
15. Бронирование талона в больницу с указанием причины посещения (обследование, болезнь…….), необходимые врачи для посещения и т.д.

# **Контрольные вопросы**

1. Какие существуют в Java библиотеки визуальных компонентов для создания графического интерфейса пользователя?
2. Какие основные элементы управления библиотеки Swing вам известны?
3. Какое действие выполняет конструктор JFrame()?
4. Что такое менеджер размещения и для чего он служит?
5. При помощи каких методов задаются координаты и размер элементов?
6. Что такое *“источник”* события, и что им может являться?
7. Что такое ActionListener, и как метод он требует для реализации?
8. При помощи какого метода обработчик добавляется к кнопке?
9. Для чего нужны внутренние классы?
10. Какие типы внутренних классов вам известны?