Конспект лекций по Java. Занятие 6

В.Фесюнов

1. Практическая работа

Для повторения и закрепления материала предыдущих занятий рассмотрим пример класса, в котором выполняется работа с массивами, а также реализован статический и обычный метод.

Это класс DoubleVector (файл DoubleVector.java). Он предназначен для работы с векторами. В приведенном примере реализовано только скалярное умножение векторов, но при необходимости этот класс можно расширить другими операциями над векторами.

```
public class DoubleVector {

    private double[] vector = null;

    public DoubleVector(double[] vector) {
        this.vector = vector;
    }

    /**
     * Скалярное произведение векторов
     */
    public double mult(DoubleVector anotherVector) {
        double s = 0;
        for ( int i = 0; i < vector.length; i++ ) {
            s += vector[i] * anotherVector.vector[i];
        }
        return s;
    }
```

```
    public static double mult(DoubleVector a, DoubleVector b) {
        return a.mult(b);
    }

    public static void main(String[] args) {
        double[] a = {1, 2, 3, 4};
        double[] b = {1, 1, 1, 1};
        double[] c = {2, 2, 2, 2};
        DoubleVector v1 = new DoubleVector(a);
        DoubleVector v2 = new DoubleVector(b);
        DoubleVector v3 = new DoubleVector(c);
        System.out.println("v1*v2=" + v1.mult(v2));
        System.out.println("v1*v2=" + DoubleVector.mult(v1, v2));
        System.out.println("v1*v3=" + v1.mult(v3));
    }

}
```

В классе DoubleVector есть поле — массив vector, конструктор для построения вектора из массива, два метода mult(...), один из которых статический, а также метод main(...) для проверки работоспособности класса.

Следует обратить внимание на методы mult(...). Первый из них предназначен для умножения данного вектора на другой вектор. Он используется в методе main(...) в строках

System.out.println("v1*v2=" + v1.mult(v2)); и System.out.println("v1*v3=" + v1.mult(v3));

Второй для умножения двух векторов. Он используется в строке

System.out.println("v1*v2=" + DoubleVector.mult(v1, v2));

Статический метод mult(...) реализован на базе обычного метода, путем возврата "a.mult(b)". Альтернативой является реализация алгоритма умножения в статическом методе и вызов его из обычного метода. Это выглядело бы так.

```
    public double mult(DoubleVector anotherVector) {
        return mult(this, anotherVector);
    }
```

```
public static double mult(DoubleVector a, DoubleVector b) {
    double s = 0;
    for ( int i = 0; i < a.vector.length; i++ ) {
        s += a.vector[i] * b.vector[i];
    }
    return s;
}
```

- Данный класс не лишен недостатков. Он предполагает, что при умножении оба вектора имеют одинаковую длину. Для устранения подобных недостатков в Java следует применять аппарат исключительных ситуаций (exceptions), но мы его еще не рассматривали.

2. Знакомство с библиотеками и пакетами.

Библиотека Java — это сборник классов. Если программе нужен какой-то класс, то нужно подключить библиотеку, в которой этот класс находится. Для этого либо устанавливается переменная окружения CLASSPATH, либо задается параметр вызова компилятора и JVM. Для транслятора (javac.exe) — это параметр -classpath, для JVM (java.exe) — это параметр -cp.

Например,

```
rem трансляция
d:\jdk1.3\bin\javac.exe -classpath .;d:\jdk1.3\jre\lib\rt.jar My.java
rem выполнение
d:\jdk1.3\bin\java.exe -cp .;d:\jdk1.3\jre\lib\rt.jar My
```

Обычно физически библиотека — это jar-файл (rt.jar, например). Но свою личную библиотеку можно сделать и просто в каком-либо каталоге. Кроме того, библиотека может быть zip-файлом.

Но библиотека является довольно крупным хранилищем классов. На практике требуется какой-то дополнительный механизм разбиения всего множества классов, хранящихся в библиотеке на отдельные части. В Java таким механизмом являются пакеты.

Пока познакомимся с использованием пакетов из стандартной библиотеки Java. Лучше всего обратиться к документации по API Java. Полный список пакетов стандартной библиотеки Java:

```
java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
java.awt.event
java.awt.font
java.awt.geom
java.awt.im
java.awt.im.spi
java.awt.image
java.awt.image.renderable
java.awt.print
java.beans
java.beans.beancontext
java.io
java.lang
java.lang.ref
java.lang.reflect
java.math
java.net
java.rmi
java.rmi.activation
java.rmi.dgc
java.rmi.registry
java.rmi.server
java.security
java.security.acl
java.security.cert
java.security.interfaces
java.security.spec
java.sql
java.text
java.util
java.util.jar
```

```
java.util.zip
javax.accessibility
javax.naming
javax.naming.directory
javax.naming.event
javax.naming.ldap
javax.naming.spi
javax.rmi
javax.rmi.CORBA
javax.sound.midi
javax.sound.midi.spi
javax.sound.sampled
javax.sound.sampled.spi
javax.swing
javax.swing.border
javax.swing.colorchooser
javax.swing.event
javax.swing.filechooser
javax.swing.plaf
javax.swing.plaf.basic
javax.swing.plaf.metal
javax.swing.plaf.multi
javax.swing.table
javax.swing.text
javax.swing.text.html
javax.swing.text.html.parser
javax.swing.text.rtf
javax.swing.tree
javax.swing.undo
javax.transaction
org.omg.CORBA
org.omg.CORBA_2_3
org.omg.CORBA_2_3.portable
org.omg.CORBA.DynAnyPackage
org.omg.CORBA.ORBPackage
org.omg.CORBA.portable
org.omg.CORBA.TypeCodePackage
org.omg.CosNaming
org.omg.CosNaming.NamingContextPackage
org.omg.SendingContext
```

```
org.omg.stub.java.rmi
```

Нужно разобраться с именами пакетов. Как видно, имя составное, разделенное точками. Это связано с общепринятым в Java принципом построения имен пакетов. Этот принцип состоит в том, что в имени пакета присутствует Internet-адрес разработчика пакета в обратном порядке. На примере:

Пусть Ваш адрес petr@provider.da . Тогда все имена пакетов Ваших приложений должны начинаться с "da.provider.petr.". Так, для пакета, содержащего вспомогательные сервисные классы подойдет имя "da.provider.petr.util".

Кроме того, с именем пакета связана структура каталогов, в которых должны размещаться классы. Это будет рассмотрено подробнее позже, когда будем рассматривать создание собственных пакетов.

Некоторые важные пакеты Java

- java.lang — "самый базовый" из всех базовых пакетов. Без него Java не работает.
- java.io — пакет поддержки ввода/вывода.
- java.util — классы поддержки коллекций объектов, работа с календарем, и др. полезные классы.
- java.awt — пакет AWT: поддержка визуального программирования (базовый пакет).
- javax.swing — пакет SWING: новый пакет визуального программирования. Появился в Java2, базируется на AWT и функционально замещает его (не полностью).
- java.applet — пакет для поддержки создания аплетов.

2.1. Использование пакетов в программах

Для того чтобы класс из библиотеки мог быть использован в программе, его нужно подключить. Для этого в начале java-файла нужно поместить оператор "import". Например, пусть требуется использовать класс ArrayList в разрабатываемом нами классе Ex1 (ArrayList находится в пакете java.util).

Тогда файл Ex1.java может начинаться примерно так:

```
```

```
// Ex1.java

import java.util.ArrayList;
  . . .
```

далее в программе мы уже можем использовать ArrayList. Например,

```
  ArrayList objList = new ArrayList();
  . . .
```

Если в одном программном модуле требуется несколько классов из одного пакета, то можно подключить весь пакет, например,

import java.util.*;

- Пакет java.lang можно не подключать, он подключается автоматически.

2.2. Создание своих собственных пакетов

Для создания собственного пакета нужно

- **1.** Определиться с именем пакета. Пусть, например, Internet-адрес равен my@prov.ua и мы хотим создать пакет вспомогательных программ (подходящее имя — util). Тогда полное имя пакета должно быть ua.prov.my.util.
- **2.** Создать необходимую структуру каталогов. Нужно выбрать или создать каталог, где мы будем хранить все свои пакеты. Пусть, например, это каталог c:\javaproj. В нем нужно создать каталог ua, в каталоге ua создать каталог prov, в нем — my, и наконец в my — util. Все java-файлы создаваемого пакета должны помещаться в каталог util. Т.е. структура имени пакета определяет структуру каталогов.
- **3.** Указать путь к пакету в bat-файлах трансляции и выполнения. Для удобства работы со своими пакетами желательно прописать путь к пакетам в bat-файлах трансляции и выполнения java-программ. Приведем примеры bat-файлов j.bat и jr.bat, в которых указан путь к выбранному нами для хранения всех пакетов каталогу c:\javaproj. Файл для трансляции (j.bat)

```
REM Компилятор JAVA
set JDKHOME=d:\jdk1.3
```

```
set CLASSPATH=.;%JDKHOME%\jre\lib\rt.jar;c:\javaproj
d:\jdk1.3\bin\javac %1 %2 %3 %4 %5
```

Файл для выполнения (jr.bat)

```
REM Запуск программы на JAVA
set JDKH=d:\jdk1.3
set CLASSPATH=.;%JDKH%\jre\lib\rt.jar;%JDKHOME%\jre\lib\i18n.jar;c:\javaproj
%JDKH%\jre\bin\java -cp %CLASSPATH% %1 %2 %3 %4 %5 %6
```

- **4.** В java-файлах пакета указать оператор package. Первым оператором в каждом java-файле пакета должен быть оператор package, в котором указано имя данного пакета. В нашем случае это должен быть оператор вида

```
package ua.prov.my.util;
```

Реализуем простой пример. Создадим в каталоге util такой файл S.java.

```
package ua.prov.my.util;

public class S {

    public static void o(String str) {
          System.out.println(str);
    }

}
```

Оттранслируем его командой

j S.java

Теперь вернемся к примеру в начале занятия и модифицируем его.

- **1.** В начало файла вставим

```
import ua.prov.my.util.S;
```

- **2.** Строки

```
System.out.println("v1*v2=" + v1.mult(v2));
```

```
System.out.println("v1*v2=" + DoubleVector.mult(v1, v2));
System.out.println("v1*v3=" + v1.mult(v3));
```

заменим на

```
S.o("v1*v2=" + v1.mult(v2));
S.o("v1*v2=" + DoubleVector.mult(v1, v2));
S.o("v1*v3=" + v1.mult(v3));
```

- **3.** Оттранслируем и запустим измененную программу. Если мы все сделали правильно, то она будет работать как и прежде. Можно сказать, что мы создали библиотеку (это каталог c:\javaproj). В дальнейшем мы можем расширять ее новыми пакетами и классами.

3. Практическая работа.

То, что мы изучили, достаточно для первого знакомства с визуальным программированием на Java. Некоторые элементы программ будут пока не ясны и мы будем использовать их *as is* . Но, в основном, приводимые далее программы могут быть поняты на основе изученного материала и могут быть использованы как прототипы для построения своих программ в стиле "сборка из известных компонент".

При создании визуальных приложений мы будем, в основном, использовать пакет javax.swing, а также java.awt и java.awt.event.

Рассмотрим первый пример диалоговой программы (файл Dialog1.java)

```
// Dialog1.java
// Первый пример визульного приложения на Java.

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

public class Dialog1 {

  public static void main(String[] args) {
// фрагмент as is (1)
```

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

    JFrame frm = new JFrame("Первое визуальное приложение");
    frm.setSize(300, 200);
    Container c = frm.getContentPane();
    c.add(new JLabel("Hello, привет"));

// фрагмент as is (2)
    WindowListener wndCloser = new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        System.exit(0);
      }
    };
    frm.addWindowListener(wndCloser);
//

    frm.setVisible(true);
  }
}
```

3.1. Замечания по кодировке

Для того, чтобы запустить данный пример, можно просто скопировать текст примера в буфер, а потом в некотором редакторе создать файл Dialog1.java и скопировать в него содержимое буфера.

Единственная проблема — в кодировке русских символов. При формировании диалога используется кодировка Windows 1251. Поэтому, если текст будет набран в другой кодировке, то он будет отображен в окне приложения неправильно.

Каждый текстовый редактор имеет свои способы переключения кодировок. Скажем, при использовании встроенного редактора Far-Manager изменить кодировку с DOS (KOI8-R) на Win (Windows 1251) можно при помощи клавиши F8.

3.2. Краткие пояснения к примеру

1. В примере использованы следующие классы стандартной библиотеки Java

- UIManager — рассмотрим позднее;
- JFrame — позволяет сформировать основное окно приложения. Все остальные визуальные компоненты помещаются внутрь этого окна.
- Container — класс для визуальных классов-контейнеров, т.е. визуальных компонент, которые могут внутри себя содержать другие визуальные компоненты.
- JLabel — класс для создания меток.
- WindowListener и WindowAdapter — as is.

2. "фрагмент as is (1)" обеспечивает Windows Look&Fill. Закомментируем его и посмотрим, что получится.

3. "фрагмент as is (2)" обеспечивает завершение всего приложения в случае, если закрылось главное окно приложения. Тоже полезно закомментировать и посмотреть, что получиться.

4. Строки

```
JFrame frm = new JFrame("Первое визуальное приложение");
frm.setSize(300, 200);
```

имеют очевидный смысл. (см.документацию по JFrame в пакете javax.swing).

5. Строки

```
Container c = frm.getContentPane();
c.add(new JLabel("Hello, привет"));
```

требуют пояснения. Главное окно (JFrame) устроено сложным образом. Оно состоит из ряда элементов. Тот элемент, в который следует добавлять другие визуальные компоненты, может быть получен при помощи метода getContentPane(). Мы запоминаем ссылку на него в переменной 'c' и потом используем его во второй строке фрагмента, которая просто создает и добавляет (метод add) метку на окно.

- При добавлении визуальных компонент в Java не указывается ни их позиция, ни размер. Это связано с тем, что все визуальные приложения Java масштабируемы и

для обеспечения масштабируемости используются другие принципы размещения визуальных элементов, а именно, используются различные Layout'ы.

6. Строка frm.setVisible(true);

выводит окно на экран и активизирует диалог с пользователем.

4. Задание на дом

- **1.** Поместить класс DoubleVector в пакет algebra. Для этого убрать из него main-метод, создать отдельно класс TestVector, перенести в него этот main-метод, оттранслировать его и запустить на выполнение.
- **2.** Переделать TestVector в диалоговую программу. Т.е. вместо вывода на консоль операторами System.out.println(...) нужно обеспечить вывод на экран. - Предлагается такой простейший вариант (не очень элегантный, но...). У нас должно быть три операции вывода результата. Делаем на экране не одну, а три метки и заполняем их нужными значениями либо в конструкторе, либо методом setText(...) класса JLabel. *При простом добавлении трех меток возникнет небольшая проблема — они будут размещены одна поверх другой — и видна будет только одна из них. Чтобы избежать этого эффекта, достаточно добавить в конструктор строку*

```
  c.setLayout(new FlowLayout());
```

перед строками добавления меток.