

Модуль 02 - Бассейн Java

Ввод/Вывод, файлы

Резюме: сегодня вы узнаете, как использовать ввод/вывод в Java и реализовать программы для управления файловой системой.

Содержание

I Предисловие

II Инструкции

III Упражнение 00: Подписи файлов

IV Упражнение 01: Слова

V Упражнение 02: Файловый менеджер

Глава I

Предисловие

Операции ввода-вывода играют важную роль в развитии корпоративной системы. Часто необходимо реализовать функционал загрузки и обработки пользовательских файлов, отправки различных документов по почте и т.д.

Видимо, ввод/вывод никогда не сводится к работе с файловой системой. Любое клиент-серверное взаимодействие между приложениями подразумевает операции ввода-вывода. Например, технология Java Servlets, используемая в веб-разработке, позволяет форматировать страницы HTML с помощью класса `PrintWriter`.

Важно помнить, что функциональность ввода-вывода не ограничивается стеком ввода-вывода Java. Существует множество библиотек, которые значительно упрощают взаимодействие с потоками данных. Apache Commons IO является одним из них.


Глава II

Инструкции

- Используйте эту страницу как единственную ссылку. Не слушайте никаких слухов и домыслов о том, как приготовить свой раствор.
- Теперь у вас есть только одна версия Java, 1.8. Убедитесь, что на вашем компьютере установлены компилятор и интерпретатор этой версии.
- Вы можете использовать IDE для написания и отладки исходного кода.
- Код читается чаще, чем пишется. Внимательно прочитайте документ, в котором приведены правила форматирования кода. При выполнении каждой задачи убедитесь, что вы следуете общепринятым стандартам Oracle.
- Комментарии в исходном коде вашего решения запрещены. Они затрудняют чтение кода.
- Обратите внимание на разрешения ваших файлов и каталогов.
- Для оценки ваше решение должно находиться в вашем репозитории GIT.
- Ваши решения будут оценены вашими товарищами.
- Вы не должны оставлять в своем каталоге никаких других файлов, кроме тех, которые явно указаны в инструкциях к упражнению. Рекомендуется изменить ваш .gitignore, чтобы избежать несчастных случаев.
- Когда вам нужно получить точный вывод в ваших программах, запрещается отображать предварительно рассчитанный вывод вместо правильного выполнения упражнения.
- Есть вопрос? Спросите у соседа справа. В противном случае попробуйте с соседом слева.
- Ваш справочник: гугля/интернет/гугл. И еще кое-что. На Stackoverflow есть ответ на любой вопрос, который у вас может возникнуть. Научитесь правильно задавать вопросы.
- Внимательно прочитайте примеры. Они могут требовать вещи, которые иначе не указаны в теме.
- Используйте "System.out" для вывода

Глава III

Упражнение 00: Подписи файлов

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

Классы ввода/вывода в Java представлены широкой иерархией.

Ключевыми классами, описывающими поведение ввода/вывода байтов, являются абстрактные классы `InputStream` и `OutputStream`.

Они не реализуют специальные механизмы для обработки потоков байтов, а делегируют их своим подклассам, таким как `FileInputStream/FileOutputStream`.

Чтобы понять, как использовать этот функционал, вам следует реализовать приложение для анализа сигнатур произвольных файлов.

Эта подпись позволяет определить тип содержимого файла и состоит из набора «магических чисел». Эти номера обычно располагаются в начале файла.

Например, сигнатура для файла типа PNG представлена первым восемью байтами файла, которые одинаковы для всех изображений PNG:

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

Вам необходимо реализовать приложение, принимающее на вход файл `signatures.txt` (его следует описать самостоятельно, имя файла явно прописано в коде программы).

Он содержит список типов файлов и их соответствующих подписей в формате HEX. Пример (необходимо соблюдать указанный формат этого файла):

PNG, 89 50 4E 47 0D 0A 1A 0A

GIF, 47 49 46 38 37 61

Во время выполнения ваша программа должна принимать полные пути к файлам на жестком диске и сохранять тип, которому соответствует сигнатура файла. Результат выполнения программы должен быть записан в файл `result.txt`. Если сигнатура не может быть определена, результатом выполнения является `UNDEFINED` (никакая информация не должна записываться в файл).

Пример работы программы:

```
$java Program
-> C:/Users/Admin/images.png
PROCESSED
-> C:/Users/Admin/Games/WoW.iso
PROCESSED
-> 42
```

Содержимое файла result.txt (загружать этот файл в итоге не нужно):

PNG


GIF

Заметки:

- Мы можем точно определить тип содержимого, проанализировав подпись файла, поскольку расширение файла, содержащееся в имени (например, image.jpg), можно изменить, просто переименовав файл.
- Файл сигнатур должен содержать не менее 10 различных форматов для анализа.

Глава IV

Упражнение 01: Слова

	Exercise 01
Words	
Turn-in directory : <i>ex01/</i>	
Files to turn in : *.java	
Allowed functions : All	
Recommended types : Java Collections API, Java IO	

В дополнение к классам, предназначенным для обработки потоков байтов, в Java есть классы для упрощения обработки потоков символов (char). К ним относятся абстрактные классы Reader/Writer, а также их реализации (FileReader/FileWriter и т.д.).

Особый интерес представляют классы BufferedReader/BufferedWriter, которые ускоряют обработку потока с помощью механизмов буферизации.

Теперь нужно реализовать приложение, которое будет определять уровень сходства между текстами. Самый простой и очевидный способ сделать это — проанализировать частоту встречаемости одних и тех же слов.

Предположим, что у нас есть два следующих текста:

- aaa бба бба а ссс
- бба а а бб xxx

Давайте создадим словарь, содержащий все слова в этих текстах:
а, aaa, бб, бба, ссс, xxx

Теперь давайте создадим два вектора с длиной, равной длине словаря. В *i*-й позиции каждого вектора, мы будем отражать частоту встречаемости *i*-го слова в нашем словаре в первом и последнем 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 встречаемости слов из нашего словаря.

Определим сходство между векторами по следующей 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 сходства для этих векторов равно:

```
Numerator A · B = (1 * 3 + 1 * 0 + 0 * 1 + 2 * 1 + 1 * 0 + 0 * 1) = 5
Denominator ||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 цель — реализовать приложение, которое принимает два файла в качестве входных данных (оба файла передаются в качестве аргументов командной строки) и отображает результат их сравнения сходства (косинусная мера).

Программа также должна создать файл Dictionary.txt, содержащий словарь на основе этих файлов.

Пример работы программы:


```
$ java Program inputA.txt inputB.txt
Similarity = 0.54
```

Заметки:

1. Максимальный размер этих файлов 10 МБ.
2. Файлы могут содержать небуквенные символы.

Глава V

Упражнение 02. Файловый менеджер

	Exercise 02
	File Manager
	Turn-in directory : <i>ex02/</i>
	Files to turn in : *.java
	Allowed functions : All Recommended types : Java Collections API, Java IO, Files, Paths, etc.

Давайте реализуем утилиту для работы с файлами. Приложение должно отображать информацию о файлах, содержимом и размере папок, а также предоставлять функции перемещения/переименования. По сути, приложение эмулирует командную строку Unix-подобных систем.

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

- **mv ЧТО КУДА** - позволяет передать или переименовать файл, если КУДА содержит имя файла без пути.
- **ls** - отображает текущее содержимое папки (имена и размеры файлов и подпапок в КБ)
- **cd FOLDER_NAME** - изменяет текущую директорию

Предположим, что на диске C:/ (или в корневом каталоге, в зависимости от ОС) есть папка MAIN со следующей иерархией:

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

Пример работы программы для ГЛАВНОЙ директории:

```
$ 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 протестировать функциональность программы, используя собственный набор файлов/папок.