Practical Work: Advanced Java Programming

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. Persistence with JPA and Hibernate
5. JAX-RS + REST
6. Angular development

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’ll 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.

# 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***

1. Create a text file “data.csv” and place those data in it :

|  |
| --- |
|  |

1. Then create a class “Passenger” with as attributes names the column titles to represent the file data.
2. Create a class (think well about the class name) able to read data from this file and transform it as a list of instances of the Passenger Class.
3. Add a method in that Class that sorts the list by the Passenger Comfort Class (PClass).
4. Create a third Class that will be able to test the deserialization feature of this program.

Think well about where you place the text file and the 3 classes.

**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
* 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, 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 passengers present in the file.

**Exercice JUN3**

*5 minutes*

Create a JUnit test 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.

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:

|  |
| --- |
|  |

**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 Passenger class from exercise MVN2, Create a new JUnit Test named “**TestSPR3**” in the **fr.epita.tests.${id}** package

Inject then an instance of Passenger constructed by spring with whatever values you want, except for the passenger name which should be “${id}” (your id)

**Exercice SPR4**

*10 minutes*

Create an H2 embedded database instance through spring using this 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.

Domain 4. JPA with Hibernate (50 minutes)

**Exercise HIB1**

*10 minutes*

Considering the following snippet that you must complete:



Replace the ***???*** by appropriate values and inject the SessionFactory instance in the testcase **TestHIB1**. Write an assertion to check that hibernate is well initialized.

**Exercise HIB2**

*10 minutes*

Annotate the Passenger class with the appropriate JPA so that it can be managed by hibernate as a persistent class.

Write a **TestHIB2** testcase where you verify that you can record all the Passenger entries read from the data.csv file, to a database thanks to JPA and Hibernate.

**Exercise HIB3**

*15 minutes*

Create a DAO class that relies on a session factory to manage data persistence of the Passenger class. For the search method, whatever property of the Passenger class can be part of the criteria.

Write a test **TestHIB3** where you will test the 4 DAO methods.

**Exercise HIB4**

*15 minutes*

Create a duplicate configuration file “applicationContextEM.xml” that uses a JPA EntityManager instead of a session factory and create another DAO that uses this entity manager.

**Help** : you can use this code and integrate it directly in your project

|  |
| --- |
|  |

Copy and paste the TestHIB3 content in a new TestHIB4 testcase, the only change is the location of the new spring file.

Domain 5. JAX-RS REST implementation (30 minutes)

**Exercise JXR1**

*5 minutes*

Create a new maven project with packaging type “war” (java web project).

This project must have this artifactId: **${id}-adv-java-web** and should have **${id}-adv-java** in its dependencies. It will be referenced as **${web-project}**

Integrate the following dependencies as well:

|  |
| --- |
|  |

Build the project with maven and copy paste the console output to this file : “**${web-project}/output/mvnbuild3.out**

**Exercise JXR2**

*15 minutes*

Create a new REST service using the JAX-RS annotations. This REST service will be able to serve a Passenger resource and provide the necessary methods to perform CRUD operation against this resource. You can take the Passenger class to represent that resource.

You must deploy your rest service on a **tomcat** server

**Help** : to integrate jersey in a web application you have to modify the web.xml file as follows



Replace the **???** by appropriate value.

Create a postman test to validate minimal connectivity. Save your postman collection named “JXR2Collection” under **${web-project}**/postman/JXR2Collection. In this collection, create one postman test per method (you should have 4 methods).

Don’t forget to save after getting the result.

**Exercise JXR3**

*10 minutes*

Reuse one of the two JPA-based DAOs and inject it in your REST service thanks to Spring.

**Help** : to load spring at application startup, you have to add the following code in the web.xml:



Create another collection named “JXR3Collection” and save it under **${web-project}**/postman/JXR3Collection. It should be the same calls than in JXR2Collection, and don’t forget to save the results of your queries.

Domain 6. Developing Angular Applications (45 minutes)

**Exercise ANG1 (Bonus)**

*45 minutes*

Develop an angular application that presents the following screens, allowing to handle Passengers data:

* Search screen (welcome page)
* Modification screen
* Deletion screen
* Creation screen

This project must be in the same folder as your eclipse workspace.