NATIONAL UNIVERSITY OF SINGAPORE

CS2103/T – SOFTWARE ENGINEERING

(Semester 1: AY2019/20)

Part 2

Time Allowed : 1 Hour

INSTRUCTIONS TO STUDENTS

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

STUDENT NO: A_____

This portion is for examiner's use only

Question	Marks	Remarks
Q1	/12	
Q2	/10	
Q3	/10	
Q4	/8	
Total	/40	

Q1 [8+2+2=12 marks]

(a) [8 marks] Illustrate the class structure of the following code using a UML diagram. Also incorporate the following information into the diagram:

- An `Activity` object can consist of other `Activity` objects i.e., sub-activities.
- A `Watcher` object may not be associated with more than 5 `Activity` objects.
- `UiWidget` class inherits the `ProgressWatcher`.

Show all known information. Show associations as lines, not as attributes. Use the given layout.

```
class Activity{
    private Watcher[] watchers = new Watcher[ProgressWatcher.MAX];
    //...
    void watch(Watcher w){
        //add w to watchers
    }

    Activity getInstance(){
        //...
    }
}
```

```
interface Watcher{
    void update(int value);
}
```

```
abstract class ProgressWatcher implements Watcher{
    static int MAX = 10;
}
```

Activity

Watcher

ProgressWatcher

UiWidget

For examiner's use only		
	1	
	2	
	3	
	4	
	5	
	6	
	7	
	8	
	9	
	10	

(b) [2 marks] Which *design pattern* (out of the ones covered in the module) is being used in the above design?

(c) [2 marks] Draw a UML diagram to illustrate an object structure that has two `Activity` objects `a1` and `a2`, both being watched by a `UiWidget` object `u`. Furthermore, `a2` is a sub-activity of `a1`. Show all known information. Show associations as lines, not as attributes. Use the given layout.

a1

u

a2

Q2 [6+4=10 marks] Consider the code below:

```
class Client {
    Client(Server s, int port) {
        init(s);      // parallel
        s.set(port);  // parallel
    }

    void init(Server s) {
        s.add(this);
    }
    //...
```

```
class Main {
    //...
    void launch() {
        while (isNeeded) {
            new Client(s, 0).run();
        }
    }
//...
```

(a) [6 marks] Use a UML sequence diagram to illustrate the interactions caused by the statement `Client c = new Client(s, 100)`. Note that the two statements in the constructor execute in parallel to each other (the extra code required for parallelizing them is not shown for simplicity).

For examiner's use only		
	1	
	2	
	3	
	4	
	5	
	6	
	7	

(b) [4 marks] Draw the execution flow caused by calling the `launch()` method as an activity diagram. Show each method call as an *action* in the activity diagram. Use the layout given.

For examiner's use only		
	1	
	2	
	3	
	4	
	5	

Client()

init() add() set()

run()

Q3 [2+2+3+3 = 10 marks]

(a) The Java method `isSameLength(String[] values)` returns `true` only if the `String` in `values[0]` is the same length as the `String` in `values[1]`.
e.g., `values=["aa", "bb"]` returns `true` but `values=["a", "bbb"]` returns `false`.

(a1) [2 marks] Given on the right are two test cases that expect **`true`**. Justify why the 2nd test case is redundant, citing the relevant test case design heuristic.

	values
1	["aa", "bb"]
2	["aaa", "bbb"]

(a2) [2 marks] Give a good (i.e., *efficient* and *effective*) test case to replace the redundant test case in part (a1). The expected output for the new test case should be **`true`**.

(a3) [3 marks] Assuming you can have no more than four additional test cases, give four good test cases for which the expected output is **NOT `true`** (i.e., `false` or error condition)

(b) [3 marks] Consider the method `proceed(Status entryStatus, Status exitStatus)` where `Status` is an enumeration with values `A, B, X, Y`.
`A` and `B` are considered *valid* statuses while `X` and `Y` are considered *invalid* statuses.
The method succeeds if both parameters are valid statuses. Else, it aborts the execution.
It has been decided that only the four values `A, B, X, Y` will be included in test cases. One test case for testing this method is given below. Give up to five more test cases (divided into the two categories) that you will use to test this method in an efficient and effective manner.

Test cases that **succeed**:

	entryStatus	exitStatus
1	A	B
2		
3		
4		
5		

Test cases that **abort** the method:

	entryStatus	exitStatus
1		
2		
3		
4		
5		

For examiner's use only

	1	
	2	
	3	
	4	

Q4 [2+6=8 marks] Consider the method below, taken from a `Parser` class of a CLI application:

```
1  // Returns the first word of the given input.
2  // @param input the string to be parsed.
3  // @return the first word of the input.
4  // @throws InvalidInputException if the string is invalid.
5  public String getCommandWord(String input)
6      throws InvalidInputException {

7      assert input != null : "null not expected at this point";

8      log(WARNING, "Getting command word from input " + input);

9      if (input.isEmpty()) {
10         throw new InvalidInputException();
11     }

12     String[] word = input.split(" ");
13     String CommandWord = word[0];
14     History.add(CommandWord);

15     log(WARNING, "Command word is " + CommandWord);
16     return CommandWord;
17 }
```

(a) [2 marks] Suggest how to fix two coding standard violations (w.r.t. the coding standard used in the module). Cite the relevant line number(s). One example is given.

Suggestion
Lines 1-6: Should be formatted as a Javadoc comment.

For examiner's use only		
	1	
	2	
	3	

(b) [6 marks] Select <u>three areas</u> (from the ones given in the table below) in which the `getCommandWord` method can be improved the most. For each of them, give one way to improve the code in that area. Cite the relevant line numbers in your suggestion.

Area	**Suggestion**
Assertions	
Cohesion	
Comments	
Exceptions	
Logging	
Other	

For examiner's use only		
	1	
	2	
	3	
	4	
	5	
	6	
	7	

BLANK SPACE

If you have **doubts/queries or any other information** that you communicate with the examiner (e.g., potential errors in questions), please write them in this box, together with the question number.

END OF PAPER