

NATIONAL UNIVERSITY OF SINGAPORE

CS2103/T – SOFTWARE ENGINEERING

(Semester 2: AY2018/19)

Part 1 (mock)

Time Allowed : 1 Hour

INSTRUCTIONS TO STUDENTS

1. Please write your **Student Number only**. Do not write your name.
2. This assessment paper contains **80** questions and comprises **TEN** printed pages.
3. Students are required to answer **ALL** questions.
4. This is an **OPEN BOOK** assessment.
5. All questions **carry equal marks**. Total marks: **20**
6. **Questions in this paper are confidential**. You are not allowed to reveal them to anyone. All pages of the assessment paper are to be returned at the end of the exam.
7. The meaning of options are
 - A. Agree. If the question has multiple statements, agree with all of them.
 - B. Disagree. If the question has multiple statements, disagree with at least one of them.
8. **Notation:** [x | y | z] means ‘x and z, but not y’.
e.g. SE is [boring | useful | fun] means SE is not boring AND SE is useful AND SE is fun.

STUDENT NO: _____

Section A

A1. The answer to this question is **A** (this question is used to identify which bubble sheet is for which section of the assessment paper)

A2. A use case can have *preconditions*.

A3. Committing saves [all | staged] changes to tracked files in the revision control history.

GIT

A4. Testing is a [verification | validation] type QA activity.

A5. The following command feeds the text in `input.txt` into `AddressBook` program and saves the output in the `output.txt` file.

```
java input.txt > AddressBook > output.txt
```

A6. Refactoring can improve the performance of the refactored code.

A7. A coding standard [can contain rules that can be objectively enforced | may contain rules that are subjective].

Suppose you are reviewing the code below. Assume the coding standard is similar to the one you used in the CS2103/T. **JAVA**

```
/**
 * Sets the address to the given value
 */
public void address(String address){
    //set address to a default value if null
    if(address == null){
        this.address = "ABC avenue";
        return;
    }

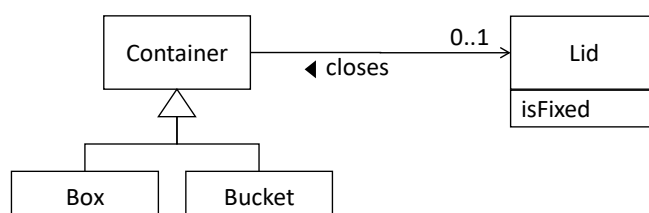
    //set address to given value
    this.address = address;
}
```

A8. `setAddress` is a better name for this method.

A9. The commenting intensity *inside* the method (i.e. excluding the header comment) is reasonable. i.e., not too much and not too little

A10. The header comment is missing some vital information about the method's behavior.

A11. The code has an issue related to *magic numbers*.



Consider the diagram above. **UML**

A12. Container is an abstract class.

A13. A Lid must be associated to a Container but it is optional for a Container to have a Lid.

A14. There is a [dependency | coupling | association] between Container and Lid.

A15. Assuming the Box() constructor exists, the following line compiles. **JAVA**

```
Container c = new Box();
```

A16. The following object diagram is compliant with the given diagram.



A17. The code below is compliant with the given class diagram. **JAVA**

```

class Lid{
    Container c;
    boolean isFixed;
    //..
}
  
```

A18. A use case describes the interaction between major components of the system for a specific usage scenario.

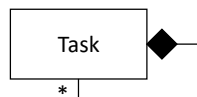
A19. The code below compiles. **JAVA**

```

void bar(boolean isValid) throws Exception{
    if(!isValid){
        throw new String("Invalid data");
    }
}
  
```

A20. JUnit can be categorized as a testing [approach | framework | platform].

A21. This is a correct usage of *composition* to represent the fact that a Task may consist of other Tasks. **UML**

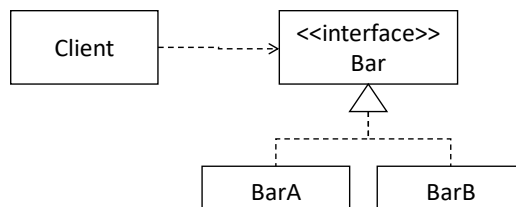


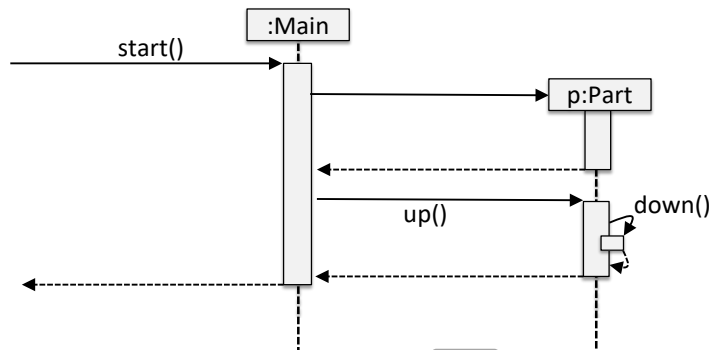
A22. The module project uses a *depth-first* iterative model. **SDLC**

A23. Method overloading can happen within a single class. **JAVA**

A24. *Polymorphism* can be achieved using *interfaces*, as shown in the example below. **OOP**

UML **JAVA**





Consider the sequence diagram above: **UML**

A25. The diagram has one or more notation errors.

A26. The code below is compliant with the diagram. **JAVA**

```

class Main{
    void start(){
        // some code here
        Part p = new Part();
        // some more code here
    }
}
  
```

A27. The code below is compliant with the diagram. **JAVA**

```

class Main{
    void start(){
        // some code here
        p.up();
        p.down();
        // some more code here
    }
}
  
```

A28. An *architecture diagram* can use the following symbol to represent a component.



A29. Gradle is a *continuous integration* tool.

A30. The *n-Tier architectural style* is also known as the *layered architectural style*.

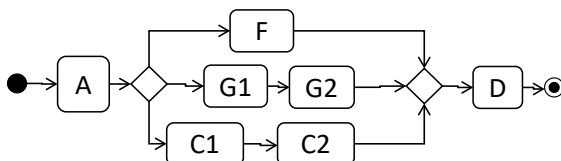
A31. The *MVC* pattern is an example of the *Separation of Concerns* principle being applied.

A32. High-level user stories that cover bigger functionalities are called [*epics* | *legends*].

A33. AddressBook-level4 uses *continuous deployment*.


A34. *System testing* covers mostly negative test cases while *acceptance testing* covers mostly positive test cases.

A35. The following *activity diagram* supports the action sequence A G1 G2 D **UML**



A36. *Defensive programming* can result in slower code.

A37. Path coverage is harder to achieve than statement coverage.

-  **A38.** Grey-box testing uses [some | full | no] knowledge of the SUT specification and [some | full | no] knowledge of its implementation.
- A39.** When developing a software to compete with Facebook, an iterative approach is more suitable than a sequential approach.
- A40.** *Equivalence partitions* cannot give a Neumann-complete test suite.
- A41.** Code coverage as reported by IntelliJ IDEA is an example of a [dynamic | static] analysis.
- A42.** Scrum is an [agile | iterative | sequential] process.
- A43.** An OO domain model can be helpful in building an OO solution.
- A44.** The earlier a bug is found, the easier to have it fixed.
- A45.** *Single Responsibility Principle* is highly related to the concept of *cohesion*.
- A46.** *Architectural styles* provide a high-level vocabulary to communicate about architecture.
- A47.** *Early and frequent* integration is recommended over *late and one-time* integration.
- A48.** UML Sequence Diagrams can be used when designing APIs of components in an architecture design.
- A49.** When writing developer documentation, it is more important to be *comprehensive* than *comprehensible*.
- A50.** Design patterns are documented in a specific format that requires the use of UML models.
-

Section B

B1. The answer to this question is B

B2. More questions here ...

B3. ...

...

B50. ...

--- End of part assessment paper (Part 1) ---

Answers

Answers are given below. If you have any doubts about any of the question, please post in our forum to discuss

- | | |
|-------|-------|
| 1. A | 26. A |
| 2. A | 27. B |
| 3. A | 28. A |
| 4. B | 29. B |
| 5. B | 30. A |
| 6. A | 31. A |
| 7. B | 32. A |
| 8. A | 33. B |
| 9. B | 34. B |
| 10. A | 35. A |
| 11. A | 36. A |
| 12. B | 37. A |
| 13. B | 38. B |
| 14. B | 39. A |
| 15. A | 40. - |
| 16. A | 41. B |
| 17. B | 42. A |
| 18. B | 43. A |
| 19. B | 44. A |
| 20. B | 45. A |
| 21. A | 46. A |
| 22. B | 47. A |
| 23. A | 48. A |
| 24. A | 49. B |
| 25. B | 50. B |

This is a nonsense question, put there to let you practice how to respond to an incorrect question. No such deliberately incorrect questions in the actual exam paper though