Классы и объекты. Пакеты.

Шевченко Д.В. Проскурина Г.В.

5 марта 2013 г.

Аннотация

Ключевые слова: иерархия объектов, разделение по пакетам, декомпозиция приложения.

1 Цель работы

Освоить базовые принципы ООП. Получить практические навыки по работе с классами и объектами на платформе java. Научиться проводить декомпозицию приложений с использованием пакетов.

2 Общие сведения

2.1 Классы

Рассмотрим пример кода, реализующего класс **Bicycle** (Велосипед), дающий общее представление об описании класса. Далее каждый раздел описания класса будет рассмотрен подробно, поэтому сейчас не останавливайтесь на деталях.

```
1  public class Bicycle {
2
3      // the Bicycle class has three fields
4      protected int cadence;
5      protected int gear;
6      protected int speed;
7
8      // the Bicycle class has one constructor
9      public Bicycle(int startCadence, int startSpeed, int startGear) {
10         gear = startGear;
11         cadence = startCadence;
12         speed = startSpeed;
13     }
14
15     // the Bicycle class has four methods
16     public void setCadence(int newValue) {
17         cadence = newValue;
18     }
19
20     public void setGear(int newValue) {
21         gear = newValue;
22     }
```

```
    public void applyBrake(int decrement) {
        speed -= decrement;
    }

    public void speedUp(int increment) {
        speed += increment;
    }

}
```

Описание класса **MountainBike**, который является подклассом класса **Bicycle**, может выглядеть следующим образом:

```
public class MountainBike extends Bicycle {

    // the MountainBike subclass has one field
    public int seatHeight;

    // the MountainBike subclass has one constructor
    public MountainBike(int startHeight, int startCadence, int startSpeed,
      int startGear) {
        super(startCadence, startSpeed, startGear);
        seatHeight = startHeight;
    }

    // the MountainBike subclass has one method
    public void setHeight(int newValue) {
        seatHeight = newValue;
    }

}
```

MountainBike наследует все поля и методы класса **Bicycle** и добавляет поле **seatHeight** и метод **setHeight** для установки значения этого поля.

2.2 Объявление класса

Как вы заметили, класс определяется следующим образом

```
class MyClass {
    //Fields, constructors and methods
}
```

В общем виде, объявление класса может включать следующие компоненты:

- Модификаторы (public, private...) назначение которых вы узнаете далее.
- Имя класса
- Имя родительского класса (superclass), если он существует, предваренное ключевым словом *extends*. Класс может иметь только один родительский класс.
- Список интерфейсов которые реализует класс, если существуют, предваряются ключевым словом implements.
- Тело класса заключается в {}.

2.3 Области видимости классов

У классов и методов имеются модификаторы доступа, указывающие, кто к ним может обращаться - они перечислены в таблице 1. Следует отметить, что по умолчанию в современных IDE при генерации нового класса проставляется доступ **public** - самый либеральный из всех. В результате (и такое не является гипотетическим, а встречается на практике очень часто) в проекте все классы могут быть объявлены, как публичные. Это резко противоречит духу проектирования больших систем (не только в программировании). Как правило, большой проект представляет из себя несколько модулей, отвечающих за различные слои приложения. Эти модули должны взаимодействовать друг с другом через строго ограниченный набор классов и интерфейсов, определяющих протокол взаимодействия. Следовательно, публичными должны быть именно эти классы и интерфейса, а все остальные элементы модулей не должны иметь такой видимости!

Образно говоря, ситуацию можно представить следующим образом. Нам нужно устройство, которое поддерживает определенный протокол (например, USB). Мы просматриваем характеристики устройств, имеющихся в наличии, и выбираем то, что поддерживает этот протокол. Это все, что нам нужно знать о данном устройстве. Но вместо этого нам еще предоставляют полностью документацию на это устройство (принципиальную схему устройства и т.д.), что нам вовсе и ни к чему. Несоблюдение этого аспекта проектирования приводит к резкому замедлению работы IDE, поскольку с ростом проекта возрастает количество возможных связей между классами проекта. Области видимости позволяют резко сократить это количество связей, облегчив компилятору и IDE жизнь.

Модификатор	Класс	Пакет	Подкласс	Все классы
public	Y	Y	Y	Y
protected	Y	Y	Y	N
default	Y	Y	N	N
private	Y	N	N	N

Таблица 1: Модификаторы доступа

2.4 Объявление методов

Вот пример объявления метода

```
public double calculateAnswer(
   double wingSpan,
   int numberOfEngines,
   double length,
   double grossTons
) {
        //Method body
}
```

Определение метода состоит из шести компонентов:

- Модификаторы – public, private и другие.
- Возвращаемый тип – тип данных, возвращаемых методом или void, если метод ничего не возвращает.
- Имя метода.
- Список параметров, заключенный в скобки – разделенные запятыми пары Тип имя переменной. Если входных параметров нет, необходимо оставить скобки пустыми.
- Список исключений – будет рассмотрен в дальнейшем.

- Тело метода – код метода, заключенный в фигурные скобки.

2.5 Добавление конструктора

Класс содержит конструкторы, которые вызываются для создания экземпляра класса. Конструктор выглядит как объявление метода за исключением того, что имя конструктора должно совпадать с именем класса и отсутствует возвращаемый тип. Так, например, **Bicycle** содержит один конструктор:

```
public Bicycle(int startCadence, int startSpeed, int startGear) {
        gear = startGear;
        cadence = startCadence;
        speed = startSpeed;
}
```

Для создания нового объекта класса **Bicycle**, конструктор вызывается с помощью оператора *new*.

```
Bicycle myBike = new Bicycle(30, 0, 8);
```

3 Создание объекта

Класс является образцом для создания объекта, а объекты – экземплярами своего класса. Допустим, имеется следующий класс:

```
public class Point {
    public int x = 0;
    public int y = 0;
    //constructor
    public Point(int a, int b) {
        x = a;
        y = b;
    }
}

// Somewhere in code
Point originOne = new Point(23, 94);
```

Состояние этой переменной будет выглядеть так, как показано на рисунке 1.

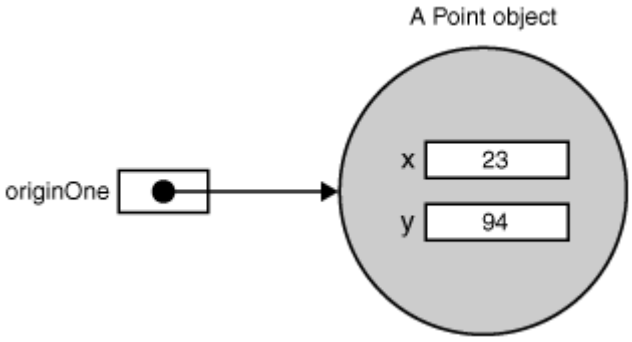

Рис. 1: Переменная и объект в памяти

4 Пакеты

Программа на Java представляет собой набор пакетов (packages). Каждый пакет может включать вложенные пакеты, то есть они образуют иерархическую систему.

Кроме этого, пакеты могут содержать классы и интерфейсы, и таким образом группируют типы, что необходимо сразу для нескольких целей.

Во-первых, чисто физически невозможно работать с большим количеством классов, если они 'свалены в кучу'.

Во-вторых, модульная декомпозиция облегчает проектирование системы. К тому же, как будет рассмотрено ниже, существует специальный уровень доступа, позволяющий типам из одного пакета более 'тесно' взаимодействовать друг с другом, чем с классами из других пакетов. Таким образом, с помощью пакетов производится логическая группировка типов. Из ООП известно, что большая связность системы, то есть среднее количество классов, с которыми взаимодействует каждый класс, серьезно усложняет развитие и поддержку такой системы. С применением пакетов гораздо проще эффективно организовать взаимодействие подсистем друг с другом.

Наконец, каждый пакет имеет свое пространство имен, что позволяет создавать одноименные классы в различных пакетах. Таким образом, разработчикам не приходится тратить время на разрешение конфликта имен.

5 Пример создания приложения с пакетами

Рассмотрим проект со следующей структурой:

```
bin
src
-->com
----->vlsu
-------->MainClass.java
-------->util
---------->UtilClass.java
```

Это типовая структура проектов, используемая на практике. Исходники располагаются в папке **src**. Как несложно видеть, в этих исходниках есть два класса, первый это **com.vlsu.MainClass** следующего содержания:

```
1  package com.vlsu;
2
3  import com.vlsu.util.UtilClass;
4
5  public class MainClass {
6     public static void main(String [] args) {
7        System.out.println("The main class started successfully");
8        UtilClass uc = new UtilClass();
9        System.out.println("The answer to all questions of life is: "
10          + uc.getAnswer());
11    }
12 }
```

Как несложно видеть, этот класс использует вспомогательный класс, располагающийся в другом пакете. Хотя может показаться, что пакет **com.vlsu.util** является дочерним по отношению к **com.vlsu**, но его следует воспринимать как независимый пакет (хотя с точки зрения организации проекта между ними разработчик видит определенную связь). Компилятор не увидит классы этого подпакета в коде классов родительского пакета, если не произвести явный импорт, что и делается в коде данного класса. Вспомогательный класс имеет следующий код:

```java
package com.vlsu.util;

public class UtilClass {
   public UtilClass() {
      System.out.println("The instance of UtilClass created successfully...");
   }

   public String getAnswer() {
      return "42";
   }
}
```

Теперь необходимо на основе этих исходников получить файлы с байт-кодом. Для этого мы используем компилятор, перейдя в корень проекта, и выполним команду, как показано на рисунке 2

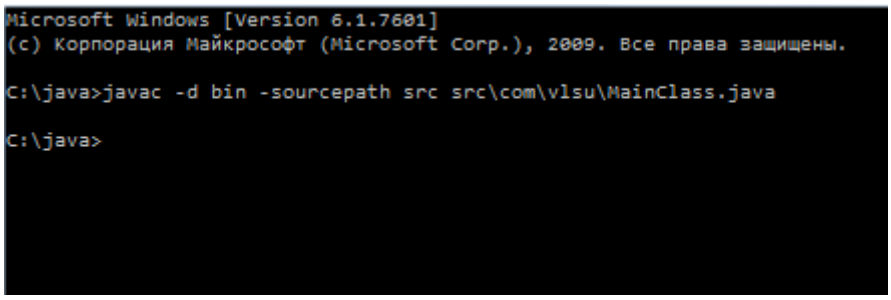

Рис. 2: Результат компиляции проекта

Здесь следует отметить несколько особенностей:

- Компилятору указывается, что сгенерированные файлы нужно поместить в каталог **bin**, в противном случае по умолчанию он разместит их там же, где находятся исходные файлы. Обычно в проекте стараются отделить исходный код от сгенерированных классов, чтобы не возникало путаницы.
- Компилятору указывается, что при необходимости поиска классов он может использовать не только стандартные пакеты, но и исходные файлы, находящиеся в каталоге **src**. Он является корнем для исходных кодов, и все пакеты размещаются в нем.
- Мы указываем компилятору целевой класс **MainClass**, который нам необходим. Однако в результате компиляции в каталоге **bin** будет размещаться и вспомогательный класс тоже, что видно на рисунке 3. Дело в том, что компилятор способен распознать такие зависимости, и автоматически компилирует заодно и вспомогательные файлы. В противном случае все это пришлось бы указывать вручную при компиляции. Для небольшого проекта это годится, однако для серьезных проектов с несколькими сотнями классов ручной подход неприемлем.

Рис. 3: Каталог со сгенерированными файлами

Теперь можно попытаться выполнить метод **main** главного класса так, как показано на рисунке 4.

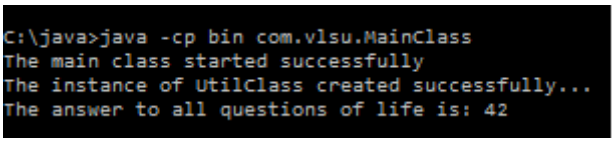

Рис. 4: Результат выполнения программы

Здесь следует также отметить несколько особенностей:

- Среде выполнения указывается лишь главный класс, но не метод - название этого метода и сигнатура указаны среде выполнения по умолчанию.
- Среде выполнения указывается, что можно воспользоваться классами, располагающимися в папке **bin**. При поиске нужного класса во время выполнения интерпретатор сначала ищет класс в этой папке (с учетом разбиения по пакетам), а затем в наборе стандартных библиотек Java (они располагаются в каталоге, прописанном в системной переменной **CLASSPATH**).

6 Варианты заданий

Независимо от варианта задания, необходимо выполнить следующее:

- Выполнить пример выше.
- Реализовать классы Rectangle и Point. Класс Point определяется двумя координатами точки, класс Rectangle определяется точками левого верхнего угла и правого нижнего угла.

Каждый вариант предполагает написание программы на языке Java, которая работает в консольном режиме и считывает входные данные с клавиатуры. Вспомогательные классы должны располагаться в отдельном пакете, а основной класс должен импортировать их. Структура проекта должна быть такой же, как показано в примере. Выражение типа 'Класс определяется следующими параметрами' следует понимать так - у класса должны быть такие приватные поля, они должны передаваться в качестве аргументов конструктору класса, и к ним должны быть методы доступа на чтение в классе.

1. Необходимо реализовать приложение, работающее с отрезками. Первый вспомогательный класс - точка - определяется параметрами x и y - координатами на плоскости. Второй вспомогательный классс - отрезок - определяется параметрами $p_1 1$ и p_2 - точками начала и конца отрезка. Помимо этого, у отрезка должен быть метод **length**, возвращающий длину этого отрезка. Основной класс должен запросить у пользователя координаты первой и второй точки, создать на их основе отрезок, после чего вывести пользователю его длину.
2. Необходимо реализовать приложение, работающее с кругами. Первый вспомогательный класс - точка - определяется параметрами x и y - координатами на плоскости. Второй вспомогательный классс - круг - определяется параметрами p и R - точкой центра круга и радиусом. Помимо этого, у отрезка должен быть метод **inside**, который в качестве аргумента принимает точку. Он должен возвращать **true**, если точка находится внутри круга, и **false** в противном случае Основной класс должен запросить у пользователя координаты центра круга и его радиус, после чего создать объект круга. После этого он должен запросить у пользователя координаты точки и вывести ответ - находится ли данная точка внутри круга.

3. Необходимо реализовать приложение, работающее с отрезками. Первый вспомогательный класс - точка - определяется параметрами x, y и z - координатами точки в трехмерном пространстве. Второй вспомогательный классс - отрезок - определяется параметрами p_1 и p_2 - точками начала и конца отрезка. Помимо этого, у отрезка должен быть метод **length**, возвращающий длину этого отрезка. Основной класс должен запросить у пользователя координаты первой и второй точки, создать на их основе отрезок, после чего вывести пользователю его длину.

4. Необходимо реализовать приложение, работающее с шарами. Первый вспомогательный класс - точка - определяется параметрами x, y и z - координатами точки в трехмерном пространстве. Второй вспомогательный классс - шар - определяется параметрами p и R - точкой центра шара и радиусом. Помимо этого, у отрезка должен быть метод **inside**, который в качестве аргумента принимает точку. Он должен возвращать **true**, если точка находится внутри шара, и **false** в противном случае Основной класс должен запросить у пользователя координаты центра шара и его радиус, после чего создать объект шара. После этого он должен запросить у пользователя координаты точки и вывести ответ - находится ли данная точка внутри шара.

5. Необходимо реализовать приложение, работающее с математическими операциями. Первый вспомогательный класс - сумматор - определяется параметрами x и y - числами, которые надо сложить. Второй вспомогательный класс - субтрактор - определяется тоже двумя параметрами x и y - числами, второе из которых надо вычесть из первого. Оба класса имеют дополнительный метод **calc**, возвращающий результат выполнения операции. Основной класс должен запросить у пользователя два числа и создать на их основе сумматор и субтрактор, после чего вывести пользователю на экран результаты выполнения операций этими двумя элементами.

6. Необходимо реализовать приложение, работающее с математическими операциями. Первый вспомогательный класс - множитель - определяется параметрами x и y - числами, которые надо перемножить. Второй вспомогательный класс - делитель - определяется тоже двумя параметрами x и y - числами, первое из которых надо поделить на второе. Оба класса имеют дополнительный метод **calc**, возвращающий результат выполнения операции. Основной класс должен запросить у пользователя два числа и создать на их основе множитель и делитель, после чего вывести пользователю на экран результаты выполнения операций этими двумя элементами.

7. Необходимо реализовать приложение, работающее с логическими операциями. Первый вспомогательный класс - конъюнктор - определяется параметрами x и y - двумя булевыми переменными, которые надо сложить с помощью операции 'AND'. Второй вспомогательный класс - дизъюнктор - также определяется двумя параметрами x и y - булевыми переменными, которые надо сложить с помощью операции 'OR'. Оба класса имеют дополнительный метод **calc**, возвращающий результат выполнения операции (также булево значение). Основной класс должен для всех возможных пар значений x и y создать экземпляры этих двух элементов и вывести на экран результат их выполнения.

8. Необходимо реализовать приложение, работающее с логическими операциями. Первый вспомогательный класс - исключитель - определяется параметрами x и y - двумя булевыми переменными, которые надо сложить с помощью операции 'XOR'. Второй вспомогательный класс - компаратор - также определяется двумя параметрами x и y - булевыми переменными, которые надо проверить на равенство. Оба класса имеют дополнительный метод **calc**, возвращающий результат выполнения операции (также булево значение). Основной класс должен для всех возможных пар значений x и y создать экземпляры этих двух элементов и вывести на экран результат их выполнения.

9. Необходимо реализовать приложение, работающее со строками. Первый вспомогательный класс - повышатель - определяется параметром *str* - строкой, все символы которой надо

преобразовать в верхний регистр. Второй вспомогательный класс - понижатель - определяется параметром *str* - строкой, все символы которой надо преобразовать в нижний регистр. Оба класса имеют дополнительный метод **output**, который возвращает результат преобразования строки. Основной класс должен запросить у пользователя строку, создать на ее основе экземпляры этих двух классов и вывести на экран результат их выполнения.

10. Необходимо реализовать приложение, работающее со строками. Первый вспомогательный класс - инвертор - определяется параметром *str* - строкой, символы которой надо вывести в обратном порядке. Второй вспомогательный класс - компрессор - определяется параметром *str* - строкой, из которой надо удалить все пробелы. Оба класса имеют дополнительный метод **output**, который возвращает результат преобразования строки. Основной класс должен запросить у пользователя строку, создать на ее основе экземпляры этих двух классов и вывести на экран результат их выполнения.

7 Контрольные вопросы

- Какие преимущества дает использование пакетов?
- Как можно обратиться к классу, который находится в другом пакете?
- Что такое статический импорт и для чего он нужен?
- Как происходит передача параметров в метод?
- Найдите ошибку в следующей программе.

```
public class SomethingIsWrong {
    public static void main(String[] args) {
        Rectangle myRect;
        myRect.width = 40;
        myRect.height = 50;
        System.out.println("myRect's area is " + myRect.area());
    }
}
```

- Каким образом происходит удаление объектов?
- Следующий фрагмент кода создает два объекта Point и Rectangle.

```
...
Point point = new Point(2,4);
Rectangle rectangle = new Rectangle(point, 20, 20);
point = null;
...
```

Сколько ссылок существует на каждый объект и будут ли удалены объекты после выполнения?