Java Practical Work: Fundamentals

The Goal of this practical work is to assess the student's ability to write and complete Java code, to show his operational skills in real situation.

This Practical Work is composed of 4 domains

- Data modeling
- Serialization / Deserialization
- Business Logic Implementation
- Java Data Base Connectivity

Each domain needs to be completed to go on, as those domains are ordered from the most general to the most specific.

Remark: you'll have a global bonus If you follow good coding practices, like putting comments on critical code, using loggers, Javadoc, code formatting, naming conventions, code organization…

Remark : all the Test* classes defined later in this document will have a method "public static void test()".

Domain 1: Data Modeling (15 minutes - 3 points)

Exercise DMO1

*5 minutes **1pt***

Create a new Java Project that contains runnable class in the appropriate package. This class should be named "**Launcher**"

Exercise DMO2

*10 minutes 2**pt***

a. Create a new Java Class named "**Person**" that will represent a person bio-characteristics, following this provided UML Class

The Sex attribute can have 2 values ("M" or "F"), all the numerical values (age, height, weight) are integers.

b. Create the appropriate constructor(s) and handle encapsulation.
c. Add a **toString()** method that could return the following kind of output when called on an instance:

```
"Person [name="Alex", weight="150"]";
```

d. Create a Test class named **TestDMO2** that will create an instance of Person that will produce the previous output in the console.
Add the test execution to the main execution by adding the following line to the main method:

```
// Previous code

TestDMO2.test() ;
```

Domain 2: Deserialization / Serialization (55 minutes – 9 points)

Exercise SER1

20 minutes ***3pts***

a. Consider the following text file provided in the "subject" folder (data.csv)

```
"Name",     "Sex", "Age", "Height (in)", "Weight (lbs)"
"Alex",       "M",   41,       74,      170
"Bert",       "M",   42,       68,      166
"Carl",       "M",   32,       70,      155
"Dave",       "M",   39,       72,      167
"Elly",       "F",   30,       66,      124
"Fran",       "F",   33,       66,      115
"Gwen",       "F",   26,       64,      121
"Hank",       "M",   30,       71,      158
"Ivan",       "M",   53,       72,      175
"Jake",       "M",   32,       69,      143
"Kate",       "F",   47,       69,      139
"Luke",       "M",   34,       72,      163
"Myra",       "F",   23,       62,       98
"Neil",       "M",   36,       75,      160
"Omar",       "M",   38,       70,      145
"Page",       "F",   31,       67,      135
"Quin",       "M",   29,       71,      176
"Ruth",       "F",   28,       65,      131
```

b. Create a Test class named "TestSER1" that contains a test() method, that will read all the lines from the file and will display the 2nd line of the file in the console.

c. Add the test execution to the main method in the launcher class as the following

```
// Previous code

TestSER1.test() ;
```

Hints

- You can use the ***split(",")*** method on each line to extract the sub parts of the csv line
- You can use the ***trim()*** method on each part to eliminate the unnecessary blank spaces (don't forget to work on the result of this call, the original variable is left untouched)

Exercise SER2

20 minutes ***3pts***

a. Consider the same data.csv file

b. Create a ***PersonCDVDAO*** class in the appropriate package and create a ***readAll()*** method in it. Inspire from the ***TestSER1*** test method code and place it under the ***readAll()*** method . The ***readAll()*** method should return a list of Person instances, sorted by their height.
c. Create a ***TestSER2*** class with a ***test()*** method that will invoke the ***PersonCSVDAO.readAll()*** instance and that will display it in the console.
d. Add the test execution to the main method in the launcher class as the following

```
// Previous code
TestSER2.test() ;
```

Hints
You can use ***Collections.sort(<your_list>)*** to sort the List of ***Person***, or perform the sorting by "manually" implementing it.

Exercise SER3

15 minutes ***3pts***

a. Consider the previous ***PersonCSVDAO*** class.
b. Create a method "***writeAll(List<Person> persons)***" in that class that will produce the same CSV file but with different Column order like the following.
c. The output should be placed in **data_output.csv**

The order of the columns was :

```
"Name",     "Sex", "Age", "Height (in)", "Weight (lbs)"
```

And should now be :

```
"Name",     "Height (in)", "Weight (lbs)", "Age", "Sex"
```

d. Create a test class TestSER3 that will invoke the ***readAll()*** method to get a List<Person> variable, and then use the ***writeAll()*** method to write this as an output.
e. Add the test execution to the main method in the launcher class as the following

```
// Previous code
TestSER3.test() ;
```

Domain 3. Business Logic Implementation (15 minutes – 3 points)

For that domain, if you were not successful to read from the csv file (domain 2 exercises), you can consider working on a “handmade” list of Persons.

Exercise BLI1

15 minutes ***3pts***

a. Write a class PersonDataService that will perform some statistics / filtering on the persons data through the following methods:
   1. ***`int averageAge(List<Person> persons)`***
      Calculate the mean age of the persons list.

   2. ***`List<Person> filter(List<Person> persons, int thresholdAge)`***
      Returns the list of persons which are specifically above the given threshold age.

   3. ***`int calculateYearOfBirth(Person person)`***
      returns the birth year of the given person.

b. Write a class named ***`TestBLI1`*** with a *`test()`* method that will contain the following use cases :
   1. Call to the ***`averageAge`*** function on the person list with output of the result in the console.
   2. Call to the ***`filter`*** on the persons list with **32** as a threshold, with output of the list size in the console.
   3. Call to the ***`calculateYearOfBirth`*** on the 1st person in the list (Alex) with output of the result to the console.

c. Invoke that test() method in the main method of your launcher class

```
// Previous code
TestBLI1.test() ;
```

Domain 4. JDBC (35 minutes – 5 points)

The goal of this domain is to put the data present in the CSV file in a relational database using java

Exercise JDB1

15 minutes ***2pt***

a. Create a class ***`TestJDB1`*** with a ***`test()`*** method that will be able to connect to an in-memory embedded h2 database (driver provided in the "subject" folder)
b. Complete the ***`test()`*** method by preparing a statement that will create the table "***`PERSONS`***" with the appropriate columns names, types, and constraints (you can add an auto_increment feature on a new "id" column)

c. Invoke that test() method in the main method of your launcher class

```
// Previous code

TestJDB1.test() ;
```

Hints

- To setup the connection, you need to include the h2 jdbc driver as seen during lectures.
  When using eclipse, you have to go in the **Project Properties** > **Build Path** and add the h2 jar (provided in the subject folder) in the **Libraries** tab.
- To open a connection in an in-memory h2 database, one have to connect to this url:
  `jdbc:h2:mem:testdb`
- Important : remember that the in-memory db will lose all the data between two connections, so beware to reuse the same connection or you will get unexpected behavior.

Exercise JDB2

20 minutes ***3pt***

a. Create a ***`PersonJDBCDAO`*** class that will contain a ***`create()`*** and a ***`search()`*** methods
b. Write a TestJDB2 test class that will gather the following operations:
   1. From the csv file, read the persons list
   2. For each person in the list, call the ***`create()`*** method from the ***`PersonJDBCDAO`***
   3. Call the ***`search()`*** method to validate the fact that everything is well inserted.
   4. Display the result list in the console
c. Add the following code to your launcher main method

```
// Previous code

TestJDB2.test() ;
```