CST8277 (21S) Assignment 3: BloodBank (JPA Mapping & JUnit Tests)

Theme for Assignment 3:

After completing this assignment, you will have achieved the following:

1. Gain experience in mapping a real-world database schema to POJOs
2. Gain experience in creating complex JPA queries
3. Gain experience in building JUnit test cases
4. Use the Maven 'Surefire' plugin to generate a pass-fail report for all test cases

Download and importing the project to your Eclipse workspace:

1. Download *BloodBank-JPA-Mapping-Junit-Skeleton.zip*
2. Unzip the content to some location
3. Go to **File** menu, choose **Open Projects from File System**
4. On the **Import source**, choose the location of the project folder
5. Make sure that there is a check mark beside the project folder
6. Click **Finish**
7. To generate the *<Classname>_.java* files, right-click on the project name, select **Properties**
8. Select **JPA** on the left, and on **Canonical metamodel (JPA 2.0)**, on the **Source folder** dropdown, select src/main/java, and finally click **Apply and Close**; after you fix the errors on the classes, the *<Classname>_.java* files will be auto-generated

After you have imported the project to your Eclipse workspace, do the following steps to view all the TODOs/tasks and hints for this assignment:

1. Go to **Window** menu, select **Preferences**
2. Type "task" on the "type filter text" textbox on the left
3. Under **Java Compiler**, you should see "Task Tags"
4. Select **Task Tags** and on the right, make sure the **Enable project specific settings** is checked, and then click **New…**
5. On the **Tag** text box, type "Hint"
6. On the **Priority** dropdown, select **Low** and click **OK**
7. Click **Apply** (Note: you have to let Eclipse rebuild your project if it asks) and then click **Apply and Close**
8. If you don't see the **Tasks** on your Eclipse, do the following:
    a. Go to **Window** menu, select **Show View**, select **Other…**
    b. Under **General**, select **Tasks** and click **Open**
9. To filter the tasks, do the following:
    a. On the **Tasks** bar, select the filter icon on the right
    b. It should bring up the **Filters** window

c. Make sure that **Show all items** is not checked
d. Make sure that **TODOs** is checked
e. Under **Scope**, make sure that **On elements in selected projects** is selected
f. Then click **New…**, select **New Configuration**, click **Rename**, enter "Hints" on the **Configuration Name** pop-up, and then click **OK**
g. Again, make sure that **On elements in selected projects** is selected for Hints
h. Then under **Description**, type in "Hint" beside **Text: contains** and click **Apply and Close**
i. Then make sure that **Show all items** checkbox is checked on the **Filters** window

Approach for this assignment:

1. First of all, finish all the TODOs in the models (Person, Phone, DonationRecord, BloodDonation, BloodBank, PrivateBloodBank, PublicBloodBank); you may use the Address and Contact entities as guides/references in completing the other entities/models
2. Refer to the attached ER and class diagrams (*BloodBank-Diagrams.pdf*) to understand how the entities and tables are related to each other; you must annotate the classes/entities based on how they are related as shown in *BloodBank-Diagrams.pdf*
3. After finishing the TODOs in the models, next is to write your test cases for the rest of the models (a sample has already been provided for Contact (i.e. *TestCRUDContactOriginal.java*)); **Note:** you should be able to run *TestCRUDContactOriginal.java* after fixing the models
4. At the very least, you have to write test cases for Person, Address, Phone, & DonationRecord
5. Note that you have to also finish the TODOs in *JUnitBase.java*; the methods getTotalCount(), getAll(), getWithId(), and getCountWithId() are all optional and you do not have to complete them…however, they will greatly help reduce code repetition once you have completed them and used them in your own JUnit test classes

Note: The *BloodBankDriver.java* is the driver class for the project. It is a basic class and you do not have to do anything here. When it is executed, it initializes the database for the project. It also populates the database with some data as can be seen inside the addSampleData() method. The driver class uses the following files to initialize the database:

- *src/main/resources/META-INF/persistence.xml*
- *src/main/resources/META-INF/sql/bloodbank-drop.sql*
- *src/main/resources/META-INF/sql/bloodbank-create.sql*

Inside *persistence.xml* file, it defines the following properties:

```
<property name="javax.persistence.schema-generation.database.action" value="drop-and-
create"/>

<property name="javax.persistence.schema-generation.create-source" value="script"/>
<property name="javax.persistence.schema-generation.create-script-source"
value="META-INF/sql/bloodbank-create.sql"/>

<property name="javax.persistence.schema-generation.drop-source" value="script"/>
```

```
<property name="javax.persistence.schema-generation.drop-script-source" value="META-INF/sql/bloodbank-drop.sql"/>
```

As can be seen above, every time the application is run, it first drops the database (using the script *bloodbank-drop.sql*) and then creates it (using the script *bloodbank-create.sql*).

JUnit tests:

In this assignment, JUnitBase should serve as the base class of your JUnit test classes that you will create. The following methods are required to be implemented:

- deleteAllData()
- deleteAllFrom() – this method is called by deleteAllData() several times to delete the contents of tables in the database

The following methods below are not required to be implemented. However, completing and invoking them will greatly help reduce code repetition in completing this assignment.

- getTotalCount()
- getAll()
- getWithId()
- getCountWithId()

Once the above 4 methods are implemented, you can invoke them in your JUnit test classes.

Note: Method setupAll() (annotated with @BeforeAll) is invoked before all test cases are executed and method tearDownAll() (annotated with @AfterAll) is called after all test cases are performed.

Please refer to *TestCRUDContactOriginal.java* on how to write your JUnit test classes for the remaining entities.

To order your test cases, use the following annotations in your JUnit test class (please also see *TestCRUDContactOriginal.java*):

- `@TestMethodOrder(MethodOrderer.OrderAnnotation.class)`
- `@Order(1), @Order(2), @Order(3), …`

Ideally, each test case should be completely independent of other test cases.

Note that for each JUnit test class that you write, it should contain around eight (8) individual test cases (similar to *TestCRUDContactOriginal.java*).

Once you have completed the tasks above, do the following to build and test your assignment:

1. Right-click on your *BloodBankDriver.java* and choose **Run As** and then choose **Java Application**
2. Make sure there are no errors reported on the console
3. Then, run your JUnit tests:

a. To run your JUnit tests individually, right-click on your JUnit test class (e.g. *TestCRUDContactOriginal.java*) and choose **Run As** and then choose **JUnit Test**; make sure that all your test cases passed
b. To build your project and run all the test cases, right-click on your project, choose **Run As**, select **Maven build…**, on the **Goals** text box type "**clean install test surefire-report:report site -DgenerateReports=false**"
c. The above will generate *target/site/surefire-report.html* file which will show the results of the test cases when opened with a web browser

Submission:

Zip your entire project folder (**Note:** please do not include the target sub-folder) and submit the zip file on Brightspace/Activities/Assignments/Assignment 3. Name your zip file as follows: *<lastname>_<firstname>_<student number>_21SAssignment3.zip*

Example: yap_teddy_12345_21SAssignment3.zip

– End –