CSCI 5308 – Assignment 1: TDD

Let's pretend that you are employed by a company manufactures automobile parts. The company is in the process of implementing a REST API that allows you to receive orders for products that the company sells through authorized dealers.

The API will:

1. Authenticate the request, ensuring the request is coming from an authorized dealer.
2. Identify the dealer submitting the order.
3. **Validate the order, ensuring the dealer is ordering a valid part.**
4. **Process the order, recording it in the database for manufacturing and delivery.**
5. **Return a response to the dealer indicating success or failure.**

You work on a very large team, some team members are working on the database component of this API that stores dealer info and part info, others are working on the security and authorization for the API. Your company uses an internal 3rd party system for tracking manufacturing requests and delivery that the API will communicate with called PARTMANAGER.

You have been tasked with tasks 3, 4 and 5 from the above list. You must write logic in the business layer of the API that validates the incoming XML order. You must write logic in the business layer of the API that sends the XML order to the 3rd party system that tracks manufacturing requests and delivery (PARTMANAGER), the PARTMANAGER system will return success or failure results to your logic. You must also write logic in the business layer that formulates an XML response for the dealer.

Because you are developing your components in parallel with other members of your team, **you will use test driven development to ensure that your systems will work when integrated with the other components.** PARTMANAGER can only run in the production environment, so it must be mocked so that you can verify that your code calls it properly and responds to success and failure responses from it. The security team is very busy and won't be done until weeks after your code is ready, therefore your code must be set up to receive mocked data for security authentication results in your unit tests. The database team is also very busy, so you must mock the database logic used to validate parts.

See **Appendix A** for the XML format for incoming orders.

See **Appendix B** for the XML format for responses to be sent to the dealers.

See **Appendix C** for the interface the security team will implement, you must mock this. Your mock will simulate returning authentication successes and failures.

See **Appendix D** for the interface the 3rd party PARTMANAGER system provides. You must mock this object to simulate returning different part responses.

The database team is filled with mediocre programmers. They told you to define the interface to communicate with them and they'll just "make it work". **Define an interface and provide a mock object and mock data that your code can use to test your logic.** The interface must allow your system to validate whether a part number is a valid part number for your company.

Requirements:

1. All files you submit for this assignment must be in the private gitlab repository set up for you for this course. Go to https://git.cs.dal.ca and you will find this repository already created for you.
    1. The first thing you must do is clone this repository to your local machine. This is where you will do all of your assignment work for this semester. You may need to create the main branch, follow the instructions in gitlab to configure your repo. Add a readme.md file to check you have things working.
    2. Next, in your repository create a folder named **A1**
    3. **Put all of your java files directly in this A1 folder, no subfolders.**
    4. When you are done, commit and push your work to the **MAIN** branch. This is the code we will grade.
2. Write the "main" entry point for the API.
    1. Authenticate the dealer using the security interface.
    2. Validate the incoming order XML
        1. Validate the format of the XML, return an invalid XML response if the XML is invalid.
        2. Use the database interface to ensure the part number is valid.
    3. Submit the part to the PARTMANAGER interface.
    4. Return the proper XML response to the dealer. (valid / invalid parts, PARTMANAGER responses, etc...)
3. Because you are programming a class in the business layer, every class and method you write should be covered by a comprehensive set of unit tests. Use this assignment as an introduction to test driven development. Write your test cases first, then write the logic to satisfy the tests.
4. Ensure the mock objects and mock data you create cover **all of** the possible inputs and outputs of the API.
5. You must deliver to the TA for marking a **Java command-line program** that can take as input a file with order XML in it, and writes the result XML to a file. For example, the marker will run your program like so: *java Main.java inputfile.xml*
6. Your system must be capable of being executed by the TA to verify that there are unit tests, and that all unit tests pass. To satisfy this requirement, write your main() method so that if no command-line argument is passed with an XML file, instead it calls all of your unit tests. We will not be using JUnit for this assignment, just write your tests simply and output via System.out.println() messages of the format "PASS - <test name>" or "FAIL - <test name>".
7. The marker will compile your code via the following command **only**: *javac *.java* , to ensure your code works I suggest not using an IDE or maven for this one because otherwise it will put things in weird folders and your code won't compile. You should diligently check your code compiles by pulling your git repository to Dal Timberlea unix server and compiling and running it there.

You will be marked with the following rubric:

1. Successful implementation of the business logic – 30%
2. Comprehensiveness and quality of unit tests – 40%
3. Comprehensiveness and quality of mock objects / mock data – 30%
4. **Code does not compile: Automatic 0 for the assignment.**
5. **Code not in the A1 folder in the main branch: Automatic 0 for the assignment.**

NOTE:

Your markers will only have the BASE java SDK installed on their computers. Do not import any libraries that are not in the base SDK or your program will not compile. Use the built-in SAXParser or DOMParser libraries to parse the JSON input or your code will not compile.

Appendix A – XML format for incoming orders

```xml
<order>
        <!-- The dealer submitting the order -->
        <dealer>
                <!-- The dealer ID, identifies the dealer. Must validate that the dealer is a known
                     dealer. -->
                <dealerid>XXX-1234-ABCD-1234</dealerid>
                <!-- The dealer access key, used by the security class to authenticate the dealer. -->
                <dealeraccesskey>kkklas8882kk23nllfjj88290</dealeraccesskey>
        </dealer>
        <!-- A list of items ordered by the dealer. -->
        <orderitems>
                <!-- An item in the list of items ordered by the dealer. -->
                <item>
                        <!-- The part number to submit to PARTMANAGER. -->
                        <partnumber>1234</partnumber>
                        <!-- The quantity ordered to submit to PARTMANAGER. -->
                        <quantity>2</quantity>
                </item>
                <!-- Another item, there can be an unlimited number of items in the orderitems list. -->
                <item>
                        <partnumber>5678</partnumber>
                        <quantity>25</quantity>
                </item>
        </orderitems>
        <!-- The delivery address to send the parts to. -->
        <deliveryaddress>
                <!-- The name field can be blank. -->
                <name>Mrs. Jane Smith</name>
                <!-- The street is required. -->
                <street>35 Streetname</street>
                <!-- The city is required. -->
                <city>Halifax</city>
                <!-- The province is required. -->
                <province>NS</province>
                <!-- The postal code is required. -->
                <postalcode>B2T1A4</postalcode>
        </deliveryaddress>
</order>
```

Appendix B – XML format for dealer responses

```
<!-- The following response is sent when the dealer is not authorized. -->
<order>
        <result>failure</result>
        <error>Not authorized</error>
</order>

<!-- The following response is sent when the incoming XML is not valid. -->
<order>
        <result>failure</result>
        <error>Invalid order</error>
        <!-- Error message possible values are: -->
                <!-- Invalid delivery address -->
                <!-- Invlid order item list -->
                <!-- Invalid order item entry -->
        <errormessage>Invalid delivery address</errormessage>
</order>

<!-- The following response is sent when the dealer is authorized. -->
<order>
        <orderitems>
                <!-- An item in the list of items ordered by the dealer. -->
                <item>
                        <!-- The part number submitted to PARTMANAGER. -->
                        <partnumber>1234</partnumber>
                        <!-- The quantity submitted to PARTMANAGER. -->
                        <quantity>2</quantity>
                        <!-- Whether the company will manufacture an deliver the part. -->
                        <!-- Possible values are: success, failure -->
                        <result>success</result>
                        <!-- If the value above is "failure", this message explains the reason. -->
                        <!-- Possible values are: out of stock, no longer manufactured, invalid part -->
                        <errormessage></errormessage>
                </item>
                <item>
                        <partnumber>5678</partnumber>
                        <quantity>25</quantity>
                        <result>failure</result>
                        <errormessage>invalid part</errormessage>
                </item>
        </orderitems>
</order>
```

Appendix C – Security class interface (Java)

```
public interface Security
{
       // Returns whether the dealer is authorized to order parts on behalf of their customers.
       public boolean IsDealerAuthorized(String dealerid, String dealeraccesskey);
}
```

Appendix D – PARTMANAGER interface (Java)

```
public class DeliveryAddress
{
        public String name;
        public String street;
        public String city;
        public String province;
        public String postalCode;
}

public interface PARTMANAGER
{
        public enum PartResponse
        {
                SUCCESS,
                OUT_OF_STOCK,
                NO_LONGER_MANUFACTURED
        }

        // Submit part for manufacture and delivery.
        public PartResponse SubmitPartForManufactureAndDelivery(
                int partNumber,
                int quantity,
                DeliveryAddress deliveryAddress);
}
```