NATIONAL UNIVERSITY OF SINGAPORE

CS2103/T – SOFTWARE ENGINEERING

(Semester 1: AY2016/2017)

Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. Please write your Student Number only. Do not write your name.
2. This assessment paper contains **FIVE** questions and comprises **TEN** printed pages.
3. Answer **ALL** questions within the space in this booklet.
4. This is an OPEN BOOK assessment.

STUDENT NO: ______________________

This portion is for examiner's use only

Page	Marks	Remarks
2	/5	
3	/4	
4	/5	
5	/11	
6	/8	
7	/7	
Total	/40	

The **problem description** used for all questions in this assessment paper is given in the appendices (last two pages of this assessment paper).

Q1. Please read *Appendix A* of the problem description. When answering this question, you may consider the requirements specified in the problem description as well as additional features that you would like to propose for the product.

a) **[3 marks]** Assume that you have categorized TrackPR user stories into categories *must-have*, *nice-to-have*, and *unlikely-to-have*. Give one *must-have* and two *nice-to-have* user stories for TrackPR. All three user stories must help a tutor determine the level at which a particular student has been submitting CS2103/T tutorial work as GitHub PRs.

1. [must-have]

2. [nice-to-have]

3. [nice-to-have]

b) **[2 marks]** Give two *non-functional requirements* of TrackPR that is directly related to functional requirements specified in your answer to part (a) above.

c) **[2 marks]** Complete the following use case.

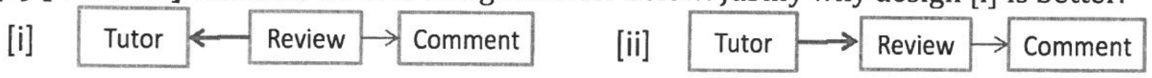
Use case: Filter PRs to be reviewed by a specific Tutor for a specific phase
Actor: Tutor
Preconditions: The app is running and ready for use

Q2
(a) [2 marks] Propose an architecture for TrackPR. Your architecture must be limited to the components given in *Appendix B*.

(b) [5 marks] Consider the Logic API given in *Appendix B* and the architecture you designed in the previous question.
Use a suitable UML diagram to describe how the architectural components interact during the app launch (i.e. from the time user double-clicks the executable to the time the app is ready for use).
To save time, you may omit the GitHubConnect component from your diagram.

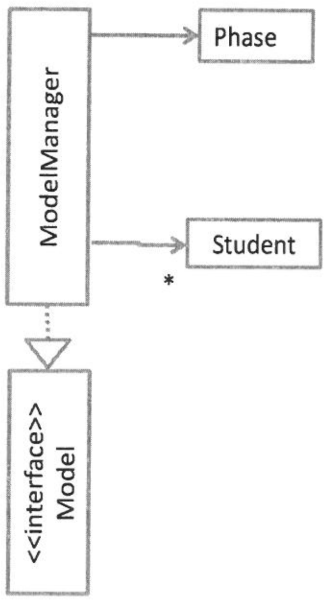
Q3. In this question, we attempt to design the internal structure of the Model component. Some suggested classes for the Model component are given in *Appendix C.*

(a) [2 marks] Consider the two design choices below. Justify why design [i] is better.



(b) [3 marks] Refer to the part (a) above. Suggest a way to improve design [i] further using one or more of the SOLID principles. Explain which principle(s) you used and why they make your design better.

(c) [6 marks] Complete the partial class diagrams given below to propose an object-oriented design for the `Model` component. Show <u>all</u> *navigabilities* and *multiplicities.* Try to minimize *associations.*
Show methods/attributes only if omitting them makes the diagram misleading.
Your design must include, but is not limited to, the classes listed in *Appendix C.* Include no more than 12 classes (i.e. you may omit less important classes if there are more than 12).



Q4.

(a) [6 marks] Review the addStudentToModule method below, which belongs to a class in the Model component. The student number is an integer 1 .. ClassSize generated internally by some other part of the code (e.g. if the class has 10 students, they are given numbers 1 to 10).
Suggest at least 5 (but no more than 10) improvements. One example is given.

```java
/**
  * Add student to the list of students enrolled in the module.
  * @param studentNumber should be a valid student number
  * @param gitHubId cannot be null
  * @throws InvalidUserException if the given user is not a valid GitHub user
  * @throws IncorrectDataException if the student number is not a valid student number
  */
void addStudentToModule(int studentNumber, String gitHubId)
                          throws InvalidUserException, IncorrectDataException {

    assert !isExistingStudent(studentNumber) : "student is already in the module";
    assert gitHubId != null;

    Log(Logger.WARN, "adding student to the module");

    if(studentNumber > classSize)
          throw new IncorrectDataException("invalid student number");

    if(!new GitHubConnect().isValidUser(gitHubId))
            throw new InvalidUserException();

    Student addStudent = new Student(studentNumber, gitHubId);

    classList.add(addStudent); //adds the student
    ui.updateStudentCount(classList.size());

}
```

Remove redundant comment

(b) [2 marks] It was found that the above code compiles without errors even if the throws InvalidUserException clause is removed from the method signature. How do you explain that?

Q5.

(a) [5 marks] Assume you have been asked to test the addStudentToModule method described in the previous question in black box fashion (you can view the header comment but not the implementation). Assume the number of students in the module is 100.

Give 4 more studentNumber values you would use as test inputs and explain why you chose each value.

#	studentNumber	explanation
1	50	A valid student number
2		
3		
4		
5		

Similarly, give 3 test inputs you would use for the gitHubId parameter.

#	gitHubId	explanation
1		
2		
3		

Using the values you chose above, give a test case that results in a student being added successfully.

#	studentNumber	gitHubId
1		

Assume the above test case is already in the test suite. Give two test cases that are NOT good test cases (i.e. they do not follow test case design heuristics). Explain the reason why each test case is not a good test case. Hint: to get full marks, the two cases should have two different reasons.

#	studentNumber	gitHubId	Reason
1			
2			

(b) [2 marks] How does the *Singleton* pattern affect the testability of a Java application?

[This page may be used if you need extra space for any of the answers]

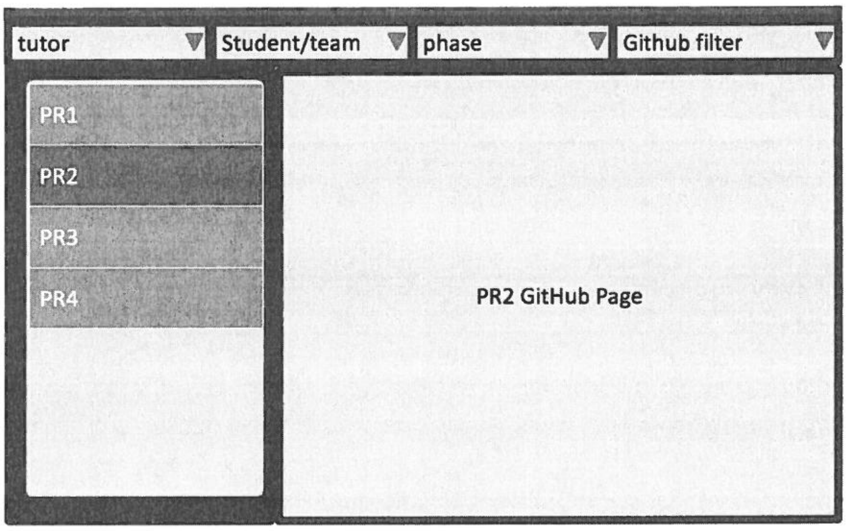
--- End of Paper---

Given below is the **problem description** used for all questions in this assessment paper. You may detach this page from the assessment paper. There is no need to submit this page.

Appendix A. TrackPR product description

Your team has been asked to build a software application called **TrackPR** which will be used by CS2103/T tutors for tracking GitHub Pull Requests done by CS2103/T students.

1. TrackPR is a desktop application with a GUI. It is not a CLI application.
2. TrackPR is used for tracking student work submitted as GitHub PRs. It should be able to load GitHub Web pages.
3. Given below is a sample prototype for the app GUI given as one possible design. Note that this is a sample only and you are free to design the GUI to be different.



4. For simplicity, assume the following:
    a. The module has 3 phases: A, B, C. In each phase, students will be in a different team of 3-5 members. Tutors are assigned one tutor per team per phase, and the tutor assignment can be different in each phase. This is similar to how things were done in the current semester.
    b. All CA work is submitted as PRs containing 1 or more team members. The app tracks PRs only. It does not track which tutorial or which activity a PR is for. The app ignores the concepts 'Tutorial', 'Activity', and 'Project' but it tracks which PR belongs to which phase.
5. TrackPR downloads commit history from the GitHub server. All data for the entire cohort is downloaded at once at the app launch. The data is then analyzed and converted into an object structure that captures the information about work students did in PRs.
6. Some example capabilities expected of Track CA are,
    a. Filter PRs by tutor, student, team, phase, etc.
    b. Filter PRs using Github filters e.g. is:open label:reviewed
    c. Show student work as various visualization/statistics.
    E.g. a graphs showing number of commits by a team over time

7. TrackPR does not store the downloaded data in the user's Computer.
8. Some configuration data (e.g. login credentials of the user for GitHub, repos used by students, mapping from matric number to github user ID, etc.) are to be put in a file named config.txt. The user is expected to create that file manually, following a specific format. TrackPR reads that file at launch.

Appendix B. Suggested components for the architecture

- **GUI**: The Graphical User Interface. Uses JavaFX. Some UI elements will be bound to the data in the Model component so that the UI can update itself as the Model changes.
- **Logic**: The main logic of the application. The API is as follows:
    - **Logic()**
      Creates an empty Logic component. Not usable until the init() method is called.
    - **init():Model**
      Initializes the other components needed by Logic and returns the initialized Model component.
    - Other methods omitted

- **Storage**: A temporary component used only during app launch. It is used to read configuration data from the hard disk.
- **GitHubConnect**: Used to download data from GitHub. This component will be taken from an existing app and reused (as is) in TrackPR.
- **Model**: Created by the Logic component. Holds the object structure containing the data required to respond to user requests.
- **Utils**: Contains the utility classes used by other components e.g. Logging classes

Appendix C. Suggested classes for the Data component

- Phase: A phase in the CS2103 module structure. E.g. Phase A, Phase B, Phase C
- Tutor: A tutor in the teaching team.
- Commit: A commit made by a student.
- PR: Contains commits by one or more students.
- Comment: A comment given by a tutor for a specific PR (assume comments are given for a PR, not for a Commit).
- Review: contains Comments given by a tutor for a specific PR.

As mentioned earlier, TrackPR does not recognize the concepts of Tutorial, Activity, and Project. It does not track the creator of a PR as any team member can create a PR on behalf of a team. It only tracks the authors of commits in a PR.

-- End of problem description—