МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П.КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»

А. В. Гаврилов

Объектно-ориентированное программирование на Java

Задания на лабораторные работы

Самара
2011

Автор: ГАВРИЛОВ Андрей Вадимович

Набор заданий на лабораторные работы по курсу «Объектно-ориентированное программирование» предназначен для бакалавров второго курса факультета информатики направления 010400.62 «Прикладная математика и информатика».

Лабораторная работа №1

Задание на лабораторную работу

В процессе написания тестовых заданий ознакомиться со структурой исходного кода для Java, изучить особенности областей видимости и использования пакетов.

Каждое следующее задание надо выполнять в новом каталоге (например, `Task1`, `Task2` и так далее).

Задание 1

Запустите компилятор `javac` без параметров и ознакомьтесь с форматом задания параметров компилятора.

Запустите программу `java` без параметров и ознакомьтесь с форматом задания параметров запуска виртуальной машины Java (JVM).

Задание 2

Создайте файл `MyFirstProgram.java`, содержащий исходный код одного пустого класса с именем `MyFirstClass`:

```
--- MyFirstProgram.java --------------------------------
       class MyFirstClass {
       }
--------------------------------------------------------
```

Откомпилируйте его с помощью компилятора `javac`. Для этого необходимо выполнить команду "`javac MyFirstProgram.java`". Запустите полученный файл класса на выполнение с помощью команды "`java <Имя класса>`".

Обратите внимание на то, что на вход компилятора необходимо подавать имя файла с расширением, а на вход JVM – без расширения.

Также обратите внимание на то, что в операционных системах, в которых имена файлов чувствительны к регистру, имена файлов для компиляции и

запуска следует указывать с учетом регистра. При этом расширение файла с исходным кодом должно быть “.java” (маленькими буквами).

Занесите полученные после запуска результаты в отчет.

Добавьте в класс метод main():

```
--- MyFirstProgram.java --------------------------------
        class MyFirstClass {
            void main(String[] s) {
                System.out.println(“Hello world!!!”);
            }
        }
--------------------------------------------------------
```

Точкой входа программы является метод main() с параметрами String[] s.

Для вывода текстовой информации на экран в данном случае используется функция println() класса PrintStream. Поле out этого типа является статическим полем класса System, поэтому может использоваться без создания объекта типа System.

Откомпилируйте и запустите программу на выполнение. Внесите результаты запуска в отчет.

Логично предположить, что метод main() должен быть статическим, т.к. на момент запуска программы ни одного объекта типа MyFirstClass не существует. Сделайте метод статическим, снова откомпилируйте и запустите программу, внесите результаты запуска в отчет.

Внесите в текст программы необходимые для ее запуска изменения, откомпилируйте и запустите программу, внесите результаты запуска в отчет.

Задание 3

Замените текст метода main() на следующий:

```
--- MyFirstClass.main ---------------------------------
        for (int i = 0; i < s.length; i++)
            System.out.println(s[i]);
-------------------------------------------------------
```

Откомпилируйте и запустите программу, добавив в командную строку ряд аргументов. Например, следующим образом: "`java MyFirstClass arg1 arg2 arg3 arg4 arg5`". Внесите полученные результаты в отчет.

Задание 4

В том же файле `MyFirstProgram.java` после описания класса `MyFirstClass` добавьте описание второго класса `MySecondClass`, реализующего следующую функциональность:

- имеет два приватных поля типа `int`;
- методы для получения и модификации их значений;
- конструктор, создающий объект и инициализирующий значения полей;
- метод с возвращаемым типом `int`, реализующий над этими числами действие в зависимости от варианта лабораторной работы.

Список действий над числами в зависимости от номера варианта приведен в таблице 1.

Таблица 1. Параметры программы для различных вариантов.

№ варианта	Выполняемое над числами действие
1	Сложение
2	Вычитание
3	Умножение
4	Деление
5	Взятие остатка от деления
6	Побитовое «и»

7	Побитовое «или»
8	Побитовое «исключающее или»
9	Выбор максимального из чисел
10	Выбор минимального из чисел

Код метода `MyFirstClass.main()` при этом следует заменить на следующий (в угловых скобках указаны элементы, имена которых следует подставить при написании программы):

```
--- MyFirstClass.main ----------------------------------
        <Создание и инициализация объекта “o” типа
        MySecondClass>;
        int i, j;
        for (i = 1; i <= 8; i++) {
            for(j = 1; j <= 8; j++) {
                o.<Метод установки значения первого
        числового поля>(i);
                o.<Метод установки значения второго
        числового поля>(j);
                System.out.print(o.<Метод, реализующий
        действие над числами>());
                System.out.print(" ");
            }
            System.out.println();
        }
--------------------------------------------------------
```

Откомпилируйте и запустите программу. Полученные результаты внесите в отчет.

Задание 5

Вынесите код класса `MySecondClass` без изменений в отдельный файл с именем `MyFirstPackage.java`, и поместите его в поддиректорию `myfirstpackage`, откомпилируйте. Попробуйте откомпилировать файл `MyFirstProgram.java`.

Добавьте в начало исходного кода в файле `MyFirstProgram.java` следующий код:

```
--- MyFirstProgram.java --------------------------------
      import myfirstpackage.*;
--------------------------------------------------------
```

Снова попробуйте откомпилировать `MyFirstProgram.java`. Далее, следуя сообщениям компилятора и изменяя исходный код программы, добейтесь ее работоспособности. В отчет внесите структуру каталогов и имена исходных файлов, вместо выходного текста программы кратко укажите ошибки, исправлявшиеся вами.

Задание 6

Запустите программу `jar`, предназначенную для создания архивов, и ознакомьтесь с форматом задания ключей для формирования архивов.

Скопируйте в рабочую папку, сохранив структуру каталогов, только файлы с расширением `class`, полученные в результате выполнения задания 5.

Создайте файл `manifest.mf`, содержащий следующий код:

```
--- manifest.mf ----------------------------------------
      Manifest-Version: 1.0
      Created-By: <Ваши фамилии>
      Main-Class: MyFirstClass
--------------------------------------------------------
```

Обратите внимание на то, что после имени класса надо обязательно поставить символ новой строки.

Создайте архив `myfirst.jar`, включив в него полученные ранее файлы классов и указав созданный вами манифест-файл. Переместите полученный файл в другую директорию (например, поддиректорию `MyJar`) и запустите его на выполнение. Внесите результаты в отчет, указав вместо выходного текста программы команду запуска файла на выполнение.

Лабораторная работа №2

Задание на лабораторную работу

В процессе написания тестовых заданий ознакомиться с основными конструкциями языка Java, принципами создания классов.

Задание

Создать класс, реализующий работу с векторами (набор вещественных чисел, координат) и базовые операции векторной арифметики. Класс должен удовлетворять следующим требованиям.

Экземпляр должен соответствовать вектору фиксированной длины (она задается как параметр конструктора).

Должны быть реализованы следующие методы:

доступа к элементам вектора (получения значения и изменения значения),

получения «длины» вектора (количества его элементов),

поиска минимального и максимального значений из элементов вектора,

сортировки вектора (по возрастанию или убыванию – на выбор),

нахождения евклидовой нормы,

умножения вектора на число,

сложения двух векторов,

нахождения скалярного произведения двух векторов.

В процессе выполнения задания НЕЛЬЗЯ пользоваться утилитными классами Java (кроме метода Math.sqrt()).

Точка входа программы может быть реализована в классе, в отладочных целях, но не обязательна к написанию.

Лабораторная работа №3

Задание на лабораторную работу

В процессе написания тестовых заданий ознакомиться с механизмом исключений в Java и концепцией интерфейсов.

Задание 1

Модифицировать класс из прошлой работы, оставив в нем следующие методы:

конструктор,

доступа к элементам вектора,

получения размерности вектора,

получения нормы вектора.

Назвать получившийся класс `ArrayVector`.

Задание 2

Создать отдельный класс `Vectors`, содержащий статические методы работы с векторами:

умножения вектора на скаляр,

сложения двух векторов,

нахождения скалярного произведения двух векторов.

Задание 3

Объединить получившиеся типы в пакет `vectors`.

Задание 4

В этом же пакете описать классы ошибок выхода за границы вектора `VectorIndexOutOfBoundsException` и несоответствия длин векторов `IncompatibleVectorSizesException`.

Изменить методы классов так, чтобы они корректно обрабатывали ошибки и выбрасывали исключения.

Задание 5

В этом же пакете описать класс, реализующий функциональность, сходную с классом из задания 1 (доступ к элементу по номеру, возвращение размерности, вычисление нормы), основанный на связном списке (достаточно односвязного), а также методы:

добавления элемента,

удаления элемента.

Назвать получившийся класс `LinkedListVector`.

Задание 6

В этом же пакете описать интерфейс `Vector` взаимодействия с векторами, имеющий методы, соответствующие общей функциональности двух созданных классов векторов. Сделать так, чтобы оба класса реализовывали этот интерфейс.

Задание 7

Исправить класс `Vectors` таким образом, чтобы он работал со ссылками типа интерфейса.

Лабораторная работа №4

Задание на Лабораторную работу

В процессе написания тестовых заданий ознакомиться с механизмом систем ввода и вывода данных.

Задание 1

Сохранив структуру методов класса `LinkedListVector` (т.е. доступ к элементам ведется только через метод, возвращающий ссылку на узел), изменить вид связного списка.

Список должен быть: двусвязным, циклическим, с выделенной головой (т.е. голова не хранит значение, а «пустой» список состоит из одной головы, замкнутой на себя). В соответствии с новой структурой должны быть переписаны методы добавления значения в список, удаления элемента из списка по номеру и метод доступа к узлу по его номеру.

Кроме того, должна быть реализована оптимизация скорости доступа: внутри экземпляра списка должна храниться ссылка узел (и его номер), к которому производилось предыдущее обращение, а метод доступа к узлу по его номеру должен двигаться к заданному узлу от ближайшей точки: от «предыдущего» узла, от головы вперед, от головы назад («от хвоста»).

Задание 2

Модифицировать класс `Vectors` из предыдущей работы, добавив в него новые методы:

- записи вектора в байтовый поток

```
public static void outputVector(Vector v, OutputStream out),
```

- чтения вектора из байтового потока

```
public static Vector inputVector(InputStream in),
```

- записи вектора в символьный поток

```
public static void writeVector(Vector v, Writer out),
```

- чтения вектора из символьного потока

  `public static Vector readVector(Reader in).`

В обоих случаях записанный вектор должен представлять собой последовательность чисел, первым из которых является размерность вектора, а остальные числа, естественно, являются значениями координат вектора.

В случае символьного потока рекомендуется считать, что один вектор записывается в одну строку (числа разделены пробелами). Для чтения вектора из символьного потока рекомендуется использовать класс `StreamTokenizer`.

Описать класс `Main`, находящийся вне пакета `vectors`, содержащий точку входа программы. В методе `main()` проверить возможности методов записи, в качестве реальных потоков используя файловые потоки, а также потоки `System.in` и `System.out`.

Задание 3

Модифицировать классы `ArrayVector` и `LinkedListVector` таким образом, чтобы они были сериализуемыми.

Продемонстрировать возможности сериализации (в методе `main()`), записав в файл объект, затем считав и сравнив с исходным (по сохраненным значениям).

Лабораторная работа №5

Задание на лабораторную работу

В процессе написания тестовых заданий ознакомиться с механизмом образцов проектирования «Итератор» и упрощенной версией «Фабричного метода».

Задание 1

Модифицировать интерфейс `Vector` таким образом, чтобы в нем был объявлен метод `java.util.Iterator iterator()`.

Задание 2

Реализовать этот метод в классах, реализующих данный интерфейс. Для этого, естественно, следует описать некие дополнительные классы с некими соответствующими методами.

Задание 3

Проверить работу итераторов.

Задание 4

Описать новый интерфейс `VectorFactory`, содержащий единственный метод, создающий новый экземпляр вектора по его длине.

Задание 5

В классе `Vectors` создать статическое поле типа `VectorFactory` и соответствующий ему метод `setVectorFactory`, позволяющие, соответственно, хранить ссылку и устанавливать ссылку на текущую фабрику векторов. По умолчанию поле должно ссылаться на объект некоторого класса (его также требуется описать), порождающего экземпляры класса `ArrayVector`.

Задание 6

В классе `Vectors` описать метод `public static Vector createInstance(int size)`, с помощью текущей фабрики создающий новый экземпляр вектора с указанным размером. В остальных методах класса `Vectors` заменить прямое создание экземпляров вектора на вызов этого метода.

Лабораторная работа №6

Задание на лабораторную работу

В процессе написания тестовых заданий ознакомиться с методами класса `Object` и расширить функциональность имеющегося пакета.

Задание 1

Добавить в классы векторов `ArrayVector` и `LinkedListVector` реализации методов `String toString()`. Рекомендуется использовать для формирования строки экземпляр класса `StringBuffer`.

Задание 2

Добавить в классы векторов реализации методов `boolean equals(Object obj)`. Метод должен возвращать `true` только в том случае, если объект, на который передана ссылка, является вектором и имеет те же значения координат, что и текущий объект. Рекомендуется оптимизировать работу методов с учетом знания о внутренней структуре класса.

Задание 3

Добавить в классы векторов реализации методов `int hashCode()`. Значение хеш-функции вычислять как значение побитового исключающего ИЛИ битовых представлений всех элементов вектора в случае типа `float` базового элемента вектора, а в случае типа `double` как значение побитового исключающего ИЛИ первых 4 байтов и вторых 4-х байтов битовых представлений (следует воспользоваться вспомогательными методами классов-оберток).

Задание 4

Добавить в классы векторов реализации методов `Object clone()`. Клонирование должно быть глубоким.

Задание 5

Добавить в класс со статическими методами обработки векторов реализацию метода `Vector unmodifiableVector(Vector vector)`, возвращающего ссылку на экземпляр неизменяемой оболочки указанного вектора. Для этого, соответственно, написать класс-декоратор для типа `Vector`, выбрасывающий исключение `UnsupportedOperationException` в случае попытки изменения состояния вектора.

Лабораторная работа №7

Задание на лабораторную работу

В процессе написания тестовых заданий ознакомиться с общими принципами создания многопоточных приложений.

Задание 1

Создать два класса нитей `WriteThread` и `ReadThread` (наследуют от класса `Thread`), взаимодействующих с помощью промежуточного объекта типа `Vector`. Первая нить последовательно заполняет вектор (изначально он заполнен нулями) произвольными различными величинами (например, случайными), отличными от нуля. Каждый раз, когда она помещает значение в вектор, она выводит на экран сообщение вида "`Write: 100.5 to position 3`". По достижении конца вектора нить заканчивает свое выполнение. Вторая нить последовательно считывает значения из вектора и выводит их на экран сообщениями вида "`Read: 100.5 from position 3`". По достижении конца вектора нить заканчивает свое выполнение.

В методе `main()` следует создать 3 участвующих в процессе объекта и запустить нити на выполнение. Запустите программу несколько раз. Попробуйте варьировать приоритеты нитей.

Задание 2

Создайте два (новых!!!) модифицированных класса нитей `SequentalWriter` и `SequentalReader` (реализуют интерфейс `Runnable`), обеспечивающих последовательность операций чтения-записи (т.е. на экран сообщения выводятся в порядке write-read-write-read-…) независимо от приоритетов потоков.

Для этого потребуется описать некий вспомогательный класс, объект которого будет использоваться при взаимодействии нитей. По своей идее класс будет напоминать пример из лекции (класс `Keeper`).

Задание 3

Добавить в класс со статическими методами обработки векторов реализацию метода `Vector synchronizedVector(Vector vector)`, возвращающего ссылку на оболочку (образец проектирования Декоратор) указанного вектора, безопасную с точки зрения многопоточности.

Лабораторная работа №8

Задание на лабораторную работу

В процессе написания тестовых заданий ознакомиться с базовыми принципами создания апплетов, технологией Swing и обработкой событий.

Задание

Написать и продемонстрировать (в html) работу апплета (Swing), работающего как калькулятор векторов и имеющего следующую функциональность.

На форме должно присутствовать поле ввода, куда вводится длина векторов, и кнопка, по нажатию которой создаются вектора и на форме создаются редакторы для элементов векторов.

Также на форме должны присутствовать две кнопки. По нажатию первой на форме также должен появляться результат скалярного умножения векторов. По нажатию второй на форме должен появляться результат сложения векторов.

В случае некорректного ввода значений в поля редакторов должны выводиться окна с сообщением об ошибке.

Дизайн и выбор конкретных визуальных компонентов для реализации задания могут быть произвольными, однако внешний вид программы не должен вызывать острого отвращения.

Все классы графического приложения следует разместить в новом пакете.

При выполнении задания на оценку «хорошо» можно ограничить максимальную длину векторов. При выполнении задания на оценку «отлично» допускаются только естественные ограничения (допустимые значения типа int).

Дополнительное задание: реализовать функцию сохранения редактируемых и получаемых в результате сложения векторов в файлы, а также считывания из таких файлов в редактируемые вектора. Выбор имени файла следует осуществлять через дополнительный диалог. Также следует проверять соответствие длины считываемого вектора и текущего редактируемого вектора.

Лабораторная работа №9

Задание на лабораторную работу

Ознакомиться с возможностями механизма рефлексии и нововведениями Java5.

Задание 0

Если это не было сделано ранее, добавить в класс `LinkedVector` (класс векторов, реализованный на основе связного списка) конструктор, в котором указывается начальная длина вектора (элементы при этом должны заполняться значениями по умолчанию).

Задание 1

Если вы этого еще не сделали, изменить реализации некоторых классов в пакете так, чтобы они стали вложенными. Подумайте, для каких классов это имеет смысл и позволяет повысить эффективность и простоту программы.

Задание 2

В классе `Vectors` перегрузить фабричный метод `createInstance`, чтобы он кроме длины вектора получал ссылку типа `Vector`, по которой средствами рефлексии определял реальный класс объекта, находил в нем конструктор и создавал объект не средствами фабрики, а средствами рефлексии и того же класса, что и переданный параметр. Если конструктор с единственным параметром типа `int` отсутствует, то следует использовать фабрику (вызвать предыдущую версию метода).

Заменить вызовы `createInstance` на вызовы новой версии так, чтобы создавались объекты того же типа, что и первый операнд операции.

Задание 3

Исправить код пакета векторов (интерфейс Vector и реализующие его классы) так, чтобы вектора могли использоваться в стиле for-each цикла `for`.

Задание 4

Добавить в классы векторов версии конструкторов, имеющие параметрами перечисленные элементы вектора (с использованием аргументов переменной длины).

Задание 5

Исправить метод порождения векторов в классе `Vectors`, использующий рефлексию, с учетом параметризованности классов механизма рефлексии.

Задание 6

Написать клиентскую часть сетевого приложения, формирующую 2 вектора (считывающую их из консоли), устанавливающую соединение с сервером и отправляющую вектора на сервер. После этого от сервера может быть получено либо число, являющееся результатом действия, либо сериализованная форма объекта исключения. Результат следует вывести в консоль.

Задание 7

Написать серверную часть сетевого приложения, прослушивающую порт, а при подключении получающую 2 вектора, выполняющую их скалярное умножение и возвращающую результат клиенту. В случае возникновения исключения оно сериализуется и отправляется клиенту. Сервер должен быть последовательным.