ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
КЕМЕРОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Кафедра ЮНЕСКО по новым информационным технологиям

С.Н. Карабцев

ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ НА JAVA

Лабораторный практикум

Математический факультет

Специальность 010503 – математическое обеспечение и администрирование информационных систем

Кемерово, 2009

ЛАБОРАТОРНАЯ РАБОТА №6.
ПАКЕТ javax.swing. ГРАФИЧЕСКИЙ ИНТЕРФЕЙС

1. Цель работы

Целью работы является ознакомление со средствами построения графического интерфейса пользователя GUI с помощью компонент пакетов java.awt и javax.swing.

2. Методические указания

Лабораторная работа направлена на приобретение основных понятий и навыков работы с пакетами java.awt и javax.swing и их компонентами для построения графического интерфейса пользователя. В работе рассматриваются вопросы создания многооконных графических инетрфейсов.

Требования к результатам выполнения лабораторного практикума:

- при выполнении задания необходимо сопровождать все реализованные процедуры и функции набором тестовых входных и выходных данных и описаниями к ним;
- реализацию программы можно осуществлять как из интегрированной среды разработки Eclipse, так и в блокноте с применением вызовов функций командной строки;
- по завершении выполнения задания составить отчет о проделанной работе.

При составлении и оформлении отчета следует придерживаться рекомендаций, представленных на следующей странице: http://unesco.kemsu.ru/student/rule/rule.html.

3. Теоретический материал

Для создания графического интерфейса пользователя используются классы пакетов java.awt и javax.swing.

Создание элементов пользовательского интерфейса библиотека AWT (Abstract Window Toolkit) поручала встроенным инструментальным средствам. Если нужно вывести окно с текстом, то фактически оно отображалось с помощью платформенно-ориентированных средств. Теоретически такие программы должны работать на любых платформах, имея внешний вид характерный для платформы. Однако с помощью AWT трудно создавать переносимые графические библиотеки, зависящие от родных интерфейсных элементов платформы (например, меню и скроллинг на разных платформах могут вести себя по-разному).

Абстрактный Оконный Инструментарий Java 1.0 вводил GUI, который выглядел достаточно заурядно на всех платформах. Кроме того, он был ограничен: можно было использовать только четыре шрифта и было невозможно получить доступ к любому более сложному и тонкому GUI элементу, имеющемуся в ОС. Модель программирования Java 1.0 AWT также была слабая и не объектно-ориентированная.

В 1996 году Netscape создала библиотеку программ для создания GUI и назвала ее Internet Foundation Class (IFC). Элементы пользовательского интерфейса рисовались в пустом окне. Единственно, что требовалось от оконной системы платформы, - отображение окна и рисование в нем. Таким образом, элементы GUI выглядели и вели себя одинаково, но не зависели от платформы, на которой запускались. Компании Sun и Netscape объединили свои усилия и создали библиотеку Swing. Преимущества Swing:

- содержит более богатый и удобный набор элементов пользовательского интерфейса;
- библиотека намного меньше зависит от платформы (меньше ошибок);
- обеспечивает пользователям однотипные средства для работы на разных платформах

Создание фрейма

Окно верхнего уровня в языке ява называется фреймом. В библиотеке awt для такого окна предусмотрен класс Frame. В библиотеке Swing для такого окна предусмотрен класс JFrame (аналог). Класс JFrame расширяет класс Frame и представляет собой один из немногих компонентов в библиотеке Swing, которые не отображаются на холсте (canvas). Кнопки, строка заголовков, пиктограммы и другие компоненты реализуются с помощью пользовательской оконной системы, а не библиотекой Swing.

Рассмотрим самые распространенные приемы работы с классом JFrame.

```
import javax.swing.*;
public class SimpleFrameTest{
  public static void main(String[] args){
   simple frame=new simple();  // создание экземпляра фрейма
// здесь можно создать несколько фреймов или включить их в описание
// класса SimpleFrameTest как поля

   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true); //делает фрейм видимым }
}

/* в данном классе можно реализовывать различную функциональность окна.
Например, разместить кнопки, списки, меню.
*/
class simple extends JFrame{
  public simple(){
    setSize(200,300); // задание размера окна}
}
```

По умолчанию фрейм имеет размер 0x0 пикселей. Для задания размера фрейма используется метод **setSize**(int x, int y). Метод **setDefaultCloseOperation** определяет поведение фрейма при закрытии окна. Задание константы JFrame.EXIT_ON_CLOSE определяет закрытие приложения при закрытии фрейма.

Для задания позиционирования фрейма используется метод **setLocation** (int x, int y), где x и y – координаты левого верхнего угла фрейма.

Разработчики Java для запуска приложения с фреймом рекомендуют использовать диспетчер событий. Тогда предыдущий пример будет выглядеть следующим образом.

```
Import javax.swing.*;
public class SimpleFrameTest{
  public static void main(String[] args){
  EventQueue.invokeLater (new Runnable() {
    public void run()
    {
      simple frame=new simple();  // создание экземпляра фрейма
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true); //делает фрейм видимым
   }
  });
}}

class simple extends JFrame{
  public simple(){
    setSize(200,300); // задание размера окна}
}
```

Метод **setIconImage**() сообщает оконной системе о том, какая пиктограмма должна отображаться в строке заголовка.

Метод **setTitle**() позволяет изменить текст в окне заголовка.

Метод **setResizible**(), получающий в качестве параметра логическое значение и определяющий, имеет ли пользователь право изменять размеры фрейма.

Определение подходящего размера фрейма можно выполнить с помощью методов класса Toolkit.

```
import javax.swing.*;
public class SimpleFrameTest{
  public static void main(String[] args){
  EventQueue.invokeLater (new Runnable() {
    public void run()
    {
      SizedFrame frame=new SizedFrame();  // создание экземпляра фрейма
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true); //делает фрейм видимым
```

```
   }
  });
}}

class SizedFrame extends JFrame{
  public SizedFrame(){
  //класс взаимодействия с оконной системой ОС
  Toolkit kit = Toolkit.getDefaultToolkit();
  Dimension screens = kit.getScreenSize();
  int w,h;
  w = screens.width;
  h = screens.height;
  setSize(w/2,h/2);
  setLocation(w/4, h/4);
  setTitle("My Frame");
  Image img = kit.getImage(“Icon.gif”);
  setIconImage(img);
  }
}
```

Многооконные приложения

Как известно, окно в современных графических операционных системах используются для разделения задач, выполняемых пользователем, в одном из окон он может редактировать текст, в другом окне просматривать фотографии. Использование окон позволяет ему временно забыть про остальные свои задачи, не смешивая их, заниматься основным делом (в терминах оконной системы это будет активное в данный момент окно), и, как только возникает необходимость, переместиться к другой задаче (сделав активным другое окно, то есть вытащив его на первый план и временно убрав на задний план все остальное).

Как только концепция окон была изобретена и довольно успешно начала справляться с разделением задач пользователя на основную и второстепенные, возникло желание продвинуться немного дальше и применить ее уже внутри одного приложение, то есть разделить задачу внутри окна на несколько задач используя все те же окна, но на этот раз уже внутри другого, главного, окна. Подобные интерфейсы и называются

многодокументными. Приложения, берущие на вооружение такие интерфейсы, как правило, способны выполнять задачи такого широкого спектра, что уместить все в одно окно становится затруднительно без ущемления удобства пользователя. На помощь приходят дополнительные окна, в которых можно разместить информацию, которая имеет косвенное отношение к основной, выполняемой в данный момент, задаче.

Применение многодокументных интерфейсов на практике вызывает жаркие споры. Сам по себе интерфейс операционной системы является многодокументным приложением, и встраивание в него еще одного подобного интерфейса может запутывать пользователя. Окна операционной системы работают по своему, окна приложения по своему, и все это не прибавляет уверенности пользователю в том, что происходит и как этим следует управлять. Поэтому многие развитые многодокументные приложения постарались уменьшить количество и разнообразие дополнительных окон, чтобы уменьшить сложность для пользователя. Тем не менее, как показывает практика, полностью избавиться от дополнительных окон в сложных приложениях затруднительно, так что мы все-таки изучим возможности Swing в данном вопросе.

Создание внутренних окон

Внутренние окна в Swing реализованы классом **JInternalFrame**. Данный класс унаследован от общего предка всех компонентов Swing JComponent, и, таким образом, представляет собой обычный легковесный компонент, который вы при желании можете добавить в любую панель или окно вашего приложения. Так как данный компонент призван эмулировать окно внутри приложения, у него есть элементы управления, такие как меню окна, кнопки для закрытия, развертывания и свертывания, а также заголовок и рамка. Роль рабочего стола, на котором окна размещаются, внутри которого перемещаются и занимают его пространство, исполняет компонент **JDesktopPane**.

```
import javax.swing.*;

public class SimpleMDI extends JFrame {
   public SimpleMDI() {
      super("SimpleMDI");
      setSize(400, 300);
      setDefaultCloseOperation(EXIT_ON_CLOSE);
      // создаем рабочий стол Swing
      JDesktopPane desktopPane = new JDesktopPane();
      // добавляем его в центр окна
      add(desktopPane);
      // создаем несколько внутренних окон, применяя доступные
конструкторы
      JInternalFrame frame1 = new JInternalFrame("Frame1", true);
      JInternalFrame frame2 = new JInternalFrame(
"Frame2", true, true, true, true);
      // добавляем внутренние окна на рабочий стол
      desktopPane.add(frame1);
      desktopPane.add(frame2);
      // задаем размеры и расположения, делаем окна видимыми
      frame1.setSize(200, 100);
      frame1.setLocation(80, 100);
      frame1.setVisible(true);
      frame2.setSize(200, 60);
      frame2.setVisible(true);
      // выводим окно на экран
      setVisible(true);
   }

   public static void main(String[] args) {
      new SimpleMDI();
   }
}
```

В обычном окне JFrame размещается рабочий стол JDesktopPane. Как видно, никаких параметров конструктору не требуется, так же как не требуется для нормальной работы и дополнительной настройки. О рабочем столе можно думать как о специализированной панели, специально предназначенной для размещения и управления внутренними окнами. Добавляем его в центр окна методом **add**() напрямую.

Далее создаются внутренние окна. Интересно, что класс JInternalFrame обладает целым набором конструкторов с различным набором параметров, которые позволяют задать все атрибуты внутренних окон. Обязательно присутствует лишь заголовок окна (как первый параметр всех конструкторов).

Первое окно создается с заголовком и изменяемого размера, за что отвечает второй параметр конструктора. По умолчанию, если не указывать **true**, внутренние окна считаются неизменяемого размера, без возможности сворачивания, развертывания, более того, их даже невозможно закрыть. Все эти возможности включаются либо передачей параметров в конструктор, либо меняются через соответствующие свойства.

Второе окно создается вызовом наиболее развернутого конструктора. Здесь мы указываем заголовок окна, а также делаем его размер изменяемым, для самого окна включаем поддержку развертывания, свертывания и способность закрываться по кнопке закрытия окна. Довольно своеобразное решение создателей не включать все эти свойства по умолчанию, и передавать их в довольно длинный конструктор.

Внутренние окна стараются во всем походить на своих старших братьев – окна высокого уровня, унаследованные от Window, и поэтому они по умолчанию невидимы, имеют нулевой размер и располагаются в начале экрана. Работа с внутренними окнами здесь не отличается от работы с обычными. Мы меняем размер окон, устанавливаем их позиции и делаем их видимыми. Только после этого они появятся на рабочем столе JDesktopPane.

Запустив приложение, вы увидите рабочий стол и располагающиеся на нем внутренние окна. Отличие от обычной панели лишь в том, что мы использовали для расположения окон все те же методы, что для окон, и в том, что вы можете таскать окна, сворачивать их и менять их размеры, если, конечно, они вам это позволяют. В дополнение ко всему, рабочий стол позаботится о том, чтобы активное окно перекрывало все остальные и было на переднем плане.

На этом можно считать многодокументное приложение Swing готовым. Вы можете конструировать свои интерфейсы и добавлять их во внутренние окна, располагая их, так как вас удобно. Пользователь будет в состоянии переключать свое внимание между окнами, концентрируясь на определенном аспекте приложения. Пример создания многодокументного приложения с компонентом **меню** приведено в **Examples/internalframedemo.java**.

Создание двух и более окон в приложении

При разработке интерфейса можно не создавать внутренние окна, а воспользоваться классом JFrame. В примере, показанном ниже, создается основное окно приложения, на котором размещена кнопка. Данное окно является экземпляром класса simple1. Нажатие кнопки but на фрейме frame (класса simple1 в методе main) ведет к вызову конструктора для создания второго фрейма класса simple2. Обратите внимание, что экземпляр класса simple2 включен как поле в класс simple1. Это поле называется s2. При таком подходе вся информация, доступная на фрейме s2, будет доступна и основному классу. Это демонстрируется путем вызова метода getV(), который распечатывает на экран значение переменной r, являющейся полем в классе simple2.

```
//фрейм посередине окна
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class t2 {
  public static void main(String[] args){
    simple1 frame=new simple1();
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.show();}
}

class simple1 extends JFrame{
simple2 s2;
JButton but=new JButton("Create");
  public simple1(){
```

```
  ButtonListener blistener = new ButtonListener();

  Toolkit kit = Toolkit.getDefaultToolkit();
  Dimension screens = kit.getScreenSize();
  int w,h;
  w = screens.width;
  h = screens.height;
  setSize(w/2,h/2);
  setLocation(w/4, h/4);
  setTitle("My Frame");
  setLayout(new FlowLayout());
  setLayout(null);  //отключает менеджер расположения компонентов
  add(but);
  but.addActionListener(blistener);
}

public void getV(){
System.out.println("VALUES+++"+s2.r);
}

class ButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
    if(s2==null){s2=new simple2(); }
        s2.show();
        getV();                                          }
                                                           }
}//class

class simple2 extends JFrame{
int r=5;

  public simple2(){
  Toolkit kit = Toolkit.getDefaultToolkit();
  Dimension screens = kit.getScreenSize();
  int w,h;
  w = screens.width;
  h = screens.height;
  setSize(w/3,h/3);
  setLocation(w/6, h/6);
  setTitle("My Frame");
}
}
```

Управление компоновкой

Все компоненты, с которыми мы работали до сих пор, размещались "вручную" вызовом метода add() и установкой позиции с помощью метода setLocation() или setBounds(). Вызов метода setLayout(null) запрещал использование предусмотренного по умолчанию механизма управления размещением компонентов. В языке Java реализована концепция динамической компоновки - все компоненты контейнера управляются менеджером компоновки (layout manager).

Существуют следующие виды менеджеров компоновки – потоковая компоновка (flow layout manager), компоновка рамок (border layout), сеточная компоновка (grid layout) и другие более сложные менеджеры.

Для установки компоненту менеджера компоновки используется метод **setLayout**(LayoutManager m). Данный метод необходимо вызывать в объектах классов JApplet, Applet, JFrame, Frame, JPanel и других контейнерах перед размещением на них графических компонентов построения интерфейса, таких как кнопки (JButton, Button), варианты выбора (JCheckBox), текстовые поля (TextArea, JTextArea) и др.

В качестве аргумента при вызове метода **setLayout**(LayoutManager m) указывается конкретный экземпляр класса **m** менеджера компоновки.

Менеджер FlowLayout

Класс FlowLayout реализует простой стиль размещения, при котором компоненты располагаются, начиная с левого верхнего угла, слева направо и сверху вниз. Если в данную строку не помещается очередной компонент, он располагается в левой позиции новой строки. Справа, слева, сверху и снизу компоненты отделяются друг от друга небольшими промежутками. Ширину этого промежутка можно задать в конструкторе FlowLayout. Каждая строка с компонентами выравнивается по левому или правому краю, либо центрируется в зависимости от того, какая из констант LEFT, RIGHT или

CENTER была передана конструктору. Режим выравнивания по умолчанию - CENTER, используемая по умолчанию ширина промежутка - 5 пикселей.

Конструкторы:

FlowLayout();

FlowLayout(int align);

FlowLayout(int align, int hgap, int vgap);

Примеры программ с использованием FlowLayout приведены в папке Examples/ex1 и ex2.

Менеджер BorderLayout

Класс BorderLayout реализует обычный стиль размещения для окон верхнего уровня, в котором предусмотрено четыре узких компонента фиксированной ширины по краям, и одна большая область в центре, которая может расширяться и сужаться в двух направлениях, занимая все свободное пространство окна. У каждой из этих областей есть строки-имена: String.North, String.South, String.East и String.West соответствуют четырем краям, а Center - центральной области.

Для вызова данного менеджера необходимо использовать следующую конструкцию

```
setLayout(new BorderLayout());
add(yellowButton, BorderLayout.SOUTH);
```

Пример программы с использованием BorderLayout приведен в папке Examples/ex3.

Менеджер GridLayout

Класс GridLayout размещает компоненты в простой равномерной сетке. Конструктор этого класса позволяет задавать количество строк и столбцов. Например, setLayout(new GridLayout(5,4)) – указывает желаемое количество строк (5) и столбцов (4). А setLayout(new GridLayout(5,4,3,3)) – указывает

желаемое количество строк (5) и столбцов (4) и расстояние между компонентами по горизонтали (3) и вертикали (3).

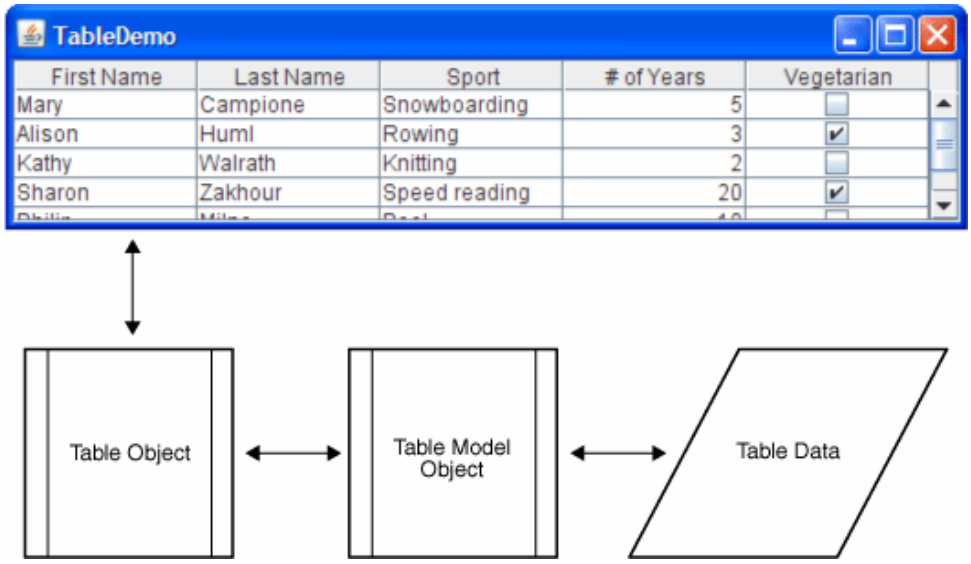
Класс JTable

Класс javax.swing.JTable используется для отображения и редактирования регулярных плоских таблиц. Класс JTable содержит следующие конструкторы:

- public JTable() – создание таблицы по умолчанию, со столбцами и строками по умолчанию.
- public JTable(TableModel dm) – столбцы и строки таблицы инициализируются моделью данных dm.
- public JTable(int numRows, int numColumns) – создание пустой таблицы размером (numRows, numColumns), используя DefaultTableModel. Название столбцов – A, B, C…
- JTable(Object[][] rowData, Object[] columnNames), где rowData – массив данных по строкам, а columnNames – массив с названиями столбцов.
- JTable(TableModel dm, TableColumnModel cm).
- JTable(TableModel dm, TableColumnModel cm, ListSelectionModel sm).
- public JTable(Vector rowData, Vector columnNames) – создает таблицу и показывает в ней данные из rowdata, которые есть Vector of Vectors. Название столбцов – вектор columnNames. ((Vector)rowData.elementAt(1)).elementAt(5); - доступ к элементу (1,5)

Для создания таблицы, которая может отображать ***актуальные*** данные, заголовки столбцов (не дефолтные) и позволяет выполнять над собой определенный набор действий (перетаскивание столбцов, выделение и форматирование ячеек и т.д.) необходимо реализовывать интерфейс **TableModel** (табличная модель) и передавать объект класса, реализующего

интерфейс табличной модели, в конструктор. На рисунке ниже показана взаимосвязь между объектом Таблица (Table Object), объектом, реализующим интерфейс табличной модели (Table Model Object) и данными, которые отображаются в таблице (Table data).



Если при создании таблицы JTable явно не реализуется интерфейс **TableModel,** то в таблицу передается объект интерфейса по умолчанию DefaultTableModel. Чтобы создать таблицу, подчиняющуюся табличной модели, необходимо:

```
TableModel myData = new MyTableModel();
JTable table = new JTable(myData);
```

При этом класс MyTableModel должен реализовывать интерфейс (т.е. перекрывать все его методы) TableModel.

Интерфейс **TableModel** содержит большое количество методов, которые управляют поведением таблицы. Поэтому на практике стараются реализовать некоторый класс-потомок интерфейса TableModel (чтобы не все методы интерфейса перекрывать). Например,

public abstract class ***AbstractTableModel*** extends Object implements ***TableModel***, Serializable

Данный абстрактный класс обеспечивает встроенную (по умолчанию) реализацию большинства методов интерфейса TableModel. Для создания табличной модели данных необходимо наследовать подкласс от этого класса

и обязательно реализовать следующие методы интерфейса TableModel: getRowCount(), getColumnCount() и getValueAt(int rowIndex, int columnIndex). Например, чтобы создать таблицу размером 10 на 10, которая хранит значение таблицы умножения необходимо:

```
TableModel dataModel = new AbstractTableModel() {
        public int getColumnCount() { return 10; } // перекрыли метод, он всегда
                                                   //возвращает 10 столбцов
        public int getRowCount() { return 10;} // всегда возвращает 20 строк

//сами значения, которые помещаются в таблицу! Обратите внимание, что
//метод называется getValueAt, а не setValueAt
        public Object getValueAt(int row, int col) { return new Integer(row*col); }
    };

    JTable table = new JTable(dataModel);
    JScrollPane scrollpane = new JScrollPane(table);
```

Тот же самый пример можно написать более явно (но длинно) следующим образом:

```
class MyTableModel extends AbstractTableModel {
        public int getColumnCount() { return 10; } // перекрыли метод, он всегда
                                                   //возвращает 10 столбцов
        public int getRowCount() { return 10;} // всегда возвращает 20 строк

//сами значения, которые помещаются в таблицу! Обратите внимание, что
//метод называется getValueAt, а не setValueAt
        public Object getValueAt(int row, int col) { return new Integer(row*col); }
    }

   MyTableModel dataModel = new MyTableModel();
    JTable table = new JTable(dataModel);
    JScrollPane scrollpane = new JScrollPane(table);
```

Вообще говоря, интерфейс TableModel содержит следующие методы:

- public int ***getRowCount***() – возвращает количество строк в табличной модели.

– public int ***getColumnCount***() – возвращает количество столбцов в табличной модели.
– public String ***getColumnName***(int columnIndex) – возвращает название столбца в модели.
– public Object ***getValueAt***(int rowIndex, int columnIndex) – возвращает значение в ячейке с номером (rowIndex, columnIndex).
– public void ***setValueAt***(Object aValue, int rowIndex, int columnIndex) – устанавливает значение aValue в ячейку с номером (rowIndex, columnIndex).

Более полную информацию с примерами о классе JTable можно найти на сайте создателя http://java.sun.com/docs/books/tutorial/uiswing/components/table.html, а также в статье в папке **examples/jtable.rar**.

Информацию о графических компонентах построения интерфейсов GUI можно найти в папке examples/prjava.rar и **examples/lec11.pdf**.

4. Порядок выполнения работы

– изучить предлагаемый теоретический материал;
– реализуйте в виде программ на языке Java следующие задачи:

1. Создайте Frame с компонентом меню Файл -> Создать. При нажатии на элемент меню «Создать» создается новое окно, на котором размещен компонент JRadioButton и кнопка JButton. Компонент JRadioButton состоит из 2 элементов – «отображать таблицу умножения» и «отображать таблицу сложения». При выборе конкретного пункта (умножения или сложения) и нажатии кнопки

JButton окно закрывается, а на основном фрейме появляется таблица умножения или сложения соответственно.

2. Окно, которое появляется при нажатии на «Файл -> Создать» можно создать по своему усмотрению одним из следующих двух способов:
   - с помощью классов JDesktopFrame и JInternalFrame (за основу можно взять файл **Examples/internalframedemo.java**).
   - с помощью классов Frame.

5. Содержание отчета

В отчете следует указать:

1. цель работы;
2. введение;
3. программно-аппаратные средства, используемые при выполнении работы;
4. основную часть (описание самой работы), выполненную согласно требованиям к результатам выполнения лабораторного практикума;
5. заключение (описание результатов и выводы);
6. список используемой литературы.

6. Литература

1. Вязовик, Н.А. Программирование на Java. [электронный ресурс] http://www.intuit.ru/department/pl/javapl (8.11.2009).
2. Хорстманн К.С., Корнелл Г. Библиотека профессионала. JAVA 2. Том 1. Основы. 8-е издание. Пер. с англ. – М.: ООО Издательский дом “Вильямс”, 2008. – 816 с.
3. Эккель Б. Философия JAVA. – СПб.: Питер, 2003. – 971с.