Practical Work: Advanced Java Programming – Spring 2021

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 6 domains:

1. Maven
2. Junit
3. DI with Spring
4. JPA with Hibernate
5. REST with JAX-RS or Spring Boot

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

For each resource which is asked to be created, you have to replace **${id}** by your Epita id, replacing the "." (dot) by a "_" (underscore).

You'll have to deliver all the projects created for that practical workshop; you must place all of them in the same folder.

Remark: You will have a global bonus If you follow good coding practices, like putting comments on critical code, using loggers instead of System.out and so on.

This test is partly automatically evaluated, and relies on maven to build your project and run the test classes.

Please ensure that you see all your code running in maven, through unit test cases run during the maven build.

In the home/subject folder, you will have access to the 4 csv files necessary to complete this practical work

- patients.csv
- medications.csv
- insurances.csv
- prescriptions.csv

Read carefully all this document before starting, it's important to have a good overview of this exam! Do your best to implement each exercise correctly before moving to the next one.

Final format for your delivery (replace ${id} by your Epita id – replace dot by underscore in your id - , replace “your-packages” and “your-classes” by your own code.

```
/home/submission
       +- ${id}-adv-java
              +- output
              +- src/main/java
                     +- (your-packages).(your classes)
              +- src/test/java
                     +- TestMVN2.java
                     +- TestJUN1.java
                     +- TestJUN2.java
                     +- TestJUN3.java
                     +- TestSPR2.java
                     +- TestSPR3.java
                     +- TestSPR4.java
                     +- TestJPA1.java
                     +- TestJPA2.java
                     +- TestJPA3.java
                     +- TestJAX1.java
                     +- TestJAX2.java
              +- src/test/resources
                     +- patients.csv
                     +- medications.csv
                     +- …
              +- pom.xml
```

Domain 1: Maven (35 minutes)

Exercise MVN1

*5 minutes **0.5pts***

Build a new java project names using maven. You must enforce the maven standard layout. You must define correct maven coordinates for your project. The project name should be **${id}-adv-java** which will be referenced as the "the project" and replaced by the variable **${project}** in the rest of this document.

Exercise MVN2

*20 minutes **2pts***

a. Consider the following datasets:

```
pat_num_HC;pat_lastname;pat_firstname;pat_address;pat_tel;pat_insurance_id;pat_sub_date
"1256987452365";Martin;Bernard;Chatillon;"0106060606";2;01/10/2010
"1852458963215";Chalme;Antoine;Paris;"0105050505";1;01/01/2017
"1985236548520";Daulne;Paul;Puteaux;"0107070707";3;01/05/2008
"2365987542365";Solti;Anna;Montrouge;"0108080808";4;01/10/2010
"2658954875210";Dart;Pauline;Bourg la reine;"0109090909";4;01/01/2015
"2758965423102";Chalme;Julie;Paris;"0105050505";1;01/06/2017
```

Patients

```
insurance_id;insurance_name
1;MACIF
2;MGEN
3;MAAF
4;ADREA
5;APICIL
```

Insurances

a. Create `Patient` and `Insurance` classes
b. Put those 2 datasets in 2 different files (*patients.csv* and *insurances.csv*)
c. Create a `PatientReader` class and a `InsuranceReader` class that can read (through a function `readAll()`) from the 2 files containing the 2 previous data.

d. Create a class `TestMVN2`, containing a `main(String[] args)` which will call those 2 classes and display the list of instances (think of implementing the `toString()` method)

Exercise MVN3

5 minutes

Perform a maven build (install), store the console output in the file "**${project}/output/mvnbuild1.out**" (in the folder "output") in your project.

Exercise MVN4

5 minutes

Add the following dependencies to the project :

- Hibernate 5.4.5-Final
- Spring-context 5.1.9.RELEASE
- Spring-orm 5.1.9.RELEASE
- Spring-test 5.1.9.RELEASE
- Log4j2-core 2.12.1
- Javax-inject 1
- JUnit 4.12

Launch a build (install) and store the output in the file "**${project}/output/mvnbuild2.out**

Domain 2. Unit Testing with JUnit (20 minutes)

Exercise JUN1

5 minutes

Create a JUnit test class `TestJUN1`, make this class produce the output "*Hi from JUnit*" while building the project through Maven.

Exercise JUN2

10 minutes

Create a JUnit test named `TestJUN2` in package `fr.epita.tests.${id}` and use correctly the `@After, @AfterClass, @Before` and `@BeforeClass` to manage the instance from the class that does the deserialization from exercise **MVN2**. The Junit test itself should just display the list of Patients and Insurances present in the file.

Exercice JUN3

5 minutes

Create a JUnit test named `TestJUN3` with correct assertions to validate that the list produced by your deserialization class is matching the results that one would expect.

Remember to follow the "given-when-then" approach seen during lectures.

Domain 3. DI with Spring (35 minutes)

Exercise SPR1

10 minutes

Declare your first bean thanks to the spring xml file.

This bean should be named "`myFirstBean`", must be of type `java.lang.String` and should contain as a value `'Hello from Spring, ${id}'`. Replace ${id} by your id.

Beware of the location where you place the spring file. It should be loadable from **the root** of the classpath.

Help : the file root content is as follows : you can copy paste this content in your own file:

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
         http://www.springframework.org/schema/beans/spring-beans.xsd
         http://www.springframework.org/schema/context
         http://www.springframework.org/schema/context/spring-context.xsd
         http://www.springframework.org/schema/tx
         http://www.springframework.org/schema/tx/spring-tx.xsd
         http://www.springframework.org/schema/mvc
         http://www.springframework.org/schema/mvc/spring-mvc.xsd">

<!-- declare your beans here -->
</beans>
```

Exercise SPR2

10 minutes

Inject the previously declared bean by using the standard JEE annotations (2 annotations to be used) in a JUnit test case. This testcase has to be named "`TestSPR2`" and has to be in the package **fr.epita.tests.${id}** in the appropriate source folder.

Help : Remember that Spring and JUnit do not integrate by default, you have to make appropriate use of `@RunWith` and `@ContextConfiguration` annotations

There's a Spring Junit 4 class runner class in the Spring test dependency.

Exercice SPR3

5 minutes

Considering the Patient and Insurances classes from exercise MVN2.

a. Create a new JUnit Test named "`TestSPR3`" in the `fr.epita.tests.${id}` package.
b. Inject then an instance of Patient constructed by spring with whatever values you want, except for the Patient firstname which should be "${id}" (your id).

Exercice SPR4

10 minutes

Create an H2 embedded database instance through spring using this bean:

```
<bean id="mainDS"
            class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="url" value="jdbc:h2:mem:test;DB_CLOSE_DELAY=-1"></property>
  <property name="username" value="test"></property>
  <property name="password" value="test"></property>
  <property name="driverClassName" value="org.h2.Driver"></property>
</bean>
```

Write a testcase `TestSPR4` in which you will inject this DataSource instance, and write an assertion that shows that this Database is ready to use.

Exercise SPR5 (this exercise is optional and counts as a bonus)

25 minutes

Create the init sql scripts for the database, based on the data you have in the **MVN2** exercise.

Run them using an `@Before` annotation in a `TestSPR5` class.

Domain 4. JPA with Hibernate

Exercise JPA1

a. Correctly annotate the `Patient` and the `Insurance` classes so that you can use JPA on them.
b. Configure an `EntityManager` through the spring configuration.
Remember that an entity manager will require the:
   - **jdbc** link (datasource),
   - the packages containing entity classes,
   - the database vendor-specific configuration
c. Write a Junit test class named `TestJPA1`, containing a test method, that will check the configuration is correctly made (no error in the console)

Exercise JPA2

a. Consider the following additional data:

```
medication_code;medication_name;medication_comment
1;Advil;anti-inflammatory
2;Doliprane;paracetamol
3;Spasfon;antispasmodic
4;Smecta;gastric dressing
5;Strepsil;antiseptic
```

Medication

```
ref;presc_code;presc_pat;presc_days
1;1;"1852458963215";2
2;2;"2758965423102";4
4;1;"1852458963215";10
6;4;"2658954875210";5
8;4;"2758965423102";5
9;1;"1985236548520";2
```

Prescription

a. Create the `Medication` and `Prescription` classes,
b. Create 4 classes, containing 4 methods (search/update/delete/create), those DAOs will rely on the `EntityManager` created in **JPA1** exercise:
   - `InsuranceDAO`
   - `PatientDAO`

- `MedicationDAO`
- `PrescriptionDAO`

c. Propose a pattern that would allow you to change of implementation if necessary,
d. Reduce the code as much as possible,
e. Create a test class named `TestJPA2` that will check that the 16 methods (be smart!).

Exercise JPA3

a. Make sure the transactional aspect of your DAOs is correct.
b. Write a Test `TestJPA3` which checks that you can run in one single transaction:
   - Creation of a Medication,
   - Creation of an Insurance,
   - Creation of a Patient (associated to an insurance),
   - Creation of a Prescription (associated to the medication).

Domain 5. REST with JAX-RS or Spring Boot

In this exercise, you'll have to build a micro-service that deals with the "patient-prescription" feature of this application.

Exercise JAX1

a. Think of a use case through a web application that could use the 4 data.
b. Create a Repository class (Data Service), representing the prescription and patient needs. This repository will aggregate the DAOs from JPA3 exercise to provide an all-in-one channel to handle the UI.

Build a `TestJAX1` test that will check the Repository functioning.

Help: Remember that a goal of a Repository is to expose data under a usable form, to feed a web page or a mobile app. It should propose simple and easy-to-use apis. To completely know what is a prescription, the user has to have the medication details and the patient details fetched, in addition to the prescription own details. in one single request.

Exercise JAX2

Expose this Repository using JAX-RS or Spring boot. Create Test `TestJAX2`, that launches an instance of this microservice and triggers a collection of calls to test this web service.