Software Developer Course Assessment

Quantitative Assessment Practice #2

Course Name: Software Design, Architecture, Testing

Current Week: (18/10/2021)

Introduction:

The purpose of this assessment is to help us understand how the class is doing in terms of the review material that we have covered during the previous couple of weeks. The only purpose of this assessment is for us to improve our approach to review and ensure that what we’re currently doing is an effective strategy. Completion of this assessment is mandatory - if you don’t submit a solution, it will be marked as incomplete. You must complete a minimum of 80% of your assigned QAPs per course – otherwise you will be marked as incomplete for that course no matter how good your other grades are. If you do submit a solution, it will be marked as complete, as you will receive full marks no matter what your actual performance was – again this is a participation grade.    
   
Again, the goal here is to help you all in the best way that we can, so please do be honest when answering the questions related to how long it took, which resources you used, etc. And please ensure that you do your own work – don't just copy off a friend to get it done, earnestly do your best with it. If you can’t get it completely working, give us what you have. While it will be graded, the grade will not count against you, it’s just a way for us to see where everybody is, and to know which concepts, if any, we, as a class, may be struggling with.

Deadline: You will have until the end of the day on **Wednesday Nov 3rd (4:00pm)** to submit your assessment solutions. Please ensure you answer all the questions outlined in the instructions portion of this document as well in your submission.

Marking: In this program core evaluation is marked with one of three possible marks: Incomplete, Pass, Pass Outstanding. For QAPs, though, where incomplete marks are more important for our own information as well as for the information of the student, we wanted to increase the resolution of our grading system. Therefore, QAPs are marked on a scale of 1-5. The details of this marking system are summarized in the table below.

Grade

Meaning

1

Incomplete. Student shows severe lack of understanding of the material – solution is heavily incomplete, non-functional, or completely off base of what the assignment was asking for.

2

Partially Complete. Students show some understanding of the material. Solution may be non-functional or partially functional, but the approach is correct, albeit with some major bugs or missing features.

3

Mostly Complete. Student demonstrates understanding of the major ideas of the assignment. Solution is mostly working, albeit with a few small bugs or significant edge cases which were not considered. Shows a good understanding of the correct approach, and is either nearly a feature-complete solution, or is a feature-complete solution with some bugs.

4

Complete (Equivalent to: Pass.) Student shows complete understanding of assigned work and implemented all necessary features. Any bugs that are present are insignificant (for example aesthetic bugs when testing the functionality of code) and do not impact the core functionality in a significant way. All necessary objectives for the assignment are completed, and the student has delivered something roughly equivalent to the canonical solution in terms of features and approach.

5

Complete with Distinction (Equivalent to: Pass Outstanding) The student demonstrates a clear mastery of the subject matter tested by the QAP. The solution goes above and beyond in some way, makes improvements on the canonical solution, or otherwise demonstrates the student’s mastery of the subject matter in some way. A solution in this category would consider all reasonable edge cases and implement more than the necessary functionality required by the assignment.

Instructions:

You are allowed to complete the assessment problems below in whatever way you can but please answer the following questions/points as part of your submission:

1. How many hours did it take you to complete this assessment? (Please keep try to keep track of how many hours you have spent working on each individual part of this assessment as best you can - an estimation is fine; we just want a rough idea.)
2. What online resources you have used? (My lectures, YouTube, Stack overflow etc.)
3. Did you need to ask any of your friends in solving the problems. (If yes, please mention name of the friend. They must be amongst your class fellows.)
4. Did you need to ask questions to any of your instructors? If so, how many questions did you ask (or how many help sessions did you require)?
5. Rate (subjectively) the difficulty of each question from your own perspective, and whether you feel confident that you can solve a similar but different problem requiring some of the same techniques in the future now that you’ve completed this one.

**Problem To Solve:**

For this QAP we will build on the concepts that we have covered in class around Object Mocking in Junit tests using the Mockito framework.

The basic setup for this assignment will be to code a set of Java objects to represent the following:

* A BloodDonationAppointmentManager object
  + Retrieves a list of appointments slots and donor info and match appointment slots and donor to create BloodDonationAppointment objects
  + Uses a **Database** object
    - Returns a list of appointment slots via a getAppointmentSlots method
      * Hardcode a list for a default impl, “Happy Path”
    - Returns a Donor via a getDonor method
      * Hardcode a list for a default impl, “Happy Path”
  + Given a Donors ID this class should get that Donor from the Database then get the list of available AppointmentSlots from the Database iterating through that list to apply the business rules and return a BloodDonationAppointment object if a match is found, or a InvalidDonationSchduelingException for any business rule that is triggered and does not create a new appointment.  This exception should have its getMessage overridden to explain the reason for it being thrown.
* A AppointmentSlot object that has the following properties:
  + Slot ID
  + Location
  + Date
  + Start Time
  + End Time
  + Blood Type
* A BloodDonationAppointment object that has the following properties:
  + Appointment ID
  + Appointment Date and Time
  + Appointment Duration
  + Location
  + Blood Type
  + First Time Donor?
  + Donor ID
* A BloodDonor Object that has the following properties:
  + Donor ID
  + First Name
  + Last Name
  + Date of Birth
  + Blood Type
  + Next Appointment
  + Last Donation Date
* Business Rules:
  + Donors must be 18 years old and less then 80 years old
  + Appointment Date must be at least 56 days from last appointment date
  + Appointment Date must be no more than 1 year from last appointment date
  + Appointment blood type must match Donor type
  + If a first-time donor the 56-day rule does not apply, and the appointment must indicate that this is a first-time donor
  + A donor can only have 1 appointment scheduled at a time

The objective here is to implement the basic logic and then build a set of JUnit test cases that mock the Database object.  The class under test will be the BloodDonationAppointmentManager object.  You’ll be expected to mock the returned appointments and donors to validate the following cases, each one in a different test case. One test case should validate the basic implemented “happy path”.  I.e. there exists and appointment slot that matches the Donors attributed and business rule mentioned above and a valid BloodDonationAppointment is returned.   These rest should be all driven by mock data.

* A valid donation date for a donor
* A valid appointment blood type and donor blood type
* A invalid birth date for a donor, both too young and too old
* A invalid blood type on the appointment and donor
* A invalid donation dates for a Donor, both too soon and too far away
* A attempt to schedule an appointment when Donor already has one

Please submit a document that has the 5 questions above answered and a link to your GitHub repo.

The project should also follow the same maven structure we have been using all term, same as the last QAP.