Министерство образования Республики Беларусь
Учреждение образования «Витебский государственный
университет имени П.М. Машерова»
Кафедра прикладного и системного программирования

С.А. Ермоченко, Е.А. Корчевская

ПЛАТФОРМЕННО НЕЗАВИСИМЫЙ ЯЗЫК ПРОГРАММИРОВАНИЯ JAVA

Методические рекомендации по выполнению лабораторных работ

Витебск
ВГУ имени П.М. Машерова
2022

УДК 004.432(075.8)
ББК 32.973.22я73
Е74

Печатается по решению научно-методического совета учреждения образования «Витебский государственный университет имени П.М. Машерова». Протокол № 1 от 05.10.2022.

Авторы: заведующий кафедрой прикладного и системного программирования ВГУ имени П.М. Машерова, кандидат физико-математических наук, доцент **С.А. Ермоченко;** доцент кафедры прикладного и системного программирования ВГУ имени П.М. Машерова, кандидат физико-математических наук, доцент **Е.А. Корчевская**

Р е ц е н з е н т :
заведующий кафедрой информационных систем и технологий УО «ВГТУ», кандидат технических наук, доцент *В.Е. Казаков*

Ермоченко, С.А.

Е74 Платформенно независимый язык программирования Java : методические рекомендации по выполнению лабораторных работ / С.А. Ермоченко, Е.А. Корчевская. – Витебск : ВГУ имени П.М. Машерова, 2022. – 51 с.

В методических рекомендациях приведены ход выполнения лабораторных работ по различным темам, помогающим студентам освоить основы платформенно независимого языка программирования Java, краткие теоретические сведения и различные примеры, демонстрирующие те или иные возможности языка программирования Java.

Предназначается для студентов специальностей «Прикладная математика» и «Прикладная информатика» (дисциплины «Разработка кросс-платформенных приложений» и «Промышленное программирование»), «Программное обеспечение информационных технологий» (дисциплина «Объектно-ориентированные технологии программирования и стандарты проектирования»), «Информационные системы и технологии (в здравоохранении)» (дисциплина «Объектно-ориентированное проектирование и программирование»), «Компьютерная безопасность» (дисциплина «Программирование на Java»).

УДК 004.432(075.8)
ББК 32.973.22я73

С О Д Е Р Ж А Н И Е

ВВЕДЕНИЕ

В данных методических рекомендациях приведены краткие теоретические сведения о принципах работы платформенно независимых языков программирования в целом и по языку программирования Java в частности. Подробно рассматриваются особенности компиляции и запуска Java-приложений при помощи виртуальной машины Java. Рассматриваются инструменты автоматизации сборки проектов Maven и модульного тестирования JUnit. Также приводятся теоретические сведения и практические задания по объектно-ориентированному программированию в Java. Приводятся примеры программ, демонстрирующие те или иные возможности языка программирования. Приводятся методические рекомендации по выполнению лабораторных работ по различным темам, позволяющим закрепить базовые знания о языке программирования Java.

По охвату тем в методических рекомендациях рассматриваются начальные сведения о синтаксисте языка программирования. Немного подробнее рассматривается работа с утилитами командной строки для компиляции, запуска и упаковки в исполняемые архивы Java-классов, исполнения модульных тестов. Так как язык программирования Java изучается студентами после базовых курсов по программированию, в которых изучается язык программирования C++, то общий синтаксис языка Java подробно не рассматривается. В основном, внимание уделяется тем отличиям этих двух языков, которые касаются применения объектно-ориентированного программирования (отсутствие в Java указателей, отсутствие в Java множественного наследования и концепция абстрактных классов и интерфейсов, заменяющая множественное наследование и др.).

Также в приведённых рекомендациях приводится несколько лабораторных работ по изучению стандартных классов языка программирования Java, позволяющих выполнять обработку строк, работать с коллекциями и потоками ввода и вывода.

Материал соответствует отдельным темам рабочих программ курсов: «Разработка кросс-платформенных приложений», «Промышленное программирование» (специальности «Прикладная математика» и «Прикладная информатика»), «Объектно-ориентированные технологии программирования и стандарты проектирования» (специальность «Программное обеспечение информационных технологий»), «Объектно-ориентированное проектирование и программирование» (специальность «Информационные системы и технологии (в здравоохранении)»), «Программирование на Java» (специальность «Компьютерная безопасность»).

ЛАБОРАТОРНАЯ РАБОТА № 1
БАЗОВЫЕ ПОНЯТИЯ И МОДУЛЬНОЕ ТЕСТИРОВАНИЕ

1.1. Теоретические сведения

Платформенно независимый (кроссплатформенный) язык программирования – это язык программирования, позволяющий разрабатывать программы под различные платформы (операционные системы).

Различают языки программирования, являющиеся кроссплатформенными на уровне исходного кода (примером может служить язык программирования C++) и кроссплатформенными на уровне готовой программы.

Кроссплатформенность на уровне исходного кода означает, что одну и ту же программу можно, без внесения изменений в исходный текст, компилировать под разные платформы. Например, следующую программу на языке программирования C++:

```
#include <iostream>
int main(int argc, char* argv[])
{
    std::cout << "Example of console program." << std::endl;
    std::cout << "Command line arguments:" << std::endl;
    for(int i = 0; i < argc; i++)
    {
        std::cout << i << "-th argument is \"" << argv[i] << "\""
                                                    << std::endl;
    }
}
```

можно скомпилировать под операционную систему семейства Windows с помощью компилятора, встроенного в IDE Visual Studio с помощью следующей команды в консоли (если исходный код будет сохранён в файле с именем `C:\Users\User\main.cpp`):

```
C:\Users\User>cl main.cpp
```

В результате компиляции будет создан исполняемый файл `main.exe`, который может быть запущен, например, следующим образом:

```
C:\Users\User>main.exe abc "def ghi" jkl
Example of console program.
Command line arguments:
0-th argument is "main.exe"
1-th argument is "abc"
2-th argument is "def ghi"
3-th argument is "jkl"
```

Эта же программа может быть скомпилирована под операционную систему семейства Linux с помощью компилятора GCC с помощью следующей команды в консоли (если исходный код будет сохранён в файле с именем `/home/user/main.cpp`):

```
user@laptop:~$ g++ main.cpp -o main.out
```

В результате компиляции будет создан исполняемый файл `main.out`, который может быть запущен, например, следующим образом:

```
user@laptop:~$ ./main.out abc def\ ghi jkl
Example of console program.
Command line arguments:
0-th argument is "./main.out"
1-th argument is "abc"
2-th argument is "def ghi"
3-th argument is "jkl"
```

Но при этом ни исполняемый файл `main.exe` не может быть запущен под операционной системой Linux, ни исполняемый файл `main.out` не может быть запущен под операционной системой Windows.

При этом даже кроссплатформенность на уровне исходного кода не гарантирует полную переносимость исходного кода программ между различными операционными системами. Например, если программа использует функции операционной системы Windows, подключая заголовочный файл `windows.h`, то такая программа не может быть даже скомпилирована под операционной системой Linux. Для полноценной разработки платформенно независимых на уровне исходного кода приложений на языке программирования C++ необходимо использовать специальные библиотеки, которые гарантируют корректную переносимость исходного кода программ. Примером такой библиотеки может быть библиотека QT.

Говоря о платформенной независимости необходимо также рассмотреть различные способы трансляции исходного текста программы.

Выше уже рассмотрено понятие компиляции. Схематично этот процесс может быть представлен следующим рисунком:

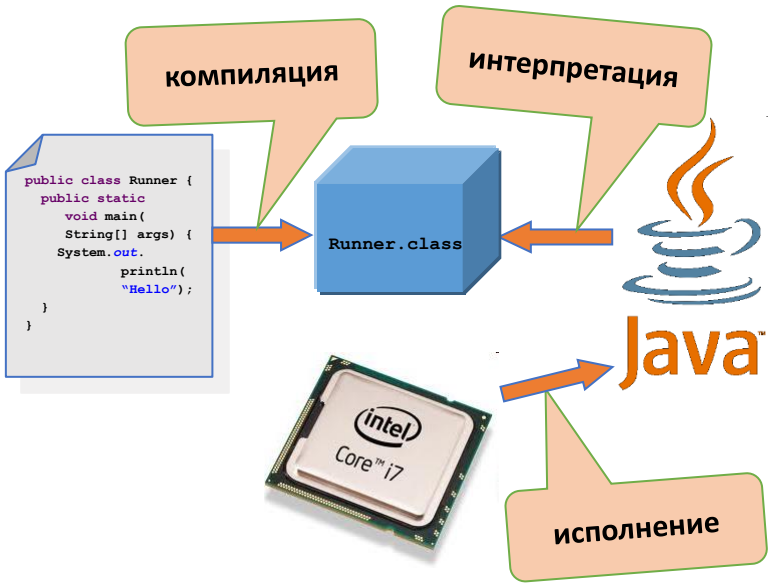
Из рисунка видно, что результатом компиляции является исполняемый файл, содержащий инструкции процессора. Исполнением скомпилированной программы также занимается непосредственно процессор. Таким образом исполняемый файл привязан к набору инструкций исполнителя (процессора), поэтому кроссплатформенность (независимость от исполнителя) обеспечена в таком случае быть не может. Но у компилируемых языков программирования нет необходимости в использовании дополнительного промежуточного программного обеспечения, что делает сам процесс исполнения программы максимально простым и быстрым.

Другим способом трансляции текста программ является его интерпретация. Примером интерпретируемого языка программирования может служить язык программирования JavaScript. Схематически процесс исполнения программы можно представить следующим рисунком:

Из этого рисунка видно, что исполнителем программы является другая программа (браузер Firefox), которая, в свою очередь, исполняется процессором. В таком случае мы можем обеспечить кроссплатформенность программы за счёт разработки программы-интерпретатора (в данном примере браузера) на всех целевых платформах. Но минусом такого подхода является необходимость интерпретатором анализа исходного текста программы, разбор синтаксиса языка программирования, на котором написана программа, при каждом её запуске. А так как эксплуатация программы не связана с её модификацией, то такой процесс имеет достаточно большие накладные расходы. Если пытаться с помощью интерпретируемых языков программирования решать сложные ресурсоёмкие задачи, то потери на накладных расходах могут оказаться достаточно большими, чтобы не позволить обеспечить требуемый уровень производительности.

Для интерпретируемых языков программирования нельзя говорить о переносимости программ на уровне исходного кода или на уровне готовой программы, так как в этих языках программирования готовая программа и есть только её исходный код.

Рассмотрим теперь особенности трансляции программ на языке программирования Java и способ обеспечения её кроссплатформенности. Схематически этот процесс можно проиллюстрировать следующим рисунком:



Из рисунка видно, что исходный текст компилируется в некоторый промежуточный байт-код, который затем интерпретируется с помощью отдельной программы-интерпретатора, которая носит название виртуальной Java-машины. Предварительная компиляция позволяет быстрее организовать процесс исполнения интерпретатором, так как исключается необходимость анализа текста исходной программы, а структура байт-кода позволяет интерпретировать его со скоростью, сопоставимой со скоростью исполнения машинных команд процессором.

1.2. Ход лабораторной работы

1. Ознакомьтесь с описанием компилятора командной строки языка программирования Java:

Для компиляции программ, написанных на языке программирования Java, используется компилятор командной строки `javac`. Общая схема использования компилятора:

```
javac [<опции>] <файлы>
```

Здесь в качестве `<файлы>` через пробел перечисляются имена файлов с расширением «`.java`». Раздел `[<опции>]` необязателен, то есть может быть опущен, но если необходимо, то могут использоваться дополнительные опции. Нижеперечисленные самые часто используемые из них:

`-classpath <путь>`	путь к папке или файлу с расширением «`.jar`», в которых находятся скомпилированные классы, используемые в компилируемой программе.
`-cp <путь>`	то же, что и опция `-classpath`

`-sourcepath <путь>`	путь к папке, в которой находятся компилируемые файлы с расширением «`.java`» (при использовании пакетов относительно этой папки будут отсчитываться все пакеты, сама папка является неименованным пакетом); если параметр пропущен, то в качестве папки используется текущая папка командной строки.
`-d <путь>`	путь к папке, в которую будут помещены скомпилированные файлы с расширением «`.class`»; если параметр пропущен, то в качестве папки используется текущая папка командной строки.
`-encoding <кодировка>`	кодировка исходных файлов с расширением «`.java`»

2. Сохраните приведённый ниже класс в кодировке KOI8-R и скомпилируйте его (то есть компиляцию нужно выполнять с ключом `-encoding KOI8-R`).

```
import java.io.BufferedWriter;
import java.io.OutputStreamWriter;
import java.io.IOException;
import java.lang.NumberFormatException;

public class Runner {
  private static final String DEFAULT_ENCODING = "KOI8-R";
  private static String encoding = null;
  private static BufferedWriter writer = null;
  public static void out(String str) throws IOException {
    if(writer == null) {
      if(encoding == null) {
        encoding = DEFAULT_ENCODING;
      }
      writer = new BufferedWriter(new OutputStreamWriter(System.out,
                                                         encoding));
    }
    writer.write(str);
    writer.flush();
  }
  public static void main(String args[]) throws IOException {
    try {
      if(args.length > 0) {
        int start = 0;
        if("-encoding".equals(args[0])) {
          if(args.length > 1) {
            encoding = args[1];
            if(args.length > 2) {
              start = 2;
            } else {
              out("Программа выводит в консоль последовательность\n");
```

```
        out("русских букв, заданных порядковыми номерами в\n");
        out("алфавите.\n\n");
        out("Использование:\n");
        out("\tjava Runner [-encoding <encoding>] <numbers>\n\n");
        out("где:\n\n");
        out("\t-encoding <encoding>  кодировка символов,\n");
        out("\t                      используемая при выводе в\n");
        out("\t                      консоль\n\n");
        out("\t<numbers>             Последовательность целых\n");
        out("\t                      чисел в диапазоне от 1\n");
        out("\t                      до 32. Каждое число – это\n");
        out("\t                      порядковый номер буквы в\n");
        out("\t                      русском алфавите\n\n");
        return;
      }
    } else {
      out("too few parameters, try to run program\n");
      out("without parameters to get help.\n\n");
      return;
    }
  }
  try {
    for(int i = start; i < args.length; i++) {
      int n = Integer.parseInt(args[i]);
      if(1 <= n && n <= 32) {
        char letter = (char)('A' + n - 1);
        out(String.valueOf(letter));
      } else {
        throw new NumberFormatException();
      }
    }
  } catch(NumberFormatException e) {
    out("Incorrect number. Run class without\n");
    out("parameters to get help.\n\n");
    out("Некорректное число. Запустите класс\n");
    out("с параметром\n");
    out("-encoding <кодировка консоли>\n");
    out("и без других параметров,\n");
    out("чтобы получить справку\n\n");
  }
} else {
  out("Program write to the console sequence of the Russian\n");
  out("letters which have been set by serial numbers in the\n");
  out("alphabet.\n\n");
  out("Usage:\n");
  out("\tjava Runner [-encoding <encoding>] <numbers>\n\n");
  out("where:\n\n");
  out("\t-encoding <encoding>  Specify character encoding\n");
  out("\t                      used by console output\n\n");
  out("\t<numbers>             Sequence of integer numbers\n");
  out("\t                      in the range from 1 to 32.\n");
  out("\t                      Each number is a serial\n");
  out("\t                      number of a letter in the\n");
  out("\t                      Russian alphabet\n\n");
```

```
      }
    } catch(IOException e) {
      System.err.println("specified encoding");
      System.err.println("is not supported");
    }
  }
}
```

При компиляции обратите внимание, какая папка в командной строке является текущей, и нужно ли указывать ключи `-sourcepath` или `-d`. Также обратите внимание, что класс описан в пакете по умолчанию. Ключ `-classpath` при компиляции этого класса использовать не нужно, так как класс не имеет зависимостей от каких-либо других классов, кроме стандартных классов языка программирования Java.

3. Ознакомьтесь с описанием запуска виртуальной машины Java в командной строке:

Для запуска программ, написанных на языке программирования Java, используется виртуальная Java-машина, которая из командной строки запускается командой `java`. Общая схема использования виртуальной машины для запуска некоторого класса (содержащего метод `main`):

```
java [<опции>] <класс> [<аргументы>]
```

Здесь в качестве `<класс>` указывается полное имя класса (включая название пакета) или только имя класса, если он описан в пакете по умолчанию. При этом задаётся имя класса, а не имя файла с расширением «`.class`», содержащего скомпилированный класс. В качестве `[<аргументы>]` указывается список строк, разделённых пробелами, которые передаются в виде массива строк в метод `main` запускаемого класса в качестве параметра. Аргументы являются необязательными. В необязательном разделе `[<опции>]` могут использоваться дополнительные опции. Ниже перечисленные самые часто используемые из них:

`-classpath <путь>`	путь к папке или файлу с расширением «`.jar`», в котором будет осуществляться поиск запускаемого класса (при использовании пакетов, относительно этой папки будут отсчитываться все пакеты)
`-cp <путь>`	то же, что и опция `-classpath`
`-Xms<N><U>`	начальный размер кучи
`-Xmx<N><U>`	максимальный размер кучи

`-Xss<N><U>`	размер стека потока исполнения

в опциях `-Xms`, `-Xmx` и -Xss обозначения `<N>` – это число, а `<U>` – единица измерения, например, `k` (килобайт), `m` (мегабайт) или `g` (гигабайт)

Запуск класса `Abc` с выделением 64 мегабайт на начальный размер кучи, 2 гигабайт на максимальный размер кучи и 256 килобайт под стек может быть выполнен следующей командой:

```
java –Xms64m –Xmx2g –Xss256k Abc
```

4. Запустите скомпилированный класс `Runner`, передавая ему необходимые аргументы командной строки. При запуске класса обратите внимание, какая папка в командной строке является текущей, и нужно ли указывать ключ `-classpath`. Помните, что в командной строке операционной системы Windows используется для вывода кодировка CP866. С помощью класса `Runner` выведите в консоль свою фамилию.

5. Ознакомьтесь с примером вычисления выражения на языке программирования Java, скомпилируйте и запустите его:

```
package by.vsu;

public class Runner {
    /* Метод для вычисления простого выражения.
     * Ключевое слово static позволит вызвать метод,
     * не создавая объект класса Runner
     * Ключевое слово public позволит вызвать метод
     * из другого класса */
    public static double expression(int a, double b) {
        return b / a;
    }
    public static void main(String[] args) {
        int a = Integer.parseInt(args[0]);
        double b = Double.parseDouble(args[1]);
        System.out.println(expression(a, b));
    }
}
```

6. Для тестирования работы отдельных методов классов, разработанных на языке программирования Java, используется специальная библиотека JUnit. Эта библиотека позволяет проводить модульное тестирование Java-классов. Для этого необходимо разработать серию тестов, в которых вызывается тестируемый метод с определёнными тестовыми данными, и указывается ожидаемый результат:

```
package by.vsu;

import org.junit.Assert;
import org.junit.Test;

/* Набор тестов. Как правило, для одного метода или класса
 * создаётся отдельный набор тестов, помещаемый в отдельный
 * класс. Имя этого класса может быть произвольным, но
 * желательно, чтобы оно отображало, какой метод или класс
 * будет тестироваться */
public class ExpressionTestCase {
    /* каждый метод, начинающийся с аннотации @Test
     * представляет собой один тест, в котором, как
     * правило, представлен один набор тестовых данных */
    @Test
    public void testDividingBiggerNumberBySmaller() {
        Assert.assertEquals(
            2.5, // ожидаемый результат
            Runner.expression(2, 5), // вызов тестируемого методы
            0.0000000001 // для вещественных чисел погрешность сравнения
        );
    }
    @Test
    public void testDividingSmallerNumberByBigger() {
        Assert.assertEquals(
            0.4,
            Runner.expression(5, 2),
            0.0000000001
        );
    }
    @Test
    public void testDividingPositiveNumberByZero() {
        Assert.assertEquals(
            Double.POSITIVE_INFINITY,
            Runner.expression(0, 2),
            0.0000000001
        );
    }
    @Test
    public void testDividingNegativeNumberByZero() {
        Assert.assertEquals(
            Double.NEGATIVE_INFINITY,
            Runner.expression(0, -2),
            0.0000000001
        );
    }
}
```

Тесты обычно пишутся перед реализацией самого тестируемого метода или класса. Такой подход позволяет сначала декларировать, какое поведение метода ожидается, а потом реализовывать метод так, чтобы его поведение соответствовало ожидаемому поведению. Кроме того, при модифика-

ции приложения, ранее написанные тесты позволяют провести так называемое регрессионное тестирование. То есть проверить, не перестали ли корректно работать классы и методы, корректно работавшие до модификации.

После разработки набора тестов, тестовые классы также нужно скомпилировать. Для этого в classpath при компиляции добавляются следующие библиотеки: `junit.jar` и `hamcrest-core.jar` (скопируйте представленные преподавателем файлы папку проекта). Скомпилируйте представленный пример с помощью следующей команды в консоли, выполненной в папке проекта (относительно которой определяются пакеты). Пусть на диске `D:` создан каталог `D:\java\project`, в него необходимо поместить следующие файлы:

- `D:\java\project\by\vsu\ExpressionTestCase.java`
- `D:\java\project\by\vsu\Runner.java`
- `D:\java\project\hamcrest-core.jar`
- `D:\java\project\junit.jar`

Тогда команда компиляции теста будет выглядеть так:

```
D:\java\project>javac -cp .;junit.jar;hamcrest-core.jar
                                          by\vsu\ExpressionTestCase.java
```

Команда запуска тестов будет выглядеть следующим образом:

```
D:\java\project>java -cp .;junit.jar;hamcrest-core.jar
                     org.junit.runner.JUnitCore by.vsu.ExpressionTestCase
```

А в результате запуска должно быть выдано примерно следующее:

```
JUnit version 4.12
....
Time: 0,006

OK (4 tests)
```

Если какие-то тесты не будут пройдены. Сообщения об ошибке будут выведены в консоль.

7. Ознакомьтесь с описанием и примером использования инструмента сборки проекта Maven:

Проблема использования компилятора языка программирования Java в командной строке заключается в том, что при работе с большими проектами, в которых количество различных классов может исчисляться сотнями, и даже тысячами, компилировать каждый класс, указывать опции компилятора, описывая зависимости классов от различных библиотек, является очень трудоёмким процессом, не связанным с прямой обязанностью программиста разрабатывать алгоритмы решения поставленных задачи и писать исходные коды программ, реализующих эти алгоритмы. Эта проблема

усугубляется ещё и тем, что в более-менее крупных проектах может использоваться большое количество сторонних библиотек, но при этом каждой библиотеке может потребоваться ещё несколько дополнительных библиотек для своей работы. Также трудности может вызывать контроль версий одной и той же используемой библиотеки.

Помимо компиляции и запуска приложения с указанием используемых библиотек, помимо запуска большого количества модульных тестов, в крупных проектах, особенно связанных с разработкой серверного программного обеспечения и веб-приложений, часто возникает необходимость при запуске проекта решать множество сопутствующих задач, связанных с подготовкой нужной структуры каталогов и созданием необходимого окружения для запуска проекта, созданием и размещением в нужных подкаталогах конфигурационных файлов, подготовкой базы данных для постоянного хранения обрабатываемой информации и т.д. и т.п.

Для решения большей части из описанных выше проблем используется специальный класс утилитарного программного обеспечения, который носит название системы автоматизации сборки проекта. Примерами таких систем для разработки программного обеспечения на языке программирования Java и с применением Java технологий являются инструменты Ant, Maven, Gradle.

Рассмотрим работу с утилитой Maven.

Прежде всего необходимо создать проект Maven. По сути, проектом Maven является отдельный каталог на жёстком диске, в котором создаётся специальная структура подкаталогов для хранения файлов исходного кода. Создать такой проект можно практически в любой IDE, поддерживающей работу с языком программирования Java. Наиболее популярными в настоящее время являются следующие IDE:

- IntelliJ IDEA
- Eclipse
- NetBeans

Выбор предпочтительной IDE – дело вкуса каждого разработчика, однако при работе над общим проектом все члены команды стараются использовать одинаковую IDE. При этом от проекта к проекту IDE может меняться, поэтому у разработчика не должно возникать проблем при необходимости перейти от использования одной IDE к другой.

При создании Maven проекта в IDE либо необходимо искать такой тип проекта, либо создавать обычный Java-проект, выбирая при создании систему сборки проекта Maven. Также в большинстве IDE есть функция преобразования существующего проекта в проект Maven. Однако Maven – это не просто инструмент, доступный в IDE. Maven можно использовать как самостоятельную утилиту, выполняя сборку проекта с его помощью в командной строке. Так, например, если установлен Maven как отдельный инструмент (скачать его можно на официальном сайте

https://maven.apache.org/), то можно создать Maven проект в консоли, используя команду:

```
C:\Users\User>mvn archetype:generate
```

После этой команды утилита переходит в интерактивный режим, в котором у пользователя запрашивается информация о создаваемом проекте. На часть вопросов можно не давать ответ, соглашаясь с предложенным вариантом. Но некоторые вопросы являются обязательными.

Каким бы образом ни создавался Maven проект, у этого проекта должны быть определены 3 параметра, которые также принято называть Maven координатами. Это:

- Group ID
- Artifact ID
- Version

Эти координаты задаются для каждого создаваемого проекта, и эти же координаты имеет любая библиотека, работа с которой может осуществляться через Maven.

Group ID – это строка, определяющая группу похожих проектов или библиотек, которые связаны общим смыслом, общей целью или просто общей компанией-разработчиком. Для решения проблемы возможного конфликта имён классов, все классы в проектах Java размещаются в пакетах, которые начинаются с доменного имени разработчика. Например, пакеты в рамках проектов ВГУ имени П. М. Машерова, который владеет доменным именем vsu.by, следует начинать с by.vsu.*. Этот общий префикс и можно использовать в качестве Group ID, или начинать Group ID с этого префикса.

Artifact ID – это строка с названием проекта. Как правило, эта строка также будет добавляться к названию пакета, поэтому желательно в данной строке использовать только маленькие латинские буквы, цифры и знак подчёркивания. Если в ВГУ имени П. М. Машерова будет выполняться проект под названием «Digital University», и для него будет выделен домен третьего уровня digital-university.vsu.by, то все пакеты в рамках этого проекта следует начинать с префикса by.vsu.digital_university.*.

Таким образом Group ID этого проекта будет «by.vsu». А Artifact ID – «digital_university».

Version – версия проекта. Задача версионирования в рамках проекта или библиотеки сама по себе очень непростая. Существует несколько различных подходов, которые в данных методических рекомендациях рассматриваться не будут. Как правило, у всех вновь создаваемых проектов Version равна «1.0-SNAPSHOT».

После создания Maven проекта основными отличительными его чертами является структура подкаталогов и файл с настройками проекта.

Рассмотрим структуру каталогов Maven проекта:

```
digital_university/
  src/
    main/
      java/
      resources/
    test/
      java/
      resources/
  pom.xml
```

Помимо каталога `src` в проекте можно создавать и другие каталоги для разных целей. Но весь код на языке программирования Java должен размещаться в каталоге `src`. Подкаталог `main` предназначен для хранения основного кода приложения. Подкаталог `test` – для хранения классов с модульными тестами. Подкаталог `java` внутри каталогов `main` и `test` предназначен для хранения непосредственно Java-классов (файлов с расширением «`.java`»). Подкаталог resources внутри каталогов main и test предназначен для хранения вспомогательных файлов, так называемых ресурсов. Обычно это файлы с расширением «`.properties`» или другие конфигурационные файлы. Но также это могут быть изображения и другие файлы, работа с которыми производится через специальный стандартный класс `ClassLoader` и его методы `getResourceAsStream()` или `getSystemResourceAsSteram()`. В данных методических рекомендациях работа с ресурсами рассматриваться не будет, поэтому данные каталоги можно удалять.

Если в нашем проекте мы создаём класс `by.vsu.digital_university.Discipline` и класс для модульных тестов `by.vsu.digital_university.DisciplineTest`, то структура каталогов и файлов будет следующей:

```
digital_university/
  src/
    main/
      java/
        by/
          vsu/
            digital_university/
              Discipline.java
    test/
      java/
        by/
          vsu/
            digital_university/
              DisciplineTest.java
  pom.xml
```

При этом физически классы `Discipline` и `DisciplineTest` находятся в разных каталогах. Но логически, с точки зрения компилятора и виртуальной машины Java, они будут находится в одном пакете.

Рассмотрим теперь файл с настройками Maven проекта. Этот файл описывает объектную модель проекта (Project Object Model – POM) и носит имя «pom.xml». Как видно из расширения файла, он является файлом в специальном текстовом формате XML и может редактироваться любым текстовым редактором. Во вновь созданном проекте pom.xml может иметь примерно следующее содержимое:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                             https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>by.vsu</groupId>
    <artifactId>digital_university</artifactId>
    <version>1.0-SNAPSHOT</version>
</project>
```

Как видно, помимо общего заголовка файла, смысл которого мы рассматривать не будет, и уже рассмотренных ранее координат проекта в этом файле других настроек нет. Но это лишь минимальный набор настроек. Как правило, в этот файл добавляют ещё секцию со свойствами (см. ниже.). Отдельным свойством определяется кодировка файлов с исходным кодом. И отдельными свойствами – версия используемого компилятора. Если опустить версию компилятора, Maven будет использовать версию 1.5, которая уже считается устаревшей и практически не используется в проектах. Дополненный файл будет выглядеть следующим образом:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                             https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>by.vsu</groupId>
    <artifactId>digital_university</artifactId>
    <version>1.0-SNAPSHOT</version>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.target>19</maven.compiler.target>
        <maven.compiler.source>19</maven.compiler.source>
    </properties>
</project>
```

После создания проекта его сборку и запуск можно осуществлять в консоли с помощью команд Maven, либо средствами IDE. Сборка в консоли может быть полезна при разворачивании проекта на сервере, на котором, как правило, в принципе не возможна установка приложений с графическим интерфейсом пользователя. Взаимодействие с сервером осуществляется

только по сети, а управление – с помощью консольных команд, передаваемых по специальному текстовому протоколу. На начальных этапах изучения языка программирования Java достаточно использовать стандартные возможности IDE.

Для добавления сторонних библиотек в Maven проект, которые называются терминологии Maven называются зависимостями, в файл `pom.xml` добавляется раздел `<dependencies>`. Для поиска нужной библиотеки и её Maven координат, можно воспользоваться специализированными сайтами – Maven репозиториями. Например, `https://search.maven.org/` или `https://mvnrepository.com/`. В таких репозиториях поиск можно осуществлять по Group ID, Artifact ID, по описанию или ключевым словам. При нахождении необходимой библиотеки в репозитории приводятся как Maven координаты библиотеки, так и пример включения этой библиотеки в файл `pom.xml`. Файл настроек проекта с добавленной зависимостью от библиотеки Junit будет выглядеть следующим образом:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                             https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>by.vsu</groupId>
    <artifactId>digital_university</artifactId>
    <version>1.0-SNAPSHOT</version>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.target>19</maven.compiler.target>
        <maven.compiler.source>19</maven.compiler.source>
    </properties>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
        </dependency>
    </dependencies>
</project>
```

После добавления зависимости при сборке проекта (которая автоматически осуществляется перед запуском проекта или перед запуском модульных тестов), Maven автоматически скачивает из репозитория необходимые зависимости и подключает их к проекту.

По аналогии с описанным выше примером создания и настройки проекта, создайте в IDE новый Maven проект с классом, вычисляющем значение выражения, из пункта 5, и с классом, содержащим модульные тесты, из пункта 6. Выполните запуск модульных тестов с помощью встроенных средств IDE.

8. Ознакомьтесь с архиватором командной строки, предназначенном для упаковки Java-программ:

Архив с расширением «.jar» представляет собой обычный архив, использующий алгоритм сжатия ZIP, но предназначен он для архивирования классов некоторого приложения, написанного на языке программирования Java, при этом виртуальная машина умеет запускать такие приложения непосредственно из архива. Для создания jar-архива используется утилита jar. Данная команда умеет не только создавать архивы, но и просматривать содержимое архивов, распаковывать архивы или обновлять содержимое архивов. Здесь мы рассмотрим только создание архивов. Общий формат использования команды для создания архива следующий:

```
jar c[fe] [<архив>] [<запускаемый класс>] <файлы>
```

Описание параметров утилиты:

c	символ указывает, что утилита должна создать архив (для других операций, таких как извлечение файлов или обновление архива, используются другие символы, но здесь мы их не рассматриваем)
f	если вместе с символом c дополнительно указан символ f, это значит, что в параметрах должно быть указано имя файла с расширением «.jar» (необязательный параметр [<архив>]), в который будет сохранён упакованный архив. Если этот параметр не указан (и отсутствует символ f в первом параметре), то содержимое архива выводится в поток вывода по умолчанию (то есть на экран командной строки). Такая возможность может быть использована при перенаправлении вывода команды в некий другой поток
e	если вместе с символом c дополнительно указан символ e, это значит, что в параметрах должно быть указано имя класса (необязательный параметр [<запускаемый класс>]), запускаемого при запуске приложения из этого архива; если имя класса не указывается (и отсутствует символ e в первом параметре), то создаётся обычный архив (например, библиотека, которая может быть подключена к другому приложению)
<архив> <запускаемый класс>	соответствующие имена архива и запускаемого класса должны указываться в том же порядке, в котором следуют символы f и e после символа c
<файлы>	имена добавляемых в архив файлов (через пробел). Это могут быть как файлы с расширением «.class», так и другие файлы. Например, файлы исходного кода с расширением «.java»

9. Упакуйте ранее скомпилированный в пункте 2 класс `Runner` в запускаемый jar-архив, используя утилиту `jar`. Следите за соответствием последовательности символов `f` и `e` в первом параметре утилиты последовательности из имени создаваемого архива и имени запускаемого класса.

10. Ознакомьтесь с особенностями запуска Java-программы, упакованной в jar-архив.

Для запуска программ, написанных на языке программирования Java и упакованных в jar-архив, также используется виртуальная Java-машина, которая из командной строки запускается командой `java`. Общая схема использования виртуальной машины для запуска программы из jar-файла:

```
java [<опции>] -jar <архив> [<аргументы>]
```

Здесь необязательные секции `[<опции>]` и `[<аргументы>]` имеют тот же смысл, что и ранее. Обязательный параметр `<архив>` задаёт имя файла с расширением «`.jar`», из которого и будет запускаться программа.

11. Запустите программу из созданного в пункте 9 архива с запускаемым классом `Runner`, передавая необходимые аргументы командной строки. С помощью этой программы выведите в консоль своё имя.

12. Скомпилируйте приведённый ниже класс.

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Formatter;

public class HashCode {
    public static String md5(String string) {
        MessageDigest digest;
        try {
            digest = MessageDigest.getInstance("md5");
            digest.reset();
            digest.update(string.getBytes());
            byte hash[] = digest.digest();
            Formatter formatter = new Formatter();
            for(int i = 0; i < hash.length; i++) {
                formatter.format("%02X", hash[i]);
            }
            String md5summ = formatter.toString();
            formatter.close();
            return md5summ;
        } catch(NoSuchAlgorithmException e) {
            return null;
        }
    }
}
```

13. Упакуйте скомпилированный класс в jar-архив `md5.jar` (не запускаемый, т. е. ключ e использовать не нужно), такой архив будет являться обычной библиотекой.

14. Скомпилируйте теперь в командной строке ниже следующий класс. Данный класс использует класс `HashCode` из библиотеки `md5.jar`, созданной выше. Для включения зависимостей необходимо использовать ключ `-classpath` при компиляции этого класса.

```
public class MD5Calculator {
    public static void main(String args[]) {
        for(String arg : args) {
            System.out.println("\"" + arg + "\" => " + HashCode.md5(arg));
        }
    }
}
```

15. Запустите скомпилированный класс `MD5Calculator` из командной строки. При запуске класса также необходимо использовать ключ `-classpath`, но в нём необходимо указать как jar-архив, так и текущий каталог, например, так «`.;md5.jar`». Класс `MD5Calculator` считывает аргументы командной строки и подсчитывает хэш-код каждого аргумента по алгоритму MD5. С помощью скомпилированного класса вычислите хэш-коды строк, содержащих дату Вашего рождения и даты рождения других 4-5 человек.

1.3. Самостоятельная работа «Линейные алгоритмы»

Напишите программу для вычисления выражения, соответствующего своему варианту. Значения величин, входящих в выражение, задавайте с помощью аргументов командной строки. Компиляцию и запуск программы осуществляйте в командной строке. Разработайте 2-3 модульных теста и запустите их через консоль.

Варианты задания:

№	Выражение	№	Выражение
1.	$b^2 - 4ac$	2.	$\frac{(a+b)(c-b)}{4}$
3.	$\left(a+\frac{b}{c}\right)\left(a-\frac{c}{b}\right)$	4.	$a^2 - \frac{b^2}{c^2+5}$
5.	$\frac{a+b+c}{abc}$	6.	$\frac{c}{b} + 3(a-b)$

7.	$3c - \frac{a+b}{2}$	8.	$\frac{a+b}{c} - 4c$
9.	$\frac{(a+b)^2}{a-c}$	10.	$\frac{a^2+1}{b - \frac{1}{c^2+1}}$
11.	$\frac{a-b}{2c+1}$	12.	$\frac{1}{a + \frac{1}{b + \frac{1}{c}}}$
13.	$a + \frac{(b-c)^2}{3}$	14.	$(a+b)(a+c)(b+c)$

1.4. Самостоятельная работа «Циклические алгоритмы»

Напишите программу для вычисления суммы ряда, соответствующего своему варианту, с применением указанного цикла. Для сравнения, то же значение вычислите с применением методов класса `Math`. Значение аргумента функции и точность вычисления задавайте с помощью аргументов командной строки. Компиляцию программы осуществляйте в командной строке.

Скомпилированный класс упакуйте в запускаемый jar-архив. Запуск программы из jar-архива осуществляйте из командной строки.

Разработайте 4-5 модульных тестов для вычисления суммы ряда. Функция, стоящая слева от знака приблизительного равенства, может быть использована для вычисления ожидаемого результата теста. Точность вычисления необходимо задавать третьим параметром метода `assertEquals`.

Варианты задания:

№	цикл	ряд	№	цикл	ряд
1.	**`while`**	$e^x \approx \sum_{n=0}^{\infty} \frac{x^n}{n!}$	2.	**`for`**	$e^x \approx \sum_{n=0}^{\infty} \frac{x^n}{n!}$
3.	**`while`**	$\sin x \approx \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}$	4.	**`for`**	$\sin x \approx \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}$
5.	**`while`**	$\cos x \approx \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!}$	6.	**`for`**	$\cos x \approx \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!}$
7.	**`while`**	$\operatorname{arctg} x \approx \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{2n+1}$	8.	**`for`**	$\operatorname{arctg} x \approx \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{2n+1}$

9. **while**	$\sinh x \approx \sum_{n=0}^{\infty} \frac{x^{2n+1}}{(2n+1)!}$	10. **for**	$\sinh x \approx \sum_{n=0}^{\infty} \frac{x^{2n+1}}{(2n+1)!}$
11. **while**	$\cosh x \approx \sum_{n=0}^{\infty} \frac{x^{2n}}{(2n)!}$	12. **for**	$\cosh x \approx \sum_{n=0}^{\infty} \frac{x^{2n}}{(2n)!}$
13. **while**	$\ln \sqrt{\frac{1+x}{1-x}} \approx \sum_{n=0}^{\infty} \frac{x^{2n+1}}{2n+1}$	14. **for**	$\ln \sqrt{\frac{1+x}{1-x}} \approx \sum_{n=0}^{\infty} \frac{x^{2n+1}}{2n+1}$

1.5. Самостоятельная работа «Использование командной строки»

Используя предоставленный преподавателем архив `calc.jar`, содержащий два запускаемых класса (имеющих метод `main`): `Calculator` и `InfixToPolizConverter`, вычислить значение арифметического выражения.

Сам архив также является запускаемым, при этом в качестве точки входа используется класс `Calculator`.

Класс `Calculator` считывает из аргументов командной строки значения переменных в формате `<имя переменной>=<значение переменной>`, разные переменные разделяются пробелом (например: `a=1 b=2 c=3`). Далее класс считывает из стандартного потока ввода командной строки (с клавиатуры, `System.in`) строку, являющуюся неким арифметическим выражением, записанным в обратной польской нотации (ПОЛИЗ), а затем вычисляет это выражение с использованием указанных значений переменных. Результат выводится в стандартный поток вывода командной строки (`System.out`).

Класс `InfixToPolizConverter` считывает из аргументов командной строки ровно один аргумент, который содержит обычное арифметическое выражение, в котором могут использоваться переменные, числа, круглые скобки и операции (+, –, *, /). Пробелы в выражении использовать нельзя. Далее класс преобразует это выражение в обратную польскую нотацию (ПОЛИЗ). Результат выводится в стандартный поток вывода командной строки (`System.out`).

Необходимо с использованием классов из данного архива вычислить выражение из самостоятельной работы 1.3.

Для того чтобы результат, выводимый одной программой в стандартный поток вывода командной строки, автоматически использовать как стандартный поток ввода командной строки (ввод с клавиатуры) для другой программы, можно в командной строке использовать символ `'|'`, позволяющий перенаправить вывод одной программы на вход другой, например:

```
commandA | commandB
```

Здесь программа commandB считывает с клавиатуры некие данные, но вместо ввода этих данных пользователем с клавиатуры ей передаются данные, выводимые на экран программой commandA (при этом данные на экран не выводятся, а сразу поступают во вторую программу).

1.6. Самостоятельная работа «Работа с массивами»

Используя предоставленный преподавателем архив painting.jar, напишите программу, которая строит в окне некоторое изображение, формируемое из отрезков, по следующему правилу. В полярной системе координат заданы две функции (см. свой вариант). Программа для значений координаты φ на отрезке [0; 2π] с шагом Δφ (шаг задаётся через аргументы командной строки) находит точку пересечения луча, соответствующего углу φ, с графиком каждой из двух функций, и проводит через указанные точки отрезок. Для создания окна и построения в созданном окне набора отрезков необходимо вызвать статический метод paint класса Painter из предложенной библиотеки. Объявление данного метода в данном классе следующее:

```
package by.vsu.mf.ai.ssd.painting;

public class Painter {
    public static void paint(int width, int height, double[][][] coords,
                             short[][] colors) { /*...*/ }
}
```

Здесь параметры width и height – ширина и высота клиентской области создаваемого окна.

Параметр coords – 3-мерный массив координат концов отрезков. Элемент этого массива coords[i][j][k]. Индекс i – это индекс отрезка (начиная с 0). Индекс j – это индекс точки-конца отрезка (может быть равен 0 или 1). Индекс k – это индекс координаты точки. Если k = 0, элемент массива определяет координату по оси абсцисс. Если k = 1, элемент массива определяет координату по оси ординат. Центр системы координат располагается в центре создаваемого окна, по оси абсцисс и ординат координаты точек в окне изменяются на отрезке [–1; 1].

Параметр colors – 2-мерный массив компонент цвета. Элемент этого массива colors[i][j]. Индекс i – это индекс отрезка (начиная с 0). Индекс j – это индекс компонента цвета (0 – красная компонента, 1 – зелёная, 2 – синяя). Каждая компонента задаётся целым числом в диапазоне от 0 до 255.

Варианты задания:

№	Первая функция	Вторая функция
1.	$r = \dfrac{2 - \left\lvert \cos\dfrac{\pi + 10\varphi}{4} \right\rvert}{2}$	$r = \dfrac{1}{5}\operatorname{tg}\dfrac{\pi(2 + \sin 5\varphi)}{8}$
2.	$r = \dfrac{1 + \sin\sqrt{\varphi}}{3}$	$r = 1$
3.	$r = 1 - \dfrac{2}{3}\sin^2\left(2\cos\dfrac{\varphi}{2}\right)$	$r = \dfrac{1}{2}$
4.	$r = \dfrac{2 - \lvert\cos 4\varphi\rvert}{4}$	$r = \dfrac{2 - \sin^2(2\cos 4\varphi)}{2}$
5.	$r = \dfrac{1 + \lvert\cos 3\varphi\rvert}{5}$	$r = \dfrac{1 + \sin^2(2\cos 3\varphi)}{2}$
6.	$r = \dfrac{1 - \lvert\cos 2\varphi\rvert}{5}$	$r = \dfrac{1 + \sin^2(2\cos 4\varphi)}{2}$
7.	$r = \dfrac{1 - \lvert\cos 2\varphi\rvert}{5}$	$r = 1$
8.	$r = \dfrac{1 + \cos^2 2\varphi}{5}$	$r = \dfrac{1 + \sin^2 3\varphi}{2}$
9.	$r = \dfrac{1 + \cos^2\varphi}{3}$	$r = \dfrac{1 + \sin^2\varphi}{2}$
10.	$r = 1$	$r = \dfrac{1 + \cos^2\varphi}{3}$
11.	$r = \dfrac{\varphi}{2\pi}$	$r = \dfrac{1 + \cos^2\varphi}{2}$
12.	$r = \dfrac{\varphi}{4\pi}$	$r = \dfrac{1 + \sin^2\varphi}{2}$
13.	$r = \dfrac{1}{3}$	$r = \dfrac{1 + \sin^2\varphi}{2}$
14.	$r = \dfrac{\varphi}{2\pi}$	$r = 1$

Указания: отрезки, формирующие изображение, должны иметь разные, плавно изменяющиеся от отрезка к отрезку, цвета.

ЛАБОРАТОРНАЯ РАБОТА № 2
ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

2.1. Ход лабораторной работы

Изучите пример объектно-ориентированного решения задачи: найти уравнение серединного перпендикуляра к отрезку, заданному координатами своих концов.

Класс `Point`:

```
/** Точка на плоскости */
public class Point {
    protected double x;
    protected double y;
    /** Создаёт точку с заданными координатами
     *  @param x абсцисса точки
     *  @param y ордината точки */
    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }
    /** Получает значение абсциссы точки
     *  @return абсцисса точки */
    public double getX() {
        return x;
    }
    /** Получает значение ординаты точки
     * @return ордината точки */
    public double getY() {
        return y;
    }
    @Override
    public String toString() {
        return "(" + x + "; " + y + ")";
    }
}
```

Класс `Vector`:

```
/** Вектор в плоскости */
public class Vector extends Point {
    /** Создает вектор, соответствующий направленному отрезку
     *  @param segment направленный отрезок */
    public Vector(Segment segment) {
        /* вызов конструктора суперкласса Point */
        super(segment.getBegin().getX() - segment.getEnd().getX(),
              segment.getBegin().getY() - segment.getEnd().getY());
    }
    /** @return длина вектора */
```

```
    public double length() {
        return Math.hypot(x, y);
    }
}
```

Класс Segment:

```
/** Направленный отрезок */
public class Segment {
    private Point begin;
    private Point end;
    /** Создаёт направленный отрезок с началом и концом в указанных точках
     *  @param begin начальная точка
     *  @param end конечная точка */
    public Segment(Point begin, Point end) {
        this.begin = begin;
        this.end = end;
    }
    /** @return начальная точка отрезка */
    public Point getBegin() {
        return begin;
    }
    /** @return конечная точка отрезка */
    public Point getEnd() {
        return end;
    }
    /** @return точка, являющаяся серединой отрезка */
    public Point getMiddlePoint() {
        return new Point((begin.getX() + end.getX()) / 2,
                         (begin.getY() + end.getY()) / 2);
    }
    /** @return длина направленного отрезка */
    public double length() {
        // передача текущего объекта в качестве параметра в
        // конструктор класса Vector
        return new Vector(this).length();
    }
}
```

Класс Line:

```
/** Прямая в плоскости */
public class Line {
    private double a;
    private double b;
    private double c;
    /** Создаёт серединного перпендикуляра к отрезку
     *  @param segment отрезок */
    public Line(Segment segment) {
        /* вызов другого конструктора этого же класса */
        this(
            segment.getMiddlePoint(),
            new Vector(
                new Segment(segment.getMiddlePoint(), segment.getBegin())
            )
```

```
        );
    }
    /** Создаёт прямую, проходящую через точку, перпендикулярно вектору
     * @param p точка, через которую проходит прямая
     * @param n вектор-перпендикуляр к прямой */
    public Line(Point p, Vector n) {
        a = n.x;
        b = n.y;
        c = -(a * p.getX() + b * p.getY());
    }
    /** @return коэффициент перед абсциссой в уравнении прямой */
    public double getA() {
        return a;
    }
    /** @return коэффициент перед ординатой в уравнении прямой */
    public double getB() {
        return b;
    }
    /** @return свободный коэффициент в уравнении прямой */
    public double getC() {
        return c;
    }
    @Override
    public String toString() {
        return "(" + a + ")x + (" + b + ")y + (" + c + ") = 0";
    }
}
```

Класс Runner:

```
public class Runner {
    public static void main(String[] args) {
        /* получение координат точек из аргументов командной строки */
        double x1 = Double.parseDouble(args[0]);
        double y1 = Double.parseDouble(args[1]);
        double x2 = Double.parseDouble(args[2]);
        double y2 = Double.parseDouble(args[3]);

        /* создание точек на основе координат */
        Point point1 = new Point(x1, y1);
        Point point2 = new Point(x2, y2);
        System.out.println("Даны две точки: " + point1 + " и " + point2);
        /* создание отрезка, проходящего через две точки */
        Segment segment = new Segment(point1, point2);
        System.out.println("Расстояние между точками: "
                                          + segment.length());
        /* создание серединного перпендикуляра к отрезку */
        Line line = new Line(segment);
        System.out.print("Уравнение серединного ");
        System.out.print("перпендикуляра к отрезку, ");
        System.out.print("проходящему через данные точки: ");
        System.out.println(line);
    }
}
```

2.2. Самостоятельная работа «Геометрическая задача»

Разработайте классы для решения предложенной задачи (см. свой вариант). При необходимости, дополните существующие классы новыми методами. Разработайте набор JUnit тестов (с использованием IDE) для всех методов классов, разработанных Вами, в том числе и для метода, решающего основную задачу.

Варианты заданий:

1. Постройте квадрат, концы одной диагонали которого имеют координаты $(x_1; y_1)$ и $(x_2; y_2)$ и описанную вокруг него окружность.
2. Определите, лежит ли точка с координатами $(x_0; y_0)$ внутри треугольника, вершины которого расположены в точках $(x_1; y_1)$, $(x_2; y_2)$, $(x_3; y_3)$.
3. Постройте окружность радиуса R, проходящую через точки с координатами $(x_1; y_1)$ и $(x_2; y_2)$.
4. Постройте квадрат, если известно, что некоторые две его вершины расположены в точках $(x_1; y_1)$ и $(x_2; y_2)$. Опишите вокруг полученного квадрата окружность.
5. Даны координаты вершин некоторого четырёхугольника: $(x_1; y_1)$, $(x_2; y_2)$, $(x_3; y_3)$ и $(x_4; y_4)$. Определите, является ли этот четырёхугольник: а) параллелограммом; б) ромбом; в) квадратом?
6. Найдите точки пересечения высот и медиан треугольника, вершины которого расположены в точках $(x_1; y_1)$, $(x_2; y_2)$ и $(x_3; y_3)$.
7. Постройте параллелограмм, если известно, что некоторые три его вершины расположены в точках $(x_1; y_1)$, $(x_2; y_2)$ и $(x_3; y_3)$. Предусмотрите все возможные варианты расположения фигуры.
8. Постройте окружность, вписанную в равносторонний треугольник, если известно, что некоторые две его вершины расположены в точках $(x_1; y_1)$ и $(x_2; y_2)$.
9. Определите, лежит ли треугольник с вершинами в точках $(x_1; y_1)$, $(x_2; y_2)$ и $(x_3; y_3)$ внутри окружности с центром в точке $(x_0; y_0)$ и радиусом R.
10. Постройте прямоугольник, если известно, что описанная вокруг него окружность имеет радиус R, а некоторые две соседние вершины расположены в точках $(x_1; y_1)$ и $(x_2; y_2)$.

2.3. Самостоятельная работа «Динамическое программирование»

Разработайте классы для решения предложенной задачи (см. свой вариант) методом динамического программирования. Разработайте набор JUnit тестов для задачи. Теорию по методу динамического программирования изучить самостоятельно.

Варианты заданий:

1. Дети играют в игру, в ходе которой игроку нужно подняться по лестнице, на каждой из N ступенек которой лежит некоторое число штрафных фишек A_i (i от 1 до N). Если игрок наступает на ступеньку, он забирает себе все штрафные фишки с этой ступеньки. С любой ступеньки игрок может шагнуть на следующую ступеньку или перешагнуть одну ступеньку. Для каждой ступеньки известно число штрафных

фишек, лежащих на ней. Найти такую последовательность ступеней, наступая на которые, игрок соберёт минимальное число штрафных фишек.

Пример:

$N = 6$, $A_1 = 1$, $A_2 = 4$, $A_3 = 5$, $A_4 = 2$, $A_5 = 3$, $A_6 = 6$

Правильным ответом будут последовательности, дающие минимальный штраф 9:

1, 3, 5

или

2, 4, 5.

2. У Вас есть калькулятор с темя кнопками:

1	Прибавить к текущему числу 1
2	Умножить текущее число на 2
3	Умножить текущее число на 3

Найти минимальную последовательность нажатия кнопок на калькуляторе, которая позволяет получить введённое пользователем число N, если в начальный момент текущее число 1.

Пример:

$N = 13$

Последовательность кнопок: 1, 1, 2, 3, 1.

3. Дана шахматная доска размерности N на N клеток ($3 \leqslant N \leqslant 7$). В левой верхней клетке стоит шахматная фигура коня. Эта клетка помечается числом 1. Затем конь может сделать ход на любою непомеченную клетку. После хода клетка, на которую походил конь, помечается числом, на 1 большую числа, которым была помечена клетка, из которой конь сделал свой ход. Для заданного числа N определить последовательность ходов, которая позволит пометить максимальное число клеток на доске.

Пример:

$N = 3$

1	6	3
4		8
7	2	5

4. Дана прямоугольная таблица из N строк и M столбцов. В каждой ячейке таблицы записано некоторое натуральное число $A_{i,j}$ ($1 \leqslant i \leqslant N$, $1 \leqslant j \leqslant M$). Определить путь из левой верхней клетки ($i = 1$, $j = 1$) до правой нижней ($i = N$, $j = M$) с наименьшей суммой чисел на этом пути. При перемещении по таблице можно переходить из текущей клетки на одну клетку вниз или на одну клетку вправо (текущий индекс строки или столбца может увеличиться на единицу, но одновременно два индекса измениться не могут).

Пример:

$N = 3$; $M = 4$

5	12	10	4
11	3	1	9
2	7	8	2

5. Человек должен подняться по лестнице, для каждой из N ступенек которой известна вероятность p_i того, что при наступании на эту ступеньку, она не сломается (i от 1 до N). При подъёме по лестнице человек может шагнуть на следующую сту-

пеньку, или перешагнуть одну или две ступеньки. Найти такую последовательность ступенек, по которым подъём будет наименее опасным (произведение вероятностей p_i соответствующих ступеней будет максимальным).

Пример:

$N = 7$, $p_1 = 0.7$, $p_2 = 0.6$, $p_3 = 0.8$, $p_4 = 0.5$, $p_5 = 0.7$, $p_6 = 0.9$, $p_7 = 0.8$

Правильным ответом будет последовательность, дающая вероятность успешного подъёма по лестнице 0.72:

3, 6.

6. Дано два сосуда с объёмом A и B литров (объём – натуральное число). С этими сосудами можно выполнить одно из следующих действий:
   - заполнить сосуд A водой (даже если он уже был частично заполнен);
   - заполнить сосуд B водой (даже если он уже был частично заполнен);
   - перелить воду из сосуда A в сосуд B (при этом либо сосуд B доливается доверху, а в сосуде A остаётся какое-то количество воды, либо сосуд A полностью опустошается, а сосуд B может остаться заполненным лишь частично);
   - перелить воду из сосуда B в сосуд A (при этом либо сосуд A доливается доверху, а в сосуде B остаётся какое-то количество воды, либо сосуд B полностью опустошается, а сосуд A может остаться заполненным лишь частично);
   - опустошить сосуд A;
   - опустошить сосуд B.

   Частичное наполнение сосуда «на глазок» не разрешается. Найти такую последовательность действий, которая позволит в одном из сосудов получить объём воды X литров.

   Пример:

   $A = 3$, $B = 5$, $X = 4$.

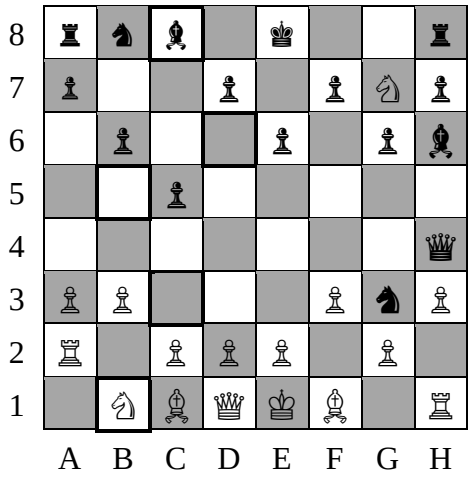
   Правильным ответом будет последовательность действий:
   - заполнить сосуд B,
   - перелить воду из сосуда B в сосуд A,
   - опустошить сосуд A,
   - перелить воду из сосуда B в сосуд A,
   - заполнить сосуд B,
   - перелить воду из сосуда B в сосуд A.
7. На шахматной доске 8×8 клеток произвольным образом расставлены шахматные фигуры. Необходимо для указанной фигуры (задаются координаты этой фигуры на доске: M – координата по вертикали, N – координата по горизонтали) определить минимальную последовательность ходов, которую фигура может сделать, чтобы попасть в клетку с заданными координатами P и Q (или указать, что это невозможно). При движении указанной фигуры остальные фигуры на доске не двигаются (ни на каком из ходов), однако фигура не может останавливаться на клетках, занятых фигурами её цвета, и не может останавливаться на клетках, находящихся под боем фигур противника, но при этом может останавливаться на клетках, занятых фигурами противника (бить их, после чего они уже не могут угрожать другим фигурам).

   Пример:

   M = 'B', $N = 1$,

   P = 'C', $Q = 8$

   Правильная последовательность показана на рисунке:



8. В магазин пришёл покупатель и выбрал товары на сумму S денежных единиц. У покупателя имеются монеты достоинством a_1, a_2, …, a_n. У продавца имеются монеты достоинством b_1, b_2, …, b_m. Необходимо выяснить, сможет ли (и каким набором монет) расплатится покупатель, и сможет ли (и каким набором монет) дать сдачу продавец. Покупатель и продавец действуют сообща. То есть, если у покупателя есть несколько вариантов расплатиться за товары с избытком (т. е. потребуется сдача), необходимо выбрать такой вариант, при котором продавец сможет дать сдачу (если такой вариант будет возможен). Программа должна вывести возможный вариант расплатиться без сдачи (с минимальным количеством монет); если это невозможно, программа должна вывести возможный вариант совершить покупку со сдачей (с минимальным общим количеством монет); если и это невозможно, программа должна вывести сообщение «покупка невозможна».

Пример:

$S = 13$

$n = 3$

a_1	a_2	a_3
5	10	20

$m = 6$

b_1	b_2	b_3	b_4	b_5	b_6
3	3	5	6	10	20

Возможно расплатиться за товар со сдачей:

Покупатель расплачивается монетами:

a_1	a_3	**сумма**
5	20	25

Продавец даёт сдачу монетами:

b_1	b_2	b_4	**сумма**
3	3	6	12

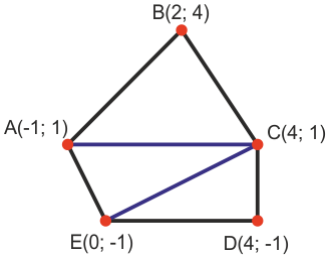
Всего 5 монет

9. Дан выпуклый n-угольник ($n > 3$), который задаётся координатами своих вершин на плоскости в порядке обхода по контуру. Необходимо разбить этот n-угольник на треугольники с помощью m ($m = n - 3$) непересекающихся диагоналей (кроме как в вершинах n-угольника). При этом сумма длин диагоналей должна быть минимальной.

Пример:

$n = 5$

На рисунке приведены координаты вершин и диагонали, сумма длин которых – минимальна:



10. В магазине каждый товар имеет отпускную цену. Чтобы привлечь покупателей, магазин ввёл скидки. Скидка заключается в том, чтобы продавать набор одинаковых или разных товаров по сниженной цене (меньше суммарной отпускной цены всех товаров набора). Необходимо вычислить наименьшую цену, которую покупатель должен заплатить за нужные ему покупки и определить, какими скидками (какими наборами) это можно сделать. Список товаров, которые требуется купить, нельзя дополнять ненужными («лишними») товарами, даже если это позволит снизить общую стоимость.
Пример:
цена 1 цветка равна 2€
цена 1 вазы равна 5€
цена набора из 3 цветков равна 5€ (вместо 6€)
цена набора из 2 ваз и 1 цветка равна 10€ (вместо 12€)
Необходимо купить 3 цветка и 2 вазы
Наименьшей ценой будет 14€ (набор из 2 ваз и 1 цветка за 10€ и 2 цветка по отдельности за 4€).

11. Имеется деревянная планка, параллельная оси Ox, в которую вбито n ($n \leqslant 100$) гвоздей. Известны координаты этих гвоздей x_i ($i = 1, \ldots, n$), при чём $x_i < x_i + 1, \forall\ i = 1, \ldots, n - 1$. Необходимо связать гвозди верёвочками так, чтобы:
- каждая верёвочка связывала ровно два гвоздя,
- к каждому гвоздю была привязана хотя бы одна верёвочка,
- суммарная длина верёвочек была бы минимальна.

Пример:
$n = 5$

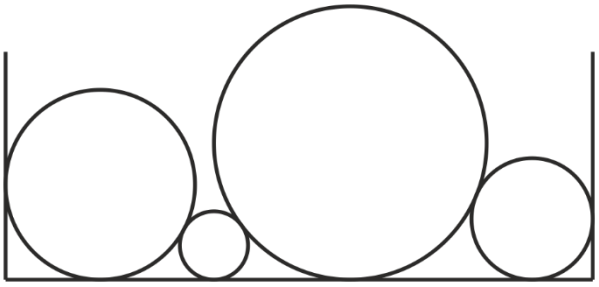
x_1	x_2	x_3	x_4	x_5
0	1	3	7	10

Связать веревочками понадобится следующие пары гвоздей:
(1, 2), (2, 3), (4, 5).
Суммарная длина верёвочек будет равна 6.

12. Имеется n дисков одинаковой толщины с радиусами $R_1, R_2, \ldots, R_n$. Эти диски упаковываются в коробку таким образом, что каждый из них стоит ребром на дне коробки, и все диски находятся в одной плоскости (то есть не могут перекрываться). Необходимо для заданной последовательности радиусов дисков определить минимальную длину коробки, в которую могут быть упакованы диски. Вывести длину коробки и последовательность размещения дисков.
Пример:
$n = 4$

R_1	R_2	R_3	R_4
9	16	25	36

Минимальная длина будет равна 155 при следующем порядке расположения: R_3, R_1, R_4, R_2



ЛАБОРАТОРНАЯ РАБОТА № 3
АБСТРАКТНЫЕ ТИПЫ И ОБРАБОТКА СТРОК

3.1. Ход лабораторной работы

Ознакомьтесь с примером сортировки массива слов. Слова передаются приложению через аргументы командной строки. Сортировка выполняется в алфавитном порядке, начиная с последних букв (словарь рифм).

Файл `StringComparator.java`:

```
import java.util.Comparator;

public class StringCompatator implements Comparator<String> {
    public int compare(String s1, String s2) {
        s1 = new StringBuilder(s1).reverse().toString();
        s2 = new StringBuilder(s2).reverse().toString();
        return s1.compareToIgnoreCase(s2);
    }
}
```

Файл `StringSorter.java`:

```
import java.util.Arrays;

public class StringSorter {
    public static void main(String[] args) {
        Arrays.sort(args, new StringCompatator());
        for(int i = 0; i < args.length; i++) {
            System.out.println(args[i]);
        }
    }
}
```

3.2. Самостоятельная работа «Сортировка массива строк»

Напишите программу, которая считывает из аргументов командной строки массив слов и сортирует этот массив по двум критериям (соответствующим варианту). Программа должна выводить в столбец сначала исходный массив слов, а затем этот же массив после каждой сортировки. После каждой сортировки программа должна выводить количество операций сравнения массива.

Указание: провести модульное тестирование с помощью JUnit-тестов.

Варианты задания:

1. По количеству вхождений заданной подстроки S в строку. По позиции первого вхождения заданной подстроки S в строку.
2. По позиции последнего вхождения заданной подстроки S в строку. По длине строки.
3. По части строки, расположенной между первым вхождением заданной подстроки S_1 и последним вхождением заданной подстроки S_2. По количеству маленьких букв.
4. По позиции первого вхождения заданной подстроки S в строку. По длине строки.
5. По количеству вхождений заданной подстроки S в строку. По количеству маленьких букв.
6. По длине строки. По части строки, расположенной между первым вхождением заданной подстроки S_1 и последним вхождением заданной подстроки S_2.
7. По количеству вхождений заданной подстроки S в строку. По позиции последнего вхождения заданной подстроки S в строку.
8. По позиции первого вхождения заданной подстроки S в строку. По количеству маленьких букв.
9. По количеству вхождений заданной подстроки S в строку. По длине строки.
10. По позиции последнего вхождения заданной подстроки S в строку. По части строки, расположенной между первым вхождением заданной подстроки S_1 и последним вхождением заданной подстроки S_2.
11. По длине строки. По количеству маленьких букв.
12. По позиции первого вхождения заданной подстроки S в строку. По части строки, расположенной между первым вхождением заданной подстроки S_1 и последним вхождением заданной подстроки S_2.
13. По позиции последнего вхождения заданной подстроки S в строку. По количеству маленьких букв.
14. По количеству вхождений заданной подстроки S в строку. По части строки, расположенной между первым вхождением заданной подстроки S_1 и последним вхождением заданной подстроки S_2.

3.3. Самостоятельная работа «Обработка строк»

Напишите программу, которая преобразует входную строку в соответствии с вариантом. Обработку строки осуществите с использованием класса `StringBuilder`.

Указание: провести модульное тестирование с помощью JUnit-тестов.

Варианты задания:

1. Удалить из строки подстроки, заключённые между последовательностями символов `/*` и `*/` (включая сами эти символы).
2. Заменить в строке все подстроки, заключённые между круглыми скобками (вместе с самими круглыми скобками), на число – порядковый номер этих круглых скобок (при этом число заключается в квадратные скобки).
3. Вставить между пустыми фигурными скобками в строке число – порядковый номер с конца данных фигурных скобок.

4. Добавить в конец строки все подстроки этой же строки, заключённые в двойные кавычки. При добавлении в конец строки двойные кавычки опускать. Перед каждой добавленной строкой добавлять символ перехода на новую строку и порядковый номер добавляемой строки.
5. Удалить из строки все подстроки, начинающиеся со знака "меньше", затем некоторого натурального числа N, затем знака "больше", и ещё N символов после знака больше.
6. Заменить все троеточия в строке на последовательные большие буквы русского алфавита. Если количество троеточий превышает количество букв алфавита, то после подстановки буквы Я вновь подставлять букву А.
7. Вставить после каждого восклицательного знака в строке его порядковый номер (в исходной строке).
8. Добавить в конец строки для каждой гласной буквы русского алфавита, встречающейся в этой же строке, количество вхождений этой буквы в строку. Перед каждым числом добавлять точку и пробел.
9. Удалить из строки все числа, кроме однозначных.
10. Заменить все однозначные числа в строке их словесным описанием (в именительном падеже).
11. Все IP-адреса в сроке (подстроку, состоящую из четырёх целых неотрицательных чисел, разделённых тремя точками) заключить в скобки вида: ${192.168.0.1}.
12. Добавить в конец строки все встречающиеся в строке номера телефонов в формате XX-XX-XX (где X – некоторая цифра). При добавлении в конец строки разделять номера телефонов запятыми.
13. Заменить в строке все подстроки, заключённые в фигурные скобки, их порядковым номеров в квадратных скобках, а удалённое содержимое добавить в конец строки, добавив перед каждой такой подстрокой символ перехода на новую строку, порядковый номер этой подстроки в строке, точку и пробел.
14. Удалить в тексте все двойные символы подчёркивания, а между буквами слова, перед которым стояло такое двойное подчёркивание, вставить по одному пробелу.

3.4. Самостоятельная работа «Использование интерфейсов»

С помощью предоставленной преподавателем библиотеки `strings.jar`, в которой описан интерфейс `Job`

```
package by.vsu.mf.ai.ssd.strings;

public interface Job {
    void perform(StringBuilder str);
}
```

и класс `Manager`

```
package by.vsu.mf.ai.ssd.strings;

public class Manager {
    public static void createWindow(Job job) {
        /* реализация */
    }
}
```

выполните самостоятельную работу 3.3, описав свою реализацию интерфейса `Job`.

Класс, реализующий интерфейс `Job`, в методе `perform` получает ссылку на экземпляр класса `StringBuilder` и выполняет с этой строкой некоторые действия, изменяя эту строку. Метод `createWindow` класса `Manager` создаёт окно, в котором присутствует поле для ввода пользователем текста, и кнопка «Обработать», по нажатию на которую текст из поля для ввода преобразуется в экземпляр класса `StringBuilder`, передаётся на управление классу, реализующему интерфейс `Job`, и после обработки изменённый текст опять заносится в поле для ввода.

Указание: для подключения библиотеки в проект в IDE сохраните jar-файл в каталоге проекта и добавьте файл в classpath в свойствах проекта (конкретные настройки могут отличаться в различных IDE).

3.5. Самостоятельная работа «Проектирование абстракций»

Выполните самостоятельную работу «Сортировка массива строк», разработав собственную реализацию двух методов сортировки (см. вариант). Классы, реализующие методы сортировки, должны.

- быть объединены в иерархию наследования с интерфейсом в вершине иерархии;
- позволять сортировать массив элементов различных типов (а не только строк), используя обобщения (Generics);
- использовать стандартный интерфейс `Comparator`.

После сортировки выводить количество операций сравнения и количество операций обмена.

Указание: провести модульное тестирование с помощью JUnit-тестов.

Варианты задания:

1. Сортировка выбором (Selection Sort). Гномья сортировка (Gnome Sort).
2. Сортировка обменом (Bubble Sort). Сортировка слияниями (Merge Sort).
3. Сортировка выбором (Selection Sort). Быстрая сортировка (Quicksort).
4. Сортировка вставками (Insertion Sort, с поиском места вставки с помощью стандартной реализации бинарного поиска). Сортировка слияниями (Merge Sort).
5. Сортировка выбором (Selection Sort). Сортировка Шелла (Shell Sort).
6. Сортировка обменом (Bubble Sort). Пирамидальная сортировка (Heapsort).
7. Сортировка выбором (Selection Sort). Сортировка слияниями (Merge Sort).
8. Сортировка обменом (Bubble Sort). Быстрая сортировка (Quicksort).
9. Сортировка вставками (Insertion Sort, с поиском места вставки с помощью стандартной реализации бинарного поиска). Шейкерная сортировка (Cocktail Sort).
10. Сортировка обменом (Bubble Sort). Гномья сортировка (Gnome Sort).
11. Сортировка выбором (Selection Sort). Пирамидальная сортировка (Heapsort).
12. Сортировка обменом (Bubble Sort). Сортировка Шелла (Shell Sort).
13. Сортировка вставками (Insertion Sort, с поиском места вставки с помощью стандартной реализации бинарного поиска). Быстрая сортировка (Quicksort).
14. Сортировка выбором (Selection Sort). Шейкерная сортировка (Cocktail Sort).

ЛАБОРАТОРНАЯ РАБОТА № 4
ВВОД-ВЫВОД. КОЛЛЕКЦИИ

4.1. Ход лабораторной работы

1. Ознакомьтесь с примером вычисления значения выражения, в котором значения входных величин вводятся с клавиатуры:

```
import java.util.Scanner;

public class Runner {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Введите скорость автомобиля (км/ч): ");
    double speed = scanner.nextDouble();
    System.out.print("Введите расстояние (км): ");
    double distance = scanner.nextDouble();
    double time = distance / speed;
    int hours = (int) time;
    int minutes = (int) ((time - hours) * 60);
    System.out.println("Затраченное время: " + hours + " ч. "
                                              + minutes + " мин.");
  }
}
```

2. Ознакомьтесь с примером построчного чтения текстового файла:

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.Reader;

public class Runner {
  public static void main(String[] args) {
    try {
      Reader reader = new FileReader("file.txt");
      BufferedReader buffReader = new BufferedReader(reader);
      String line;
      while((line = buffReader.readLine()) != null) {
        System.out.println(line);
      }
      buffReader.close();
    } catch(FileNotFoundException e) {
      System.out.println("Файл не найден");
    } catch(IOException e) {
      System.out.println("Ошибка ввода-вывода");
    }
  }
}
```

3. Ознакомьтесь с примером сериализации и десериализации объектов:

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.OutputStream;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

class Entity implements Serializable {
  private Integer someData;
  public Integer getSomeData() {
    return someData;
  }
  public void setSomeData(Integer someData) {
    this.someData = someData;
  }
  @Override
  public String toString() {
    return someData.toString();
  }
}
public class Runner {
  @SuppressWarnings("unchecked")
  public static void main(String[] args) {
    List<Entity> entities;
    try {
      InputStream is = new FileInputStream("file.bin");
      ObjectInputStream ois = new ObjectInputStream(is);
      entities = (List<Entity>)ois.readObject();
      ois.close();
    } catch(IOException | ClassNotFoundException e) {
      entities = new ArrayList<>();
    }
    System.out.println(entities);
    Entity entity = new Entity();
    entity.setSomeData(new Random().nextInt(1000));
    entities.add(entity);
    System.out.println(entities);
    try {
      OutputStream os = new FileOutputStream("file.bin");
      ObjectOutputStream oos = new ObjectOutputStream(os);
      oos.writeObject(entities);
      oos.close();
    } catch(IOException e) {
      System.out.println("Невозможно сохранить файл");
    }
  }
}
```

4.2. Самостоятельная работа «Стеки и очереди»

Напишите программу, решающую предложенную задачу (см. вариант) с применением стека и очереди. Стек и очередь должны быть реализованы самостоятельно (способ реализации указан в варианте). Программа должна считывать входные данные с клавиатуры, формировать необходимые структуру данных и выполнять решение поставленной задачи.

Указание: создать JUnit тесты для методов разработанных структур данных, а также для метода, решающего поставленную задачу.

Варианты задания.

№ варианта	Способ реализации стека	Способ реализации очереди	№ задачи (список см. ниже)
1	связанный список	массив	1
2	массив	связанный список	2
3	связанный список	массив	3
4	массив	связанный список	4
5	связанный список	массив	5
6	массив	связанный список	6
7	связанный список	массив	7
8	массив	связанный список	1
9	связанный список	массив	2
10	массив	связанный список	3
11	связанный список	массив	4
12	массив	связанный список	5
13	связанный список	массив	6
14	массив	связанный список	7

Список задач.

1. Смоделировать процесс мыться тарелок в столовой. На вход в программу поступает информация о тарелках: цвет тарелки (строковое значение), время на помывку тарелки (целое число в секундах), время поступления тарелки на обработку (целое число в секундах — время относительно предыдущей тарелки). Все тарелки поступают в конец очереди. Имеется n мойщиков тарелок (число вводится с клавиатуры). Далее каждая секунда обрабатывается отдельно. При обработки очередной секунды выполняются следующие действия:
   - У тарелки, находящейся в начале очереди, на 1 уменьшается время поступления тарелки на обработку.
   - Если у тарелки, находящейся в начале очереди, время поступления тарелки на обработку равно 0, то она помещается в стек (стопку) грязных тарелок.
   - Для каждого мойщика проверяется, если у него тарелка, которую он сейчас моет. Если её нет, он забирает себе верхнюю тарелку из стопки грязных тарелок (если там есть тарелки).
   - Для каждого мойщика проверяется, если у тарелки, которую он сейчас моет, время на помыку равно 0, то он перекладывает её в стек (стопку) чистых тарелок, иначе время помывки этой тарелки уменьшается на 1.

- Процесс продолжается до тех пор, пока не будут вымыты все тарелки из входной очереди.

В конце:

- Вывести порядок цветов тарелок, лежащих в стопке чистых тарелок (начиная с верхней).
- Для каждого мойщика тарелок вывести количество тарелок, которые он помыл, общее время мытья тарелок, суммарное время «простоя» мойщика.

2. Дан список строк. Для каждой строки сформировать очередь из символов этой строки, которые являются скобками. Рассмотреть скобки вида: (), [], {}, <>. С помощью стека проверьте баланс скобок в этой строке. При проверке баланса учитывать правильную последовательность открытия и закрытия скобок. Примеры неправильного баланса скобок: `"]["`, `"({)}"`.
3. Смоделировать процесс ремонта одежды двумя швеями. На вход в программу подаётся информация об одежде, нуждающейся в ремонте: количество пуговиц, которые необходимо пришить; время на штопку (целое число в секундах); время поступления на обработку (целое число в секундах – время относительно предыдущего предмета одежды). Вся одежда поступает в конец очереди. Одежда ремонтируется на конвейере, на котором работает две швеи. Первая берёт одежду из входной стопки, пришивает пуговицы (на каждую пуговицу она тратит *t* секунд), и кладёт её в промежуточную стопку. Вторая берёт одежду из промежуточной стопки, штопает одежду, и кладёт её в выходную стопку. Далее каждая секунда в приложении обрабатывается отдельно. При обработке очередной секунды выполняются следующие действия:
   - У предмета одежды, находящегося в начале очереди, на 1 уменьшается время поступления на обработку.
   - Если у предмета одежды, находящегося в начале очереди, время поступления на обработку равно 0, то он помещается во входную стопку (стек).
   - Для каждой швеи проверяется, есть ли у неё одежда в работе. Если нет, она забирает себе в работу очередной предмет одежды. Первая – из входной стопки (стека), вторая – из промежуточной стопки (стека). При этом если стопка пустая, то соответствующая швея ничего в это время не делает.
   - Для каждой швеи проверяется, если одежда, находящаяся в работе, уже обработана (время пришивания пуговиц или время штопки истекло), то предмет одежды откладывает его. Первая швея – в промежуточную стопку (стек). Вторая – в выходную стопку (стек). Если же одежда ещё не обработана, то время пришивки пуговиц (изначально – количество пуговиц, умноженное на *t*) или штопки уменьшается на 1.
   - Процесс продолжается до тех пор, пока не будут отремонтированы все предметы одежды.

   В конце для каждой швеи вывести суммарное время работы и суммарное время простоя.
4. Смоделировать процесс карточной игры. В начале игры формируется стопка (колода) из 52 карт (карты случайным образом перемешиваются при формировании колоды). В игру играет *n* игроков, из которых формируется очередь. Перед началом игры в отдельную стопку на столе (снос) кладётся одна карта с верха колоды. Далее на очередном ходу игрок, стоящий в начале очереди, берёт из колоды карты по одной. Каждую очередную взятую карту он кладёт в снос, если её достоинство отчается ровно на 1 от достоинства карты в верху сноса или её масть совпадает с мастью карты в верху сноса. Процесс продолжается до тех пор, пока игрок не возьмёт в

руку карту, которую он не может положить в снос. После этого он может по тем же правилам пытаться переложить в снос карты с верха своей стопки (которая в начальный момент игры пустая). Когда процесс перекладывания карт со своей стопки в снос закончился (очередная верхняя карта со своей стопки не может по правилам быть переложена в снос или своя стопка уже опустела), то карта, находящаяся в руке, кладётся на верх своей стопки. После чего текущий игрок помещается в конец очереди и ход переходит к следующему игроку. Игра продолжается до тех пор, пока не опустеет колода. В конце необходимо вывести содержимое стопок всех игроков (начиная с верхней карты) в порядке убывания количества карт в стопках. Для сортировки очереди игроков реализовать сортировку очереди методом обмена («пузырька»), используя только операции извлечения из начала и добавления в конец и используя дополнительную очередь.

5. Смоделировать процесс технического обслуживания и ремонта газовых баллонов. На вход в программу поступает информация о газовых баллонах: инвентарный номер баллона, объём газа в баллоне (в процентах от общей ёмкости, при этом считается, что общая ёмкость всех баллонов одинакова), возможность дальнейшей эксплуатации баллона (булевское значение). Все баллоны поступают в конец очереди. Обработка этих баллонов осуществляется следующим образом:
   - Из начала очереди извлекается баллон.
   - Если он может эксплуатироваться в дальнейшем, и объём газа в нём равен 100%, то он помещается в стек отремонтированных баллонов.
   - Если он может эксплуатироваться в дальнейшем, но объём газа в нём меньше 100%, то он помещается в стек баллонов, нуждающихся в заправке.
   - Если очередной баллон больше не может эксплуатироваться в дальнейшем, и он пустой (объём газа равен 0%), то он помещается в стек списанных баллонов.
   - Если очередной баллон больше не может эксплуатироваться в дальнейшем (назовём его списываемым), но он не пустой, то из стека с баллонами, нуждающихся в заправке, извлекается по одному баллоны, и дозаправляются до 100% газом из текущего списываемого баллона, на них наносится информация о том, на какой объём и из какого баллона они были дозаправлены, после чего дозаправленный баллон помещается в стек отремонтированных баллонов. Процесс продолжается до тех пор, пока либо не закончится газ в списываемом баллоне, тогда он помещается в стек списанных баллонов, а до-заправляемый баллон, если он не был дозаправлен до 100%, возвращается в стек баллонов, нуждающихся в заправке; либо пока стек баллонов, нуждающихся в заправке, не опустеет, тогда списываемый баллон помещается в ещё один стек с баллонами-донорами.
   - Процесс продолжается, пока входная очередь не опустеет.
   - Если (после опустошения входной очереди баллонов) стек баллонов-доноров и стек баллонов, нуждающихся в заправке, не пустые, тогда баллоны-доноры извлекаются по одному из стека и возвращаются в конец входной очереди, и процесс снова повторяется, пока после опустошения очереди либо не останется баллонов-доноров, либо не останется пригодных, но не дозаправленных баллонов.
   - В конце вывести статистику по всем дозаправленным баллонам (сколько раз, на какой объём и из каких баллонов они дозаправлялись).
   - Отдельно вывести статистику по всем не до конца заправленным баллонам (если таковые останутся).
   - Отдельно вывести информацию по списываемым баллонам, в которых остался газ (с указанием оставшегося объёма).
   - Отдельно вывести информацию о списанных баллонах (только их инвентарные номера).

6. Дан список строк. Каждая строка представляет собой арифметическое выражение в польской инверсной записи. В таком выражении сначала записываются два операнда, а затем знак операции. Операнды и операции разделяются пробелами. Для каждой строки сформировать очередь из операндов и операций (очередь, содержащую одно выражение в польской инверсной записи). Например, выражение $\frac{(2+3)\cdot(4+6)}{7-5}$ в польской инверсной записи будет выглядеть так:

```
2 3 + 4 6 + * 5 2 - /
```

С помощью стека вычислите каждое из выражений. Для организации вычислений каждый числовой операнд, считываемый из очереди-выражения, помещается в вершину стека. При считывании из очереди-выражения операции, из стека извлекаются операнды, вычисляется значение операции и результат помещается в вершину стека.

7. Смоделировать процесс обработки пакетов документов в канцелярии. В канцелярии в поступающих пакетах документов клерками делаются записи различных видов. Количество (n) и описание видов записей вводятся с клавиатуры. Каждый вид записи характеризуется уникальным номером (целое неотрицательное число), названием (строка), длительностью внесения записи в документ (целое число минут). В канцелярии работает m клерков. Для каждого клерка задано: его уникальный номер (целое неотрицательно число), имя (строка) и список номеров видов записей, которые он может осуществлять. Каждый клерк может осуществлять не менее одного вида записей. Каждый вид записей может осуществляться как минимум одним клерком. Ни один клерк не знает, какие виды записи могут делать другие клерки в канцелярии. Клерки сидят в канцелярии по кругу. У каждого клерка есть сосед, которому он может передать пакет документов для дальнейшей обработки, таким образом все клерки образуют связанную замкнутую цепочку, в которой уникальные номера клерков возрастают (за исключением первого и последнего клерка в цепочке, которые замыкают цепочку). На вход в программу поступает информация о пакетах с документами. Для каждого пакета задаётся: номер (целое неотрицательное число); описание (строка); список номеров видов записей, которые необходимо сделать в этих документах, чтобы пакет документов считался обработанным; время поступления пакета на обработку (целое число минут). Далее каждая минута в приложении обрабатывается отдельно. При обработке очередной минуты выполняются следующие действия:
   - У пакета документов, находящегося в начале очереди, на 1 уменьшается время поступления на обработку.
   - Если у пакета документов, находящегося в начале очереди, время поступления на обработку равно 0, то он помещается на верх стопки документов на столе некоторого клерка. У каждого клерка на столе своя стопка документов, которые он должен обработать. При выборе клерка, которому необходимо передать пакет документов из очереди, необходимо выбирать клерка с наименьшей стопкой документов. Если таких клерков несколько, то выбирается клерк с минимальным уникальным номером.
   - Для каждого клерка проверяется, есть ли у него документ в работе. Если нет, он забирает себе в работу очередной пакет документов из своей стопки. При этом если стопка пустая, то клерк ничего в это время не делает.
   - Для каждого клерка проверяется, если документы, находящиеся в работе, уже обработаны (в пакет документов уже внесены все необходимые виды записей), то пакет перемещается в общую для всей канцелярии стопку готовых документов.

- Для каждого клерка проверяется, если в документах, находящихся в работе, есть ещё не внесённые записи, но все такие записи относятся к видам записей, которые не может вносить в документ данный клерк, то пакет перемещается в стопку документов на столе соседнего клерка.
- Для каждого клерка проверяется, если в документах, находящихся в работе, есть ещё не внесённые записи, которые может внести данный клерк, и время внесения данной записи уже равно 0, то в документе фиксируется, что данная запись внесена данным клерком (фиксируется уникальный номер клерка).
- Для каждого клерка проверяется, если в документах, находящихся в работе, есть ещё не внесённые записи, которые может внести данных клерк, и время внесения этих записей ещё не истекло, то время внесения изменения одной из этих записей уменьшается на 1.
- Процесс продолжается до тех пор, пока не будут обработаны все документы из очереди (не будут перемещены в стопку готовых документов).

В конце необходимо:

- Вывести состояние стопки готовых документов. Для каждого документа из стопки (в порядке начиная с верхнего пакета) выводится номер и описание документов, все сделанные в документах записи (название вида записи и имя клерка, сделавшего запись) и общее время обработки пакета документов (от времени поступления первому клерку в стопку до времени поступления в стопку готовых документов. Также вывести среднее время обработки документов по всем обработанным в канцелярии пакетам документов.
- Вывести для каждого клерка: общее время работы клерка; общее время «простоя» клерка; количество документов, обработанных клерком; максимальное количество документов в стопке на столе за всё время обработки всех пакетов в канцелярии.

4.3. Самостоятельная работа «Обработка CSV»

Напишите программу, которая из файла формата CSV считывает данные, соответствующие индивидуальному варианту. Для хранения данных в памяти использовать возможности коллекций. Если файл отсутствует, то приложение должно запускаться без ошибок и работать с пустой коллекцией. После запуска пользователь через командную строку должен иметь возможность просматривать данные, добавлять новые, редактировать или удалять существующие, а также выполнять специфическую для варианта обработку данных. После каждой операции, изменяющей данные, все эти данные из коллекции должны сохраняться в файл (при отсутствии файла он должен создаваться). Формат CSV – это текстовый формат для хранения таблиц, ячейки которых являются строками. В файле такого формата на каждую строку таблицы отводится одна строка файла. Ячейки в строке разделяются символами «;». Таблицы в таком формате можно сохранять с помощью программы Microsoft Excel. При обработке такого файла в Java-программе можно считывать файл построчно, а затем каждую прочитанную

строку разбивать на массив строк с помощью метода `split` класса `String`, передавая методу в качестве шаблона-разделителя строку `";"`.

Указание: создать JUnit тесты для методов классов, выполняющих логику обработки данных.

Варианты задания:

1. Информация о телефонных разговорах: номер телефона вызывающего абонента; номер телефона вызываемого абонента; дата и время вызова; продолжительность звонка. Необходимо получить распечатку (в хронологическом порядке) всех звонков (и входящих, и исходящих) выбранного абонента за указанный промежуток времени с суммированием общей продолжительности входящих и исходящих вызовов.
2. Информация о сотрудниках предприятия: фамилия и инициалы; пол; дата рождения; дата приёма на работу; должность. Необходимо на указанную дату рассчитать стаж работы на данном предприятии каждого сотрудника и сформировать список пенсионеров по возрасту на эту дату (считать, что пенсионный возраст женщин — 55 лет; пенсионный возраст мужчин — 60 лет).
3. Информация о пересылаемых почтовым сервером электронных письмах: адрес отправителя; адрес получателя; дата и время отправки письма; объём письма в байтах. Необходимо получить распечатку почтового трафика для выбранного адреса за выбранный промежуток времени с суммированием общего объёма корреспонденции, в том числе и отдельно по входящей и исходящей почте.
4. Информация о пациентах участкового терапевта: фамилия и инициалы пациента; дата рождения; пол; масса в килограммах; рост метрах. Необходимо вывести список пациентов, имеющих излишнюю массу, и список пациентов, имеющих недостаточную массу. Каждый список отсортировать по возрасту пациентов. Для определения, избыточна масса, или недостаточна, использовать индекс массы тела, равный отношению массы к квадрату роста. Если данный индекс меньше 19, масса недостаточна. Если данный индекс больше 24 (для женщин) или 25 (для мужчин), масса избыточна.
5. Информация о результатах тестирования: фамилия и инициалы тестируемого; дата проведения теста; название дисциплины; количество заданных вопросов; количество правильных ответов. Необходимо сформировать протокол тестирования на заданную дату по заданной дисциплине, указав результаты каждого тестируемого в процентах (с округлением до 1 знака после запятой) и сортировкой по фамилии тестируемого или по результатам.
6. Информация о результатах сессии: фамилия и инициалы студента; курс; дисциплина; оценка на экзамене. Необходимо сформировать списки, сгруппированные по курсам: отчисляемых студентов (неудовлетворительно сдавших 3 и более экзаменов); студентов, имеющих академические задолженности (хотя бы одна неудовлетворительная оценка); студентов, успешно сдавших сессию с выводом среднего балла и отсортированного по этому среднему баллу (в пределах каждого курса).
7. Информация о киносеансах в кинотеатрах города: название кинотеатра; название фильма; жанр фильма; дата и время показа. Необходимо сформировать расписание киносеансов для выбранного кинотеатра на выбранную дату. А также расписание киносеансов по всем кинотеатрам фильмов выбранного жанра. Расписания должны быть упорядочены хронологически.
8. Информация об использовании компьютеров в пункте коллективного доступа: номер компьютера; фамилия и инициалы клиента; дата; время начала работы; время

окончания работы. Необходимо получить отчёт о времени использования каждого компьютера на указанную дату, а также на выбранную дату информацию об использовании выбранного компьютера каждым пользователем с указанием времени пользования для каждого клиента (в хронологическом порядке).

9. Информация о пользовании дисками в пункте проката: фамилия и инициалы клиента; название диска; дата выдачи диска; дата предполагаемого возврата диска; дата фактического возврата. Вывести список клиентов, просрочивших возврат дисков (в случае просрочки возврата нескольких дисков фамилию клиента выводить только один раз).
10. Информация о выдаче книг в библиотеке: фамилия и инициалы читателя; название книги; автор книги; дата выдачи книги; дата возврата книги. Необходимо для выбранного читателя на выбранную дату вывести список находившихся у него книг. А также для выбранной книги выбранного автора вывести всех читателей, бравших эту книгу.
11. Информация о расписании движения междугородних автобусов: день недели; пункт отправления; пункт назначения; время отправления; время прибытия. Необходимо сформировать расписание на выбранный день недели для выбранного пункта с выводом продолжительности нахождения каждого автобуса в пути (выводить рейсы в хронологическом порядке по времени отправления, если автобус отправляется из выбранного пункта, или времени прибытия, если автобус прибывает в выбранный пункт).
12. Информация об абитуриентах, поступающих на некоторую специальность: фамилия и инициалы; количество баллов по каждому из трёх сертификатов централизованного тестирования; средний балл аттестата. Необходимо для введённого количества мест рассчитать проходной балл и сформировать список зачисленных абитуриентов, упорядоченный по суммарному баллу. При этом следует помнить, что если есть абитуриенты, у которых суммарный балл одинаковый, но они не могут быть зачислены все, так как количество оставшихся мест меньше количества абитуриентов, то предпочтение отдаётся тому из них, у кого выше балл по первому предмету централизованного тестирования (далее по приоритету следует второй предмет централизованного тестирования, далее средний балл аттестата). Если есть абитуриенты, у которых все оценки полностью одинаковы, и они не могут быть все зачислены, такие студенты выводятся отдельно с указанием количества оставшихся мест.
13. Информация о расписании занятий на факультете: день недели; номер пары; группа; дисциплина; фамилия и инициалы преподавателя. Необходимо сформировать расписание на выбранный день для выбранного преподавателя. Также сформировать расписание на выбранный день для выбранной группы.
14. Информация о талонах на приём в поликлинике: фамилия и инициалы врача; специальность врача; фамилия и инициалы пациента; дата и время начала приёма, длительность приёма. Необходимо на выбранную дату вывести расписание приёма выбранного врача. Также на выбранную дату найти свободное время приёма специалистов выбранной специальности (считать, что все специалисты ведут приём с 8:00 до 16:00).

4.4. Самостоятельная работа «Сериализация»

Напишите программу, которая обрабатывает данные согласно варианта. Данные при запуске приложения считываются из файла с применением десериализации. Данные могут добавляться/удаляться пользователем с клавиатуры. При добавлении/удалении данных пользователем, программа должна их записывать в файл с помощью сериализации. Хранение и обработку данных осуществлять с применением стандартных коллекций (множества или карты отображений, в зависимости от варианта). При отсутствии файла при запуске приложения должна использоваться пустая коллекция.

Указание: создать JUnit тесты для методов классов, выполняющих логику обработки данных.

Варианты задания:

1. Дан список книг, для каждой из которых заданы: фамилия и инициалы автора, название, жанр, год издания. С помощью карт отображений выведите: для каждого автора список его книг, отсортированный по году издания; для каждого жанра список книг, отсортированный по фамилии автора.
2. Дан набор множеств целых чисел. Среди всех этих множеств найти такую пару множеств, объединение которых будет содержать максимальное количество элементов.
3. Дан текст. Все слова текста разделяются пробелами (табуляция, знаки препинания и другие разделители не используются). Составить с использованием множества словарь данного текста (список всех встречающихся в тексте слов в алфавитном порядке).
4. Дан список дисциплин факультета, для каждой из которых заданы: название дисциплины, фамилия и инициалы преподавателя, количество часов по данной дисциплине, специальность. С помощью карт отображений выведите: для каждого преподавателя список его дисциплин с суммарным количеством часов по каждому преподавателю; для каждой специальности список дисциплин, отсортированный по названию.
5. Дан набор множеств целых чисел. Среди всех этих множеств найти такую пару множеств, объединение которых будет содержать минимальное количество элементов.
6. Дан текст. Все слова текста разделяются пробелами (табуляция, знаки препинания и другие разделители не используются). Составить с использованием карты отображений частотный словарь данного текста (список всех встречающихся в тексте слов в алфавитном порядке с указанием частоты встречи слова в тексте — процентное отношение количества таких слов к общему количеству слов в тексте).
7. Дано множество точек плоскости. Вывести уравнения всех таких различных прямых, которые проходят через всевозможные пары точек исходного множества. Уравнения этих прямых вывести в виде $A \cdot x + B \cdot y + C = 0$.
8. Дан список заказов в Интернет-магазине, для каждого заказа заданы: фамилия и инициалы покупателя, название товара, количество заказанных единиц товара, стоимость одной единицы товара. С помощью карт отображений выведите: для каждого покупателя список заказов с суммарной стоимостью всех его заказов; для каждого товара список заказов с суммарной стоимостью.

9. Дан набор множеств целых чисел. Среди всех этих множеств найти такую пару множеств, пересечение которых будет содержать максимальное количество элементов.
10. Дан список векторов трёхмерного пространства. С использованием множеств определить, можно ли из этих векторов выбрать базис в трёхмерном пространстве.
11. Дан список задач для IT-проекта, для каждой задачи заданы: исполнитель, описание задачи, время выполнения, статус (новое, в процессе выполнения, отменённое, выполненное, невыполненное). С помощью карт отображений выведите для каждого исполнителя список его задач, сгруппированный по статусам, для каждого статуса просуммировать время выполнения.
12. Дан набор множеств целых чисел. Среди всех этих множеств найти такую пару множеств, пересечение которых будет содержать минимальное количество элементов.
13. Дан список рейсов междугородних автобусов, для каждого рейса заданы: станция отправления, станция назначения, дата рейса, время отправления, время прибытия, количество мест в автобусе. С помощью карт отображений выведите на каждую дату по каждой станции список рейсов (рейсов, для которых данная станция является станцией отправления или прибытия), отсортированный по времени отправления (если текущая станция является станцией отправления) или времени прибытия (если текущая станция является станцией прибытия).
14. Дан список блюд из меню ресторана. Для каждого блюда заданы: название, категория (салат, первое, второе блюдо и т.д.), множество ингредиентов (название ингредиента, количество в блюде), стоимость. С помощью карт отображений сначала вывести меню для повара, сгруппированное по категории блюд, в рамках каждой категории для каждого ингредиента, используемого хотя бы в одном блюде из текущей категории, вывести список блюд, в которых присутствует данный ингредиент (при этом одно и то же блюдо может выводиться несколько раз для каждого ингредиента). Затем вывести тоже меню для посетителей, сгруппированное по категории, в рамках каждой категории отсортировать блюда по стоимости.

СПИСОК ЛИТЕРАТУРЫ

1. Васильев, А. Н. Java. Объектно-ориентированное программирование: учебное пособие для магистров и бакалавров, базовый курс / А. Н. Васильев. – Санкт-Петербург: Питер, 2013. – 397 с.
2. Блинов, И. Н. Java. Промышленное программирование / И. Н. Блинов. – Минск: Универсал-Пресс, 2007. – 704 с.
3. Маслов, В. В. Основы программирования на языке Java: учебный курс / В. В. Маслов. – Москва: Горячая линия-Телеком, 2000. – 132 с.
4. Джамса, К. Библиотека программиста Java / К. Джамса. – Минск: Попурри, 1996. – 640 с.
5. Морган, М. Java 2: руководство разработчика / М. Морган. – Москва: Вильямс, 2000. – 720 с.
6. Ноутон, П. Java 2 / П. Ноутон. – Санкт-Петербург: БХВ-Петербург, 2008. – 1050 с.
7. Эккель, Б. Философия Java / Б. Эккель. – 4-е издание. – Санкт-Петербург: Питер, 2009. – 640 с.

Учебное издание

ЕРМОЧЕНКО Сергей Александрович
КОРЧЕВСКАЯ Елена Алексеевна

ПЛАТФОРМЕННО НЕЗАВИСИМЫЙ ЯЗЫК ПРОГРАММИРОВАНИЯ JAVA

Методические рекомендации по выполнению лабораторных работ

Технический редактор *Г.В. Разбоева*
Компьютерный дизайн *В.Л. Пугач*

Подписано в печать 19.12.2022. Формат 60x84 $^{1}/_{16}$. Бумага офсетная.
Усл. печ. л. 2,96. Уч.-изд. л. 2,52. Тираж экз. Заказ .

Издатель и полиграфическое исполнение – учреждение образования
«Витебский государственный университет имени П.М. Машерова».

Свидетельство о государственной регистрации в качестве издателя,
изготовителя, распространителя печатных изданий
№ 1/255 от 31.03.2014.

Отпечатано на ризографе учреждения образования
«Витебский государственный университет имени П.М. Машерова».
210038, г. Витебск, Московский проспект, 33.