NATIONAL UNIVERSITY OF SINGAPORE

CS2113/T – SOFTWARE ENGINEERING AND OBJECT ORIENTED PROGRAMMING

(Semester 1: AY2019/20)

Part 1 **(Mock)**

Time Allowed: 30 Minutes

INSTRUCTIONS TO STUDENTS

1. Please write your Student Number only. Do not write your name.
2. This assessment paper contains **50** questions and comprises **6** printed pages.
3. Answer **ALL** questions by shading the letter corresponding to the answer on the OCR form provided.
4. The meaning of options are
   A. True
   B. False
5. All questions carry equal marks. Total marks: **25**
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. This is an OPEN BOOK assessment.
8. **Clarifications about questions are not allowed for this assessment paper.**

STUDENT NO: _____

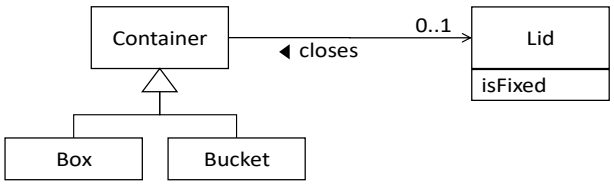
1. A use case can have preconditions.
2. Testing is a verification type QA activity.
3. The following command feeds the text in `input.txt` into `AddressBook` program and saves the output in the `output.txt` file.

   ```
   java input.txt > AddressBook > output.txt
   ```
4. Refactoring can improve performance of the refactored code.
5. A coding standard can contain rules that can be objectively enforced.

Suppose you are reviewing the code below. Assume the coding standard is similar to the one you used in the CS2113/T.

```
/**
 * 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;
}
```

6. `setAddress` is a better name for this method.
7. The commenting intensity inside this method (i.e. excluding the header comment) is reasonable. i.e., not too much and not too little
8. The header comment is missing some vital information about the method's behavior.
9. The code has an issue related to *magic numbers*.

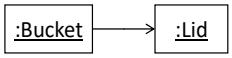


Consider the diagram above.

10. Container is an abstract class
11. A `Lid` must be associated to a `Container` but it is optional for a `Container` to have a `Lid`.
12. There is a dependency and coupling but not association between `Container` and `Lid`.
13. Assuming the `Box()` constructor exists, the following line compiles.

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

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

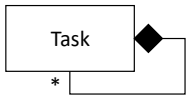


---

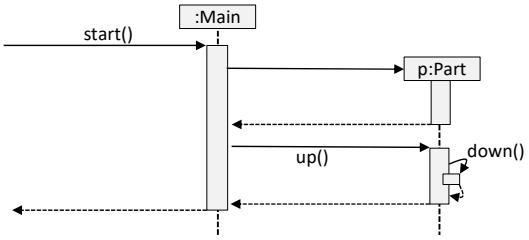
15. The code below compiles.

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

16. Junit can be categorized as a testing platform.
17. This is a correct usage of *composition* to represent the fact that a `Task` may consist of smaller `Tasks`.



18. The module project uses a *depth-first* iterative model.
19. Method *overloading* can happen within a single class.



Consider the sequence diagram above:

20. The code below is compliant with the diagram

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

21. The diagram has one or more notation errors.

---

22. Gradle is a continuous integration tool.
23. *n-Tier architectural style* is another name for the *layered architectural style.*
24. The *MVC* pattern is an example of the *Separation of Concerns* principle being applied.

25. Continuous *integration* and continuous *deployment* are the same.
26. *System testing* covers negative test cases while *acceptance testing* covers mostly positive ones.
27. *Defensive programming* can result in slower code.
28. When developing a software to compete with Facebook, an iterative approach is more suitable than a sequential approach.
29. Code coverage as reported by Intellij IDEA is an example of static analysis.
30. Dev guide: it is more important to be *comprehensive* than *comprehensible.*

~~ End of questions (Part 1) ~~

1. T
2. T
3. F
4. T
5. T
6. T
7. F
8. T (missing info on defaults)
9. T
10. F
11. F (no info on multiplicities)
12. F
13. T
14. T
15. F
16. F (framework)
17. T
18. F
19. T
20. T
21. F
22. F
23. T
24. T
25. F
26. F (system testing covers both)
27. T
28. T
29. F
30. F