Comp7013 Project OO Programming

Completion Date: 27th August 100 % of module repeat marks.
You can arrange a demo First week in September.

Vaccination Application

Your local Vaccination center want to manage the vaccination of their Clients. The centre requires a solution which manages and stores personal information about new clients, which stores information about the particular vaccine that a client has received, and is easy to use. You will need to design a graphical user interface (GUI) which allows the vaccine centre to manage this information in a logical and intuitive way.

Your task is to write an application to manage Clients and Vaccines. Each class written should have appropriate getters and setters for each field and a toString and a .equals method.

Java Classes Required

1. A **Name** class. This stores details of a person's name
   ```
   String firstname
   String lastName
   ```

2. A **Client** class. This is a super class for all people in the Application. Its attributes are:

   ```
   Name name
   String id
   String phone
   Vaccination vac
   ```
   Note that the Name class is utilised for the name attribute here.

3. A **ClientCollection** class. This holds an ArrayList of Client objects.

   List of Clients

   Operations:
   ```
   Add Client
   Remove Client
   Show all clientlist
   Find and display a particular client
   ```

4. A **Vaccination** class. Used for single dose vaccinations.

This is a superclass with the following attributes and methods:

Attributes :
```
String name
Int efficacy //range between 1% and 100%
Date startDoseDate
```

Methods

Getters and setters for each attribute

An abstract method called deliveryInfo.

5. A SecondVaccineNeeded class. Used for two dose vaccines.
   This class is a subclass of Vaccination. It has additional attribute:
   nextVacinationDate
   It has an implementation of the abstract method delivery. It has extra get/set as required.

Note :make up the detail in delivery something unique.(e.g. give 5ml or 10ml etc.)

Design

Firstly, create a UML diagram which details all the java classes used in your application and the relationships between them. Include this in your project submission.

Part A

We want to use the above classes in an application that has JavaFX GUI as the front-end (use java not scene-builder). A file that stores serializable objects should act as the persistent storage. You can adjust the classes above if needed with new attributes or even new classes if necessary.

Your application should be able to:

1. Create a new Client.
2. Remove a Client.
3. Search for a Client by supplying the Client ID.
4. Display all Clients ordered by Vaccine type. i.e. vaccine name followed by Clients..
5. Display all Clients by ID.
6. List all Clients that are due a second dose and the date of that Dose.
7. Save entire practice to a file (via serialization). Load can be done automatically on startup.
8. Quit.

Part B.

Build the same application except this time use a database as the persistant storage.

Note:

1. The Database can have as many tables as is needed.
2. Use objects for this application (Not just strings). The methods of the class that connects to the database should take objects as parameters where appropriate.
3. Use the MVC pattern for this application and use a package structure to reflect the MVC pattern.
4. Add an extra button which creates a loop that creates dummy Client objects and adds them to some collection until the application runs out of memory. Note how long it took and the memory at the point of exception. Set the vm size to half of normal then use the same button and observe what happens/how long it takes to get the out of memory exception. You can use a package like visual vm and show screen shots if you like.
5. Write Junit testcases (at least 2) and a test-suite to test some elements of your code in your project.
6. Use Javadoc for documentation purposes (well document at least one class).
7. You can use jdbc for the database, but I would be highly impressed if you choose to use JPA for the persistence.

Rough marking guide lines:

GUI 20%

Classes and UML diagram (basic) 15%

MVC

- Controller 10%
- DBconnect 20%
- Load save serialization 10%

Package structure 5%

Javadoc 5%

Junit 5%

Vm 5%

Discretionary 5%