Day 02 - Piscine Java

IO, Files

Резюме: Сегодня вы научитесь работать с вводом/выводом в Java и реализуете программы для манипуляций с файловой системой

Contents

Chapter I
Preamble

Операции ввода-вывода играют большое значение в разработке корпоративных систем. Зачастую необходимо реализовывать функционал по загрузке и обработке пользовательских файлов, обеспечить возможность отправки различных документов по почте и т.д.

Естественно, ввод-вывод никогда не сводится только к работе с файловой системой. Любое клиент-серверное взаимодействие между приложениями сводится к операциям ввода-вывода. Так, например, технология Java Servlets, используемая в Web-разработке предоставляет возможность формирования HTML-страниц с помощью класса PrintWriter.

Важно помнить, что функциональность, связанная с вводом-выводом не ограничена стеком Java IO. Существует большое количество библиотек, значительно упрощающих взаимодействие с потоками данных, среди которых - Apache Commons IO.

Chapter II
General Rules

- Use this page as the only reference. Do not listen to any rumors and speculations about how to prepare your solution.
- Сейчас для вас существует только одна версия Java - 1.8. Убедитесь, что на вашем компьютере установлен компилятор и интерпретатор данной версии.
- Не запрещено использовать IDE для написания исходного кода и его отладки.
- Код чаще читается, чем пишется. Внимательно изучите представленный документ с правилами оформления кода. В каждом задании обязательно придерживайтесь общепринятых стандартов Oracle - https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html
- Комментарии в исходном коде вашего решения запрещены. Они мешают восприятию.
- Pay attention to the permissions of your files and directories.
- To be assessed your solution must be in your GIT repository.
- Your solutions will be evaluated by your piscine mates.
- You should not leave in your directory any other file than those explicitly specified by the exercise instructions. It is recommended that you modify your .gitignore to avoid accidents.
- When you need to get precise output in your programs, it is forbidden to display a precalculated output instead of performing the exercise correctly.
- Have a question? Ask your neighbor on the right. Otherwise, try with your neighbor on the left.
- Your reference manual: mates / Internet / Google. И еще, для любых ваших вопросов существует ответ на Stackoverflow. Научитесь правильно их задавать.
- Read the examples carefully. They may require things that are not otherwise specified in the subject.
- And may the Force be with you!
- Не откладывайте на завтра то, что можно было сделать вчера ;)

Chapter III
Exercise 00 - File Signatures

Exercise 00: File Signatures	
Turn-in directory	ex00
Files to turn-in	*.java, signatures.txt
Разрешения	
Типы	Java Collections API (List<T>, Map<K, V> , etc.) InputStream, OutputStream, FileInputStream, FileOutputStream

Классы ввода-вывода в Java представлены достаточно обширной иерархией. Основными классами, описывающими поведение байтового ввода-вывода являются абстрактные классы InputStream и OutputStream. Они не реализуют конкретные механизмы для работы с потоками байтов, а делегируют это своим подклассам, например FileInputStream/FileOutputStream.

Для того, чтобы понять, как работать с данной функциональностью, вам необходимо реализовать приложение анализа сигнатуры произвольных файлов. Такая сигнатура позволяет определить тип содержимого файла и представляет собой набор “магических чисел”. Данные числа, как правило, размещаются в начале файла. Например, для файлов типа PNG сигнатура представлена первыми восемью байтами файла, которые одинаковы для всех изображений формата PNG:

```
89 50 4E 47 0D 0A 1A 0A
```

Вам необходимо реализовать приложение, принимающее на вход файл `signatures.txt` (вы должны описать его самостоятельно, название файла явно указывается в коде программы). Он содержит список типов файлов и соответствующих им сигнатур в формате HEX. Например (необходимо соблюсти указанный формат данного файла):

```
PNG, 89 50 4E 47 0D 0A 1A 0A
GIF, 47 49 46 38 37 61
```

Ваша программа во время своей работы должна принимать полные пути к файлам на жестком диске и сохранять тип, которому соответствует сигнатура того или иного файла. Результат работы программы должен быть записан в файл result.txt. Если сигнатура не может быть определена, результат работы - UNDEFINED (в файл информацию в этом случае выводить не нужно).

Пример работы приложения:

```
$java Program
-> C:/Users/Admin/images.png
```

```
PROCESSED
-> C/Users/Admin/Games/WoW.iso
PROCESSED
-> 42
```

Содержимое файла result.txt (данный файл заливать как результат не требуется):

```
PNG
GIF
```

Примечания:

- Анализ сигнатуры файла позволит достоверно определить тип содержимого. Поскольку расширение файла, которое содержится в названии (например, image.jpg), можно изменить простым переименованием файла.
- В файле signatures должно быть не менее 10 различных форматов для анализа

Chapter IV
Exercise 01 - Words

Exercise 01: Words	
Turn-in directory	ex01
Files to turn-in	*.java
Разрешения	
Типы	Java Collections API, Java IO

Помимо классов, назначением которых является работа с байтовыми потоками, в Java предусмотрены классы, упрощающие работу с потоками символов (`char`). К ним относятся абстрактные классы `Reader/Writer`, а также их реализации - `FileReader/FileWriter` и т.д. Особый интерес представляют классы `BufferedReader/BufferedWriter`, ускоряющие работу с потоками за счет механизмов буферизации.

Сейчас вам необходимо реализовать приложение, которое определит величину схожести между текстами. Самый простой и очевидный способ выяснить, насколько же похожи тексты - это проанализировать частоту встречаемости одних и тех же слов.

Пусть, имеется два текста следующего содержания:

```
1. aaa bba bba a  c c c
2. bba a a a bb xxx
```

Составим на основе этих текстов словарь, содержащий все слова из этих текстов:

```
a, aaa, bb, bba, ccc, xxx
```

Теперь сформируем два вектора, длина которых равна длине словаря. На каждой i-ой позиции векторов отразим, насколько часто повторяется i-ое слово словаря в первом и втором текстах:

```
A = (1, 1, 0, 2, 1, 0)
```

```
B = (3, 0, 1, 1, 0, 1)
```

Таким образом, каждый из этих векторов характеризует текст с точки зрения частоты появления в них слов из словаря. Схожесть между векторами определим через косинусную меру по формуле:

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

Таким образом, значение `similarity` для данных векторов составит:

```
Числитель A . B = (1 * 3 + 1 * 0 + 0 * 1 + 2 * 1 + 1 * 0 + 0 * 1) = 5
```

```
Знаменатель ||A|| * ||B|| = sqrt(1 * 1 + 1 * 1 + 0 * 0 + 2  * 2 + 1 * 1 + 0 * 0) * sqrt(3 * 3 + 0 * 0 + 1 * 1 + 1 * 1  + 0 * 0 + 1 * 1) = sqrt(7) * sqrt(12) = 2.64 * 3.46 = 9.1
```

```
similarity = 5 / 9.1 = 0.54
```

Ваша задача - реализовать приложение, принимающее на вход два файла (оба файла передаются как аргументы командной строки) и показывающее результат сравнения их схожести (косинусную меру).

Помимо этого, программа должна формировать файл dictionary.txt, содержащий словарь, построенный на основе данных файлов.

Пример работы приложения:

```
$ java Program inputA.txt inputB.txt
```

```
Similarity = 0.54
```

Примечания:

1. Размер файлов ограничен 10 МБ.
2. Файлы могут содержать символы, не являющиеся буквами.

Chapter V
Exercise 02 - File Manager

Exercise 02: File Manager	
Turn-in directory	ex02
Files to turn-in	*.java
Разрешения	
Типы	Java Collections API, Java IO, Files, Paths, etc.

Реализуем утилиту для работы с файлами. Приложение должно выводить информацию о файлах, содержимом папок, их размере, и предоставлять возможность перемещения/переименования. По сути - приложение имитирует работу командной строки Unix-подобных систем.

Программа в качестве аргумента должна принимать абсолютный путь к директории, в которой мы начинаем работать, и поддерживать следующие команды:

`mv ЧТО КУДА` - позволяет перенести файл, или переименовать его, если КУДА - содержит имя файла без указания пути.

`ls` - показывает содержимое текущей папки (имена файлов и подпапок, их размер в KB)

`cd ИМЯ_ПАПКИ` - изменяет текущую директорию

Пусть на диске C:/ (или в корневой директории, в зависимости от ОС) имеется директория MAIN со следующей иерархией:

- `MAIN`
    - `folder1`
        - `image.jpg`
        - `animation.gif`
    - `folder2`
        - `text.txt`
        - `Program.java`

Пример работы программы для директории `MAIN`:

```
$ java Program --current-folder=C:/MAIN
C:/MAIN
```

```
-> ls
folder1 60 KB
folder2 90 KB
-> cd folder1
C:/MAIN/folder1
-> ls
image.jpg 10 KB
animation.gif 50 KB
-> mv image.jpg image2.jpg
-> ls
image2.jpg 10 KB
animation.gif 50 KB
-> mv animation.gif ../folder2
-> ls
image2.jpg 10 KB
-> cd ../folder2
C:/MAIN/folder2
-> ls
text.txt 10 KB
Program.java 80 KB
animation.gif 50 KB
-> exit
```

Примечание:
Вы должны проверить работоспособность программы на своем наборе файлов/папок.

CHECKLIST:
https://docs.google.com/document/d/1VrX_fdxZz0kpEm8RgBKVeT9uimAZ32krBPdhwsfqI-I/edit?usp=sharing