NATIONAL UNIVERSITY OF SINGAPORE

CS2103/T – SOFTWARE ENGINEERING

(Semester 1: AY2018/19)

Part 2 (mock)

Time Allowed : 1 Hour

INSTRUCTIONS TO STUDENTS

- Please write your Student Number only. Do not write your name.
- This assessment paper contains **FOUR** questions and comprises **FIVE** printed pages.
- You are required to answer **ALL** questions.
- This is an **OPEN BOOK** assessment.
- You may **use pencils** to write answers.

STUDENT NO: _____

This portion is for examiner's use only

Question	Marks	Remarks
Q1	/5	
Q2	/5	
Q3	/5	
Q4	/5	
Total	/20	

Q1 [5 marks] Illustrate the structure of the following *problem domain* (related to a role playing game) using a suitable UML diagram.

> A player can build structures, fight with other players, and trade (i.e. buy/sell) assets with other players. Some assets are used for building, some are used for fighting, and some are used as currency for trading with other players. Two types of assets that can be used as currency are gold bars and salt bags. All assets have an associated value. In addition, salt bags have expiry dates. A trade between two players involves a buyer giving up one or more assets to receive one or more assets from the seller. Buyer is the player who initiates the trade.

Q2 [5 marks] Illustrate the interactions caused by calling the `Car#start()` given below using a suitable UML diagram.

```
class Car{
    Engine engine;
    boolean isReady;

    void start(){
        if(isReady){
            engine.start();
        } else {
            engine.warm();
            showDelay();
        }

    }

    private void showDelay() {
        //...
    }
}
```

```
class Engine{
    Valve valve;

    void start(){
        valve.open();
    }

    public void warm() {
        //...
    }
}

class Valve{
    void open(){
        //...
    }
}
```

Q3 [5 marks] Design an efficient and effective set of test cases for the method given below. Show the intermediate steps to your test case design. Give at least 7 but no more than 10 test cases.

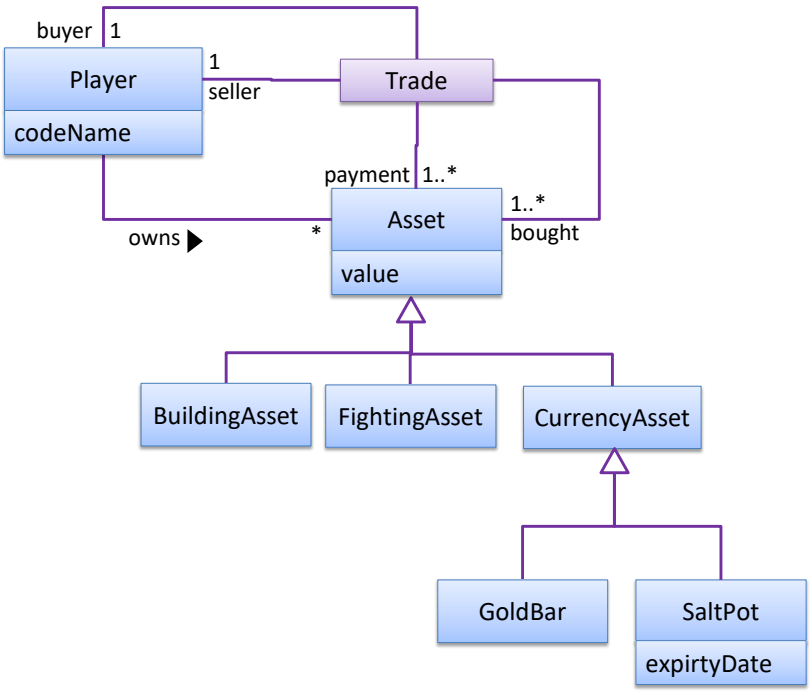
```
/**
 * Returns the class size of the specified module.
 * @param moduleCode should be in the range 1000..6999
 * @throws Exception if the moduleCode is not in range or the user is
 * not logged in or if the user is not an instructor
 */
int getClassSize(int moduleCode)throws Exception{
    //...

}
```

Q4 [5 marks] Suggest at least 5 code quality improvements to the code below. One example given.

```
/**
 * 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 gitHubId 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

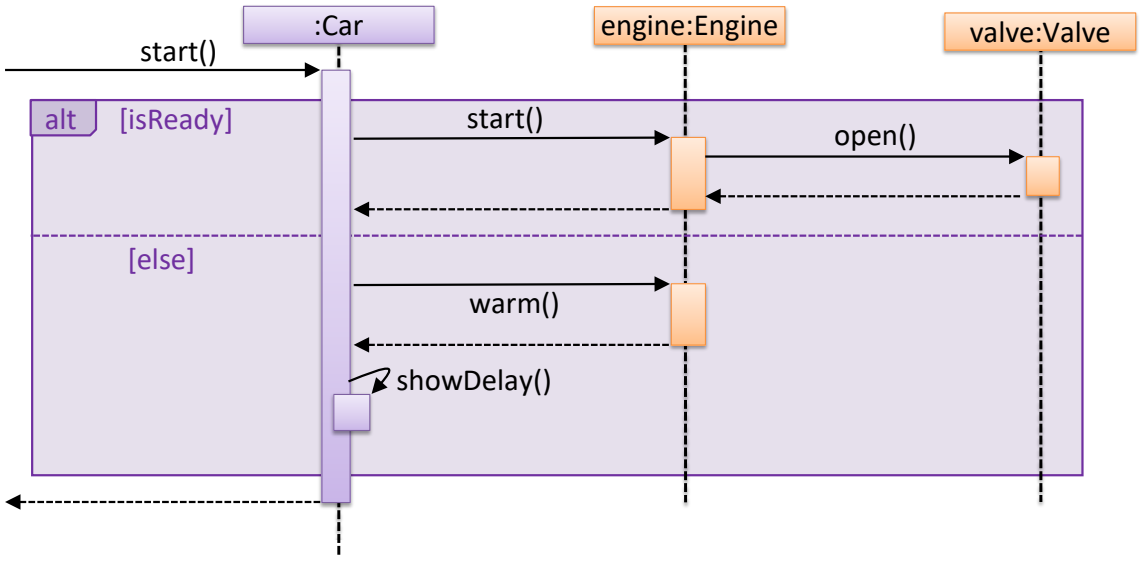
Q1.



Examiner notes:

- In questions like this one, the exact UML diagram expected may not be specified; it is your job to pick the most suitable diagram type.
- Some information in the description may not be relevant to the diagram.
- The diagram above is an OODM. That is why methods and navigability are not shown.
- There may be multiple correct answers to a question like this one.

Q2.



Q3.

	Equivalence classes	Values to test
moduleCode	=< 999	999
	1000…6999	1000, 6999
	>= 7000	7000
Is instructor?	Yes	Yes
	No	No
Is logged in?	Yes	Yes
	No	No

Test cases:

	moduleCode	Is instructor?	Is logged in?	Expected
1	1000	Yes	Yes	Success
3	6999	Yes	Yes	Success
3	999	Yes	Yes	Exception
4	7000	Yes	Yes	Exception
5	Any value in 1000…6999	No	Yes	Exception
6	Any value in 1000…6999	Yes	No	Exception
7	A non-boundary value in 1000…6999	Yes	Yes	Success

Examiner notes:

- For questions like these you are expected to use equivalence classes, boundary values, and heuristics for combining multiple test inputs.

- As we have room for 7 test cases, we can add a test case to test a non-boundary valid value (see test case 7).

Q4.

i. [line 2] Add student → Adds student (to follow the coding standard for method header comments)
ii. [2] Give more accurate details of what the method does, for example, the fact that it creates a Student object.
iii. [3] Add a blank line after the overview
iv. [3] Give more details on what 'valid' means e.g. student cannot have been added to the module already
v. [6] Give more details on what 'valid' means
vi. [11] Adding a duplicate student could be a user error, which should be handled by an exception rather than an assertion.
vii. [14] Log message can contain more details (e.g. student number).
viii. [14] Rephrase to '*Attempting* to add student …'
ix. [14] Log level should be lower than WARN
x. [16][19] Braces missing
xi. [16] Validity checking can be extracted into separate methods to increase SLAP
xii. [16] Can be more defensive: should check for numbers smaller than 1
xiii. [19] Validity checking can be extracted into separate methods to increase SLAP
xiv. [20] Supply a descriptive error message for the exception
xv. [20] Inconsistent indentation
xvi. [24] Variable named as a verb. newStudent is a better name.
xvii. [22] Perhaps creating a Student object should not be the job of this method?
xviii. [various places] More logging can be added