Конспект лекций по Java. Занятие 12

В.Фесюнов

1. Разбор домашнего задания

Файл Person.java

```
// Person.java

import java.io.*;

public class Person implements Comparable {

    private String name;
    private String phone;

    Person(String name, String phone) {
        this.name = name;
        this.phone = phone;
    }

    Person() {
        this(null, null);
    }

    String getName() {
        return name;
    }

    String getPhone() {
        return phone;
    }
```

```
    public int compareTo(Object another) {
        return name.compareTo(((Person)another).name);
    }

    public String toString() {
        return ("\nФамилия: "+name+"\nТелефоны:
"+phone+"\n");
    }

}
```

Файл PhoneNotes.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;
import java.io.UnsupportedEncodingException;

public class PhoneNotes extends JFrame {

    JTextField fldFio = new JTextField(25);
    JTextField fldPhone = new JTextField(25);
    JTextField fldCnt = new JTextField(4);
    TreeSet set = new TreeSet();

    public PhoneNotes() {
     super("Записная книжка");

     try  {
       UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
     }
     catch(Exception e) {
     }

    setSize(400, 250);
    Container c = getContentPane();
// Центральная панель
    JPanel centerPanel = new JPanel(new GridLayout(3, 1, 5, 5));
```

```
    centerPanel.setBorder(BorderFactory.createEtchedBorder());
    JPanel aPanel = new JPanel();
    JLabel aLabel = new JLabel("Фамилия ");
    aPanel.add(aLabel);
    aPanel.add(fldFio);
    centerPanel.add(aPanel);
    aPanel = new JPanel();
    aLabel = new JLabel("Телефон ");
    aPanel.add(aLabel);
    aPanel.add(fldPhone);
    centerPanel.add(aPanel);
    aPanel = new JPanel();
    JButton btn = new JButton("Печатать");
    aPanel.add(btn);
    centerPanel.add(aPanel);
    c.add(centerPanel, BorderLayout.CENTER);
// Нижняя панель
    JPanel statusPanel = new JPanel();
    statusPanel.setBorder(BorderFactory.createEtchedBorder());
    aLabel = new JLabel("Количество записей ");
    statusPanel.add(aLabel);
    fldCnt.setEnabled(false);
    statusPanel.add(fldCnt);
    c.add(statusPanel, BorderLayout.SOUTH);
// Listener'ы полей и кнопок
    fldPhone.addActionListener(new ActionListener(){
          public void actionPerformed(ActionEvent e) {
              Person prs = new Person(fldFio.getText(), fldPhone.getText());
              fldFio.setText("");
              fldPhone.setText("");
              set.add(prs);
              fldCnt.setText(""+set.size());
              fldFio.requestFocus();
          }
        });
    btn.addActionListener(new ActionListener(){
          public void actionPerformed(ActionEvent e) {
              Iterator iter = set.iterator();
              for ( int i = 0; iter.hasNext(); i++ ) {
                   Person cur = (Person)iter.next();
```

```
                    String str = cur.toString();
                    try {
                        byte[] b = str.getBytes("Cp866");
                        System.out.write(b);
                    } catch ( Exception ex ) {
                        ex.printStackTrace();
                    }
                }
            }
        });
    WindowListener wndCloser = new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        System.exit(0);
      }
    };
    addWindowListener(wndCloser);

    setVisible(true);
  }

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

Представленное решение имеет несколько особенностей. Еще при выполнении предыдущего домашнего задания можно было обратить внимание на то, что при выводе на консоль русского текста из поля ввода сам текст выводился явно не в той кодировке. Это проблемы кодировки Windows. Стандартная русская кодировка в Windows — это 1251, в то же врем DOS-консоль отображает данные в кодировке IBM-866. Внутри программы строки хранятся в кодировке Unicode, а при печати преобразуются в стандартную кодировку. Поэтому выводимые данные представлены на экране неверно. Выход состоит в преобразовании кодировок. Именно это продемонстрировано в данном варианте решения. Вместо обычной печати вида

System.out.println(str);

в программу включен блок преобразования в кодировку 866:

```
try {
    byte[] b = str.getBytes("Cp866");
    System.out.write(b);
} catch ( Exception ex ) {
    ex.printStackTrace();
}
```

Здесь использован метод getBytes(...) класса String, позволяющий преобразовать строку в массив байт согласно одной из известных таблиц кодировок. Самое главное знать название этой таблицы, в данном случае — "Cp866".

Что означает try, catch, Exception и прочее мы сейчас рассматривать не будем. Просто при необходимости преобразования в кодировку IBM-866 будем подставлять данный (или аналогичный) фрагмент.

Оттранслируем и запустим данную программу. Введем несколько записей и отпечатаем результат. Теперь введем еще одну запись с фамилией, которую мы уже вводили. Мы увидим, что счетчик записей внизу экрана приложения не изменился. Отпечатаем теперь результат.

Увы, результат не совсем такой, как нам требуется. От приложения можно было бы ожидать, что при вводе записи с фамилией, которую мы уже вводили, оно заменит список телефонов на вновь введенный. Однако программа просто проигнорировала то, что мы вводили.

Действительно, при добавлении в TreeSet объекты считаются одинаковыми, если метод compareTo(...), в случае интерфейса Comparable, или метод compare(...) интерфейса Comparator, вернул при сравнении этих объектов 0. При этом, новый объект не заменяет существующий, а просто не добавляется в множество. Именно поэтому мы получаем такой эффект.

Решение данной проблемы состоит в замене множества на ассоциативный массив (TreeSet на TreeMap). С TreeMap мы познакомимся на сегодняшнем занятии.

2. Коллекции-ассоциативные массивы (Map)

Как и в двух предыдущих случаях имеется два основных класса поддержки таких

коллекций HashMap и TreeMap.

Класс **HashMap** .

Обратимся к документации по классу HashMap. Многие из его методов такие же, как и у всех коллекций; например, size(), clone() и т.д. Рассмотрим методы, отличающие его от других коллекций.

```
public boolean containsKey(Object key)

   Проверяет есть ли в данном Map указанный ключ.


public boolean containsValue(Object value)

   Проверяет есть ли в данном Map указанное значение.


public Object get(Object key)

   Ищет объект в Map по ключу


public Set keySet()

   Возвращает множество ключей


public Collection values()

   Возвращает множество значений


public Object put(Object key, Object value)

   Заносит в Map пару (ключ, значение). Если данный ключ уже есть в
   Map, то выполняется замена.
```

```
public void putAll(Map t)

   Заносит в Map указанный в качестве параметра другой Map.
```

Нужно отметить, что HashMap дает неупорядоченное множество ключей. Для прохода по всем занесенным в HashMap объектам мы можем при помощи метода values() получить список объектов и пройти по нему, например, используя итератор. Но последовательность объектов при этом проходе — произвольная. Для получения упорядоченного множества значений следует применять TreeMap.

Класс **TreeMap** .

Отличается от HashMap тем, что его элементы упорядочены по ключу. В дополнении к обычным конструкторам имеет конструктор

public TreeMap(Comparator c)

(аналогично TreeSet).

Если же применяется конструктор по умолчанию, то объекты-ключи должны удовлетворять интерфейсу Comparable.

В связи с упорядоченностью по ключу, TreeMap имеет дополнительные методы, основанные на наличии порядка. Это методы

```
public Object firstKey()

   Получить первый ключ в Map.


public Object lastKey()

   Получить последний ключ в Map.


public SortedMap headMap(Object toKey)

   Получить Map до ключа, указанного параметром (исключая его).
```

```
public SortedMap subMap(Object fromKey, Object toKey)

   Получить Map от ключа fromKey (включая его) до ключа toKey (исключая его).


public SortedMap tailMap(Object fromKey)

   Получить Map от ключа fromKey (включая его) до конца.
```

2.1. Практическая работа

Требуется изменить программу домашнего задания так, чтобы она работала не с TreeSet, а с TreeMap.

3. Сортировка и поиск

Библиотека Java имеет развитые средства для выполнения сортировки и поиска. Эти средства реализованы в пакете java.util при помощи классов **Arrays** и **Collections** (не путать с интерфейсом Collection). Класс Arrays обеспечивает сортировку и поиск в массивах, а класс Collections — в коллекциях.

Если посмотреть документацию, то можно увидеть, что оба эти класса не имеют public-конструктора и все их методы статические. Один подобный класс мы уже рассматривали — это класс java.lang.Math. В таких классах мы пользуемся их статическими методами без порождения объекта класса.

Класс **Arrays** (см. документацию)

Имеет множество методов sort(...) для массивов всех примитивных типов, и такое же множество методов binarySearch(...). Методы sort(...) позволяют сортировать массивы, а методы binarySearch(...) — производить эффективный бинарный поиск в отсортированном массиве.

- Алгоритм бинарного поиска состоит в разбиении массива на две примерно равные части. Средний элемент массива сравнивается с ключем, что позволяет выяснить, в какой из частей расположен искомый элемент. После этого процедура повторяется

для выбранной части массива. И так далее, пока не будет найден искомый элемент массива.

Кроме методов sort(...) для массивов примитивных типов есть 4 метода sort(...) для массивов объектов:

```
public static void sort(Object[] a)

public static void sort(Object[] a, int fromIndex, int toIndex)

public static void sort(Object[] a, Comparator c)

public static void sort(Object[] a, int fromIndex, int toIndex, Comparator c)
```

Первые два метода подразумевают, что класс(ы) объектов массива удовлетворяет(ют) интерфейсу **Comparable** . Вторые два метода требуют передачи в качестве последнего параметра объекта-компаратора, который удовлетворяет интерфейсу **Comparator** и обеспечивает сравнение объектов массива.

Пример

Пусть у нас есть массив строк sAry, т.е.

String[] sAry;

тогда

Arrays.sort(sAry);

отсортирует этот массив.

Аналогично методам sort(...) есть два метода binarySearch(...)для массивов объектов:

public static int binarySearch(Object[] a, Object key)

public static int binarySearch(Object[] a, Object key, Comparator c)

Первый из них подразумевает, что объекты массива удовлетворяют интерфейсу Comparable, во втором для сравнения объектов массива используется компаратор.

Предупреждение:

Компаратор должен быть тем же, который был использован при сортировке массива.

Что является результатом binarySearch(...)? Этот метод возвращает индекс найденного элемента массива, если элемент был найден. Если же элемент не найден, то binarySearch(...) возвращает величину (-i-1), где i — это индекс элемента, после которого нужно вставить искомый элементдля того, чтобы сохранился порядок сортировки.

Пример

```
int ind = Arrays.binarySearch(sAry, "Иванов");
if ( ind < 0 )
    System.out.println("Иванов отсутствует");
```

Класс **Collections** (см. документацию)

Предназначен для работы с коллекциями, а не массивами. Как и класс Arrays имеет ряд методов для сортировки и двоичного поиска.

```
public static void sort(List list)

public static void sort(List list, Comparator c)

public static int binarySearch(List list, Object key)

public static int binarySearch(List list, Object key, Comparator c)
```

Эти методы аналогичны одноименным методам класса Arrays, только в качестве первого параметра принимают List (список).

4. Продолжение знакомства с менеджерами компоновки (Layout Managers)

Мы рассмотрели уже ряд менеджеров компоновки — BorderLayout, FlowLayout и GridLayout. Продолжим знакомство с различными менеджерами компоновки визуальной библиотеки Java.

Во-первых, отметим, что все контейнеры визуальных объектов Java используют менеджеры компоновки. Менеджер компоновки (Layout Manager) определяет принцип размещения подкомпонент в данной компоненте. Каждый из контейнеров имеет свой Layout Manager по умолчанию. Т.е., если мы не установим контейнеру некоторый Layout Manager, то будет выбран стандартный для данного контейнера. Так, для JFrame по умолчанию устанавливается BorderLayout, для JPanel — FlowLayout.

Задавая различные менеджеры компоновки и формируя панели при помощи JPanel, можно создать весьма сложные диалоговые формы. При разработке диалоговой формы рекомендуется сначала нарисовать на бумаге, как должна выглядеть форма. Потом разбить ее на панели и подобрать для каждой из них подходящий Layout Manager. Для того чтобы это сделать, нужно представлять себе, как работает тот или иной Layout Manager.

Сегодня мы рассмотрим менеджер **BoxLayout** .

4.1. BoxLayout

Это менеджер, позволяющий создать вертикальный или горизонтальный бокс. Он отличается от GridLayout(1, 0) и GridLayout(0, 1) тем, что ячейки бокса не обязательно имеют одинаковый размер.

Для указания ориентации бокса, вертикальной или горизонтальной, используются константы BoxLayout.X_AXIS и BoxLayout.Y_AXIS.

Использование BoxLayout несколько отличается от использования других менеджеров компоновки. Дело в том, что в конструкторе BoxLayout нужно указать контейнер, для которого он будет применяться. Это означает, что мы не можем сразу создать контейнер и указать в конструкторе объект BoxLayout, как мы это делали с остальными менеджерами. Например,

JPanel pn = new JPanel(new BorderLayout());

Мы должны сначала создать контейнер, потом BoxLayout, а потом связать их методом setLayout():

```
JPanel pn = new JPanel();
```

```
BoxLayout bx = new BoxLayout(pn, BoxLayout.Y_AXIS);
pn.setLayout(bx);
```

Рассмотрим пример

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Dlg2 extends JFrame {


  Dlg2() {
     super("Знакомство с BoxLayout");

     try  {
       UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
     }
     catch(Exception e) {
     }

    setSize(560, 100);
    Container c = getContentPane();
    GridLayout gl = new GridLayout(1,0);
    c.setLayout(gl);
    JPanel pn1 = new JPanel();
    JLabel l1 = new JLabel("Short");
    pn1.setBorder(BorderFactory.createEtchedBorder());
    pn1.add(l1);
    JTextField txt = new JTextField(8);
    pn1.add(txt);
    c.add(pn1);
    JPanel pn2 = new JPanel();
    JLabel l2 = new JLabel("Long Label");
    pn2.add(l2);
    JTextField txt2 = new JTextField(16);
    pn2.add(txt2);
    pn2.setBorder(BorderFactory.createEtchedBorder());
    c.add(pn2);
```

```
    WindowListener wndCloser = new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        System.exit(0);
      }
    };
    addWindowListener(wndCloser);

    setVisible(true);
  }

  public static void main(String[] args) {
    Dlg2 d = new Dlg2();
  }
}
```

Разберем, как работает данное приложение. В нем использован менеджер GridLayout с одной строкой и произвольным количеством колонок (в данном случае - две колонки). Это приводит к тому, что обе ячейки оказались одинакового размера.

Теперь заменим

GridLayout gl = new GridLayout(1,0);

на

BoxLayout gl = new BoxLayout(c, BoxLayout.X_AXIS);

Посмотрим, как изменилось поведение приложения.

4.2. Демонстрационное приложение

В конце занятия рассмотрим приложение (Dlg3.java), которое демонстрирует все рассмотренные нами менеджеры компоновки. В качестве примера экрана оно использует экран ввода информации по книге (наименование, авторы, издательство).

```
// Dlg3.java
// Визульное приложения с использованием различных Layout Manager'ов.
```

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Dlg3 extends JFrame {


  Dlg3() {
     super("Изучение концепции Layout Manager\'ов");

     try  {
       UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
     }
     catch(Exception e) {
     }

    setSize(400, 200);
    Container c = getContentPane();
    JTabbedPane tp = new JTabbedPane();
    c.add(tp, BorderLayout.CENTER);
    JPanel pn1 = new JPanel();
    addFields(pn1);
    tp.add(pn1, "FlowLayout");
    JPanel pn2 = new JPanel(new BorderLayout());
    tp.add(pn2, "BorderLayout");
    addBorderFields(pn2);
    JPanel pn3 = new JPanel(new GridLayout(3, 1, 0, 4));
    tp.add(pn3, "GridLayout");
    addFields(pn3);
    JPanel pn4 = new JPanel();
    BoxLayout bx = new BoxLayout(pn4, BoxLayout.Y_AXIS);
    pn4.setLayout(bx);
    addFields(pn4);
    tp.add(pn4, "BoxLayout-Y");



    WindowListener wndCloser = new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        System.exit(0);
```

```
    }
  };
  addWindowListener(wndCloser);

  setVisible(true);
}

private void addFields(JPanel pn) {
    JPanel p1 = new JPanel();
    JLabel l1 = new JLabel("Наименование");
    JTextField f1 = new JTextField(25);
    p1.add(l1);
    p1.add(f1);
    pn.add(p1);
    JPanel p2 = new JPanel();
    JLabel l2 = new JLabel("Автор(ы)            ");
    JTextField f2 = new JTextField(25);
    p2.add(l2);
    p2.add(f2);
    pn.add(p2);
    JPanel p3 = new JPanel();
    JLabel l3 = new JLabel("Издательство  ");
    JTextField f3 = new JTextField(25);
    p3.add(l3);
    p3.add(f3);
    pn.add(p3);
}

private void addBorderFields(JPanel pn) {
    JPanel p1 = new JPanel();
    JLabel l1 = new JLabel("Наименование");
    JTextField f1 = new JTextField(25);
    p1.add(l1);
    p1.add(f1);
    pn.add(p1, BorderLayout.NORTH);
    JPanel p2 = new JPanel();
    JLabel l2 = new JLabel("Автор(ы)            ");
    JTextField f2 = new JTextField(25);
    p2.add(l2);
    p2.add(f2);
```

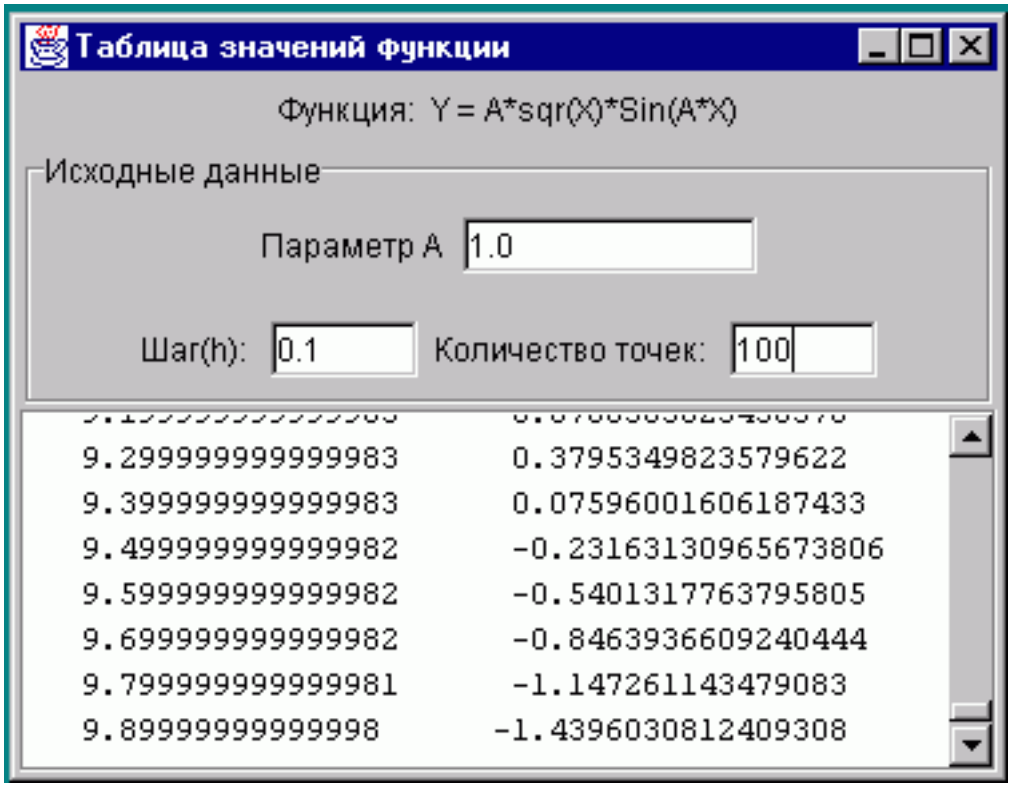
```
      pn.add(p2, BorderLayout.CENTER);
      JPanel p3 = new JPanel();
      JLabel l3 = new JLabel("Издательство  ");
      JTextField f3 = new JTextField(25);
      p3.add(l3);
      p3.add(f3);
      pn.add(p3, BorderLayout.SOUTH);
  }

  public static void main(String[] args) {
    Dlg3 d = new Dlg3();
  }
}
```

Запустим приложение Dlg3.java. Различные закладки демонстрируют различные варианты менеджеров компоновки.

- Следует обратить внимание, что, несмотря на то, что некоторые менеджеры работают (в данном случае) лучше, некоторые хуже, в целом все варианты более менее неплохо выглядят. Это потому, что подкомпоненты добавляются не непосредственно в контейнер, а в панели, вместе со своими метками-промптами. А эти панели уже помещаются в контейнер.

5. Домашнее задание

Написать программу для вычисления таблицы значений функции Y=A*sqr(X)*Sin(A*X)



В нижней части экрана — скроллируемое поле для вывода результатов. Оно образуется при помощи классов JScrollPane и JTextArea (см. Dialog3.java, занятие 7).