



Module 02 – Piscine Java

IO, Files

Summary: Today you will learn how to use input/output in Java and implement programs to manipulate a file system

Contents

I	Foreword	2
II	Instructions	3
III	Exercise 00 : File Signatures	5
IV	Exercise 01 : Words	7
V	Exercise 02 : File Manager	9

Chapter I

Foreword

Input/output operations play an important role in corporate system development. It is often necessary to implement functionality for loading and processing user files, sending various documents by mail, etc.

Apparently, input/output never boils down to working with a file system. Any client/server interaction between applications implies input/output operations. For example, Java Servlets technology used in web development enables to format HTML pages using `PrintWriter` class.

It is important to remember that the input/output functionality is not limited to Java IO stack. There are many libraries that greatly simplify interaction with data flows. Apache Commons IO is one of them.

Chapter II


Instructions

- Use this page as the only reference. Do not listen to any rumors and speculations about how to prepare your solution.
- Now there is only one Java version for you, 1.8. Make sure that compiler and interpreter of this version are installed on your machine.
- You can use IDE to write and debug the source code.
- The code is read more often than written. Read carefully the [document](#) where code formatting rules are given. When performing each task, make sure you follow the generally accepted [Oracle standards](#)
- Comments are not allowed in the source code of your solution. They make it difficult to read the code.
- 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. And one more thing. There's an answer to any question you may have on Stackoverflow. Learn how to ask questions correctly.
- Read the examples carefully. They may require things that are not otherwise specified in the subject.
- Use "System.out" for output

- And may the Force be with you!
- Never leave that till tomorrow which you can do today ;)

Chapter III

Exercise 00 : File Signatures

	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	

Input/output classes in Java are represented by a broad hierarchy. The key classes describing byte input/output behavior are abstract classes `InputStream` and `OutputStream`. They do not implement specific mechanisms for byte flows processing, rather delegate them to their subclasses, such as `FileInputStream`/`FileOutputStream`.

To understand the use of this functionality, you should implement an application for analyzing signatures of arbitrary files. This signature allows to define file content type and consists of a set of "magic numbers." These numbers are usually located in the beginning of the file. For example, a signature for PNG file type is represented by first eight bytes of a file that are equal for all PNG images:

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

You need to implement an application that accepts the `signatures.txt` as an input (you should describe it on your own; the file name is explicitly stated in the program code). It contains a list of file types and their respective signatures in the HEX format. Example (specified format of this file must be adhered to):

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

During execution, your program shall accept full paths to files on hard disk and keep the type which file signature corresponds to. The result of program execution should

be written to result.txt file. If a signature cannot be defined, the execution result is UNDEFINED (no information should be written into the file).

Example of program operation:

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

Contents of result.txt file (there is no need to load this file as a result):

PNG


GIF

Notes:

- We can accurately determine the content type by analyzing the file signature, since the file extension contained in the name (e. g. image.jpg) can be changed by simply renaming the file.
- The signatures file shall contain at least 10 different formats for analysis.

Chapter IV

Exercise 01 : Words

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

In addition to classes designed to handle byte flows, Java has classes to simplify handling of character flows (char). These include abstract classes Reader/Writer, as well as their implementations (FileReader/FileWriter, etc.).

Of special interest are BufferedReader/BufferedWriter classes which accelerate flow handling via buffering mechanisms.

Now you need to implement an application that will determine the level of similarity between texts. The simplest and most obvious method to do this is to analyze the frequency of occurrence of the same words.

Let's assume that we have two following texts:

- aaa bba bba a ccc
- bba a a a bb xxx

Let's create a dictionary that contains all words in these texts:

a, aaa, bb, bba, ccc, xxx

Now let's create two vectors with length equal to that of the dictionary. In *i*-th position of each vector, we shall reflect the frequency of occurrence of the *i*-th word in our dictionary in the former and latter texts:

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

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

Thus, each of these vectors characterizes the text in terms of frequency of occurrence of words from our dictionary. Let's determine the similarity between vectors using the following formula:

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

Thus, similarity value for these vectors is:

```
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
```

Your goal is to implement an application that accepts two files as an input (both files are passed as command-line arguments) and displays their similarity comparison outcome (cosine measure).

The program shall also create dictionary.txt file containing a dictionary based on these files.

Example of program operation:


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

Notes:

1. Maximum size of these files is 10 MB.
2. Files may contain non-letter characters.

Chapter V

Exercise 02 : File Manager

	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.

Let's implement a utility handling the files. The application shall display information about the files, folder content and size, and provide moving/renaming functionality. In essence, the application emulates a command line of Unix-like systems.

The program shall accept as an argument the absolute path to the folder where we start to work, and support the following commands:

`mv WHAT WHERE` - enables to transfer or rename a file if WHERE contains a file name without a path.

`ls` - displays the current folder contents (file and subfolder names and sizes in KB)

`cd FOLDER_NAME` - changes the current directory

Let's assume there is MAIN folder on disk C:/ (or in the root directory, depending on OS) with the following hierarchy:

- MAIN
 - folder1
 - * image.jpg
 - * animation.gif
 - folder2

- * text.txt
- * Program.java

Example of the program operation for MAIN directory:

```
$ 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
```

Note:

You should test the program functionality using your own set of files/folders.