

## Lab 10 – Input/Output

On Moodle, you can find an archive with assets related to the following exercises.

### Exercise 1 – The Twelve Days Of Christmas

*The Twelve Days of Christmas* is an English Christmas carol in the form of a cumulative song. In each verse, a different gift that will be given is added, and the gifts of the previous days are repeated:

*On the First day of Christmas,  
My true love sent to me  
A Partridge in a Pear Tree.*

*On the Second day of Christmas,  
My true love sent to me  
Two Turtle Doves, and  
A Partridge in a Pear Tree.*

*On the Third day of Christmas,  
My true love sent to me  
Three French Hens,  
Two Turtle Doves, and  
A Partridge in a Pear Tree.*

...

Among the assets, you can find the file `xmas12.txt`, which contains for each of these 12 days a row with an ordinal number representation (first, second, third, ...) and the corresponding gift. The ordinal and the gift are separated by a tab (`\t`).

Print to the console the whole lyrics of the carol, similar to the excerpt above.

#### Hint

The class `BufferedReader` might be of use. The method `String[] split(String delimiter)` of the `String` class may also come handy.

### Exercise 2 – To Buffer Or Not To Buffer

Among the assets, you can find the class `FileGenerator`, which can generate a file of a given size of random characters. It expects as command line arguments the name of the file to be generated as well as its expected size. There is also a sample output file `test.txt` of 1 MB, generated by this class.

Write a program that takes as command line argument the name of a file and that copies this file into a second file copy. The program copies the file once with buffering (using `BufferedInputStream`, `BufferedOutputStream`) and once without. Compare the execution times of both implementations. Confirm your observations with files of different sizes that you may generate with the `FileGenerator`.

### Exercise 3 – Multiple Output Stream

In this exercise, you will write a custom output stream that will clone the output to several different output streams.

Write a class `MultipleOutputStream`, subclass of `OutputStream`. The constructor takes an arbitrary number of `OutputStream` objects (cf. `varargs`). Each writing shall be done on all the different output streams.

Take care which methods of the abstract superclass `OutputStream` you need to implement or override by looking into the source code of the `OutputStream` class (which can also be found among the assets).

Write a main program to test your implementation with different output streams (e.g. `System.out`, `GZipOutputStream`, ...).

### Exercise 4 – find & grep

You know the `find` & `grep` commands from OPERATING SYSTEMS 1. In this exercise, we will mock the following behavior from the shell in a Java program:

```
$ find directory -type f -name "*.extension"  
-print -exec grep "expression" {} \; >  
output
```

Write a program that takes as a command line argument a directory path, a file extension, words to search for and an output path. If  $n$  command line arguments are given, then the first one is the directory path, the second one the file extension, the next  $n - 3$

## Lab 10 – Input/Output

arguments build a space-separated string that will be searched for in the retrieved files. The last argument will be the output path. Throw an `IllegalArgumentException` if the first argument is not a directory path or does not exist.

The `find` method will search for files with the file extension inside the given directory path and all its subfolders. It returns a list of matching files. Only consider those files that can be read.

The `grep` method takes this list of files, the expression to look for and the output path. You can assume that the files contain regular text (such as header or source files, as opposed to binary files). The lines which contain the expression will be output, together with the file name, to the indicated output path. If latter is `/dev/pts/0` (mocking a Linux pseudo terminal), then write the output to the console. Otherwise, it will be considered as a usual file path.

The main program will execute the `grep` method on all the retrieved files from the `find` method.

### Hint

The following methods might be useful:

**Arrays:** `T[] copyOfRange(T[] original, int from, int to)`

#### String:

- `String join(CharSequence delimiter, CharSequence... elements)`
- `boolean endsWith(String suffix)`

#### File:

- `File[] listFiles()`
- `boolean isDirectory()`
- `boolean canRead()`

**List<E>:** `boolean addAll(Collection<? extends E> c);`

You may also be interested in the `PrintWriter` class.


### Exercise 5\* – Popular Serialization Formats

You have seen in the lecture the serialization and deserialization of Java objects with `ObjectInputStream/ObjectOutputStream` and the `Serializable` interface.

The resulting binary representation is, however, Java-specific and imposes issues when being used in a context with different programming languages. Other formats/metalanguages such as *XML* (*eXtensible Markup Language*) or *JSON* (*JavaScript Object Notation*) are often used in such distributed scenarios.

In this exercise, you will:

- learn how to read data from a remote (Web) resource
- learn how to embed third-party libraries in an Eclipse project
- learn how to deserialize an XML file into Java objects using the `Xstream`<sup>1</sup> library
- learn how to serialize Java objects in a JSON representation with the `GSON`<sup>2</sup> library by Google

- 1° Create the classes `Teacher`, `PhoneNumber` and `TeachingTeam` as shown in the UML class diagram in figure 1. Note that for this exercise, it is crucial that you respect the names of the attributes as given in the UML. Implement appropriate `toString()` methods for visualization on the console.
- 2° Among the assets, you can find the `.jar` archives of the `Xstream` and `GSON` libraries. To be able to use them in your Eclipse project, create a directory `lib` at the root of your project (at the same level as the already existing `src` and `bin` directories) and copy both `.jar` archives into it. Right-click on your project and select `Refresh`, then `Build Path`  `Configure Build Path ...`. Under the `Libraries` pane, select `Add JARs...`. Add both `.jar` archives and select `OK`, `Apply`, `OK`. You should now be able to import and use classes from both libraries.
- 3° In the main program, "download" the XML file retrievable under `https://coast.uni.lu/teaching/programming1/team.xml` by instantiating a `URL` object and reading the (textual) data from the `InputStream` returned by the `openStream()` method of the `URL` class. You can assume that the content can be read in a single line.

<sup>1</sup><http://x-stream.github.io>

<sup>2</sup><https://github.com/google/gson>

## Lab 10 – Input/Output

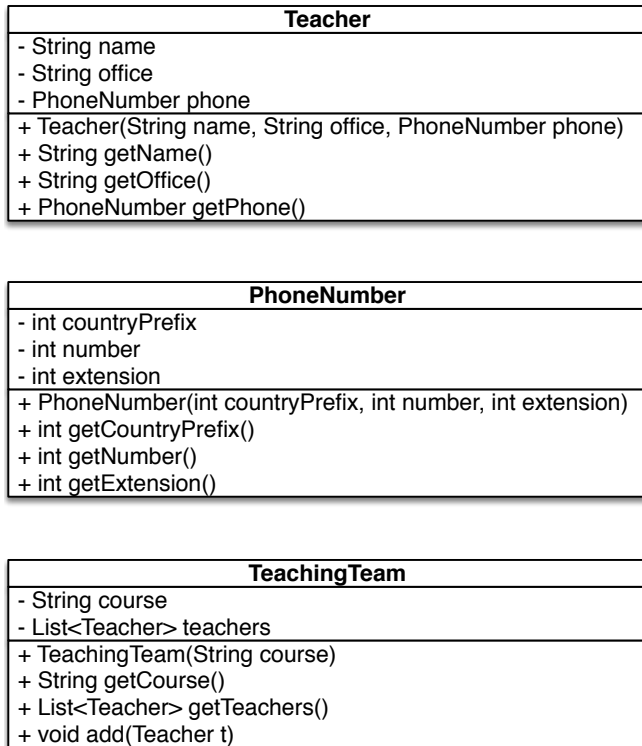


Figure 1 – UML class diagram

4° As you can see when opening the XML file in a browser, the content comprises a <team> node with different <teacher> child nodes and their related attribute nodes. You will be able to deserialize the whole as a TeachingTeam object.

First, instantiate an Xstream object:

```
1 Xstream xstream = new XStream(new
    StaxDriver());
```

Set the necessary *aliases* (a term coined by the Xstream API documentation) to map the XML tag names to the respective classes:

```
1 xstream.alias("team", TeachingTeam.class)
;
2 xstream.alias("teacher", Teacher.class);
3 xstream.alias("phone", PhoneNumber.class)
;
```

Assuming you have stored the content of the XML file in an String variable xml, you may now use

```
1 TeachingTeam team = (TeachingTeam)
    xstream.fromXML(xml);
```

to retrieve the TeachingTeam object. Print the object to the console to verify the correct deserialization.

5° Finally, serialize the TeachingTeam object again, but this time in JSON format using the GSON library:

```
1 Gson gson = new GsonBuilder().
    setPrettyPrinting().create();
2 String jsonRepresentation = gson.toJson(
    team);
```

Write the jsonRepresentation to a file team.json.

For further information, please visit the documentation of both libraries. Feel free to play around with them in more complex scenarios to evaluate their advantages and limitations.