NATIONAL UNIVERSITY OF SINGAPORE

CS2103/T – SOFTWARE ENGINEERING

(Semester 1: AY2018/19)

Part 2

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: _____

Note that the question text has been tweaked slightly to remove some ambiguous phrasings and typos (based on queries during the exam)

This portion is for examiner's use only

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

Q1 [4+1=5 marks]

(a) [4 marks] Illustrate the class structure of the following code using a suitable UML diagram. In addition to the classes in the code, include the following classes:

- `ActionX, ActionY`: Inherits from the `Action` class.
- `ActionFactory`: creates `ActionX, ActionY` objects and adds them to the `History` object.

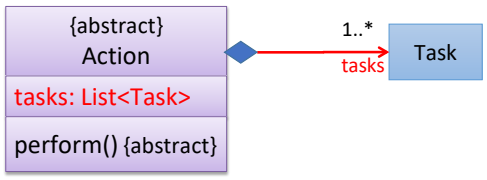
Show all associations as lines. Show attributes, methods, navigabilities, dependencies, and known multiplicities. Other optional notations can be used only when they add value.

```
/** A repeatable action to be performed by the app */
abstract class Action implements Repeatable{
    List<Task> tasks; //1 or more tasks that the action consists of
    abstract void perform();
}

interface Repeatable{
    void repeat();
}

class History{
    List<Repeatable> history;
    void add(Repeatable r){
        //..
    }
}
```

```
class Task{
    Repeatable owner; // the owner of this task
    Task previous; // the previous task, if any

    Task(Repeatable owner, Task previous){
        //..
    }
}
```

Common mistakes:

- Mistake: showing dependencies as associations e.g., `ActionFactory` creates `ActionX, ActionY` is a dependency rather than an association.
- Mistake: showing an association as both an attribute and a line (the diagram below commits that mistake). Both represent the same thing; only one of them should be used.



- `Task previous; // the previous task, if any`
  - is an association, not a dependency. Reason: there is a variable holding a permanent link to the Task object
  - has a multiplicity of 0..1, not 1. Reason: a variable can be null. Besides, the comment says 'if any'.
- `List<Task> tasks; //1 or more tasks that the action consists of`
  - is a 1..* multiplicity, not *.
  - 'consists of' was meant to hint at composition. I did not penalize for missing this though. On a related note, you should not have used composition in other places of the diagram as there were no other indication of whole-part relationships.
- Mistake: Showing `Action implements Repeatable` relationship as a solid line. It should be a dashed line.

(b) [1 mark] Which *design pattern* (out of the ones covered in the module) is being used in the above design (a close match is acceptable – does not have to be an exact match)?

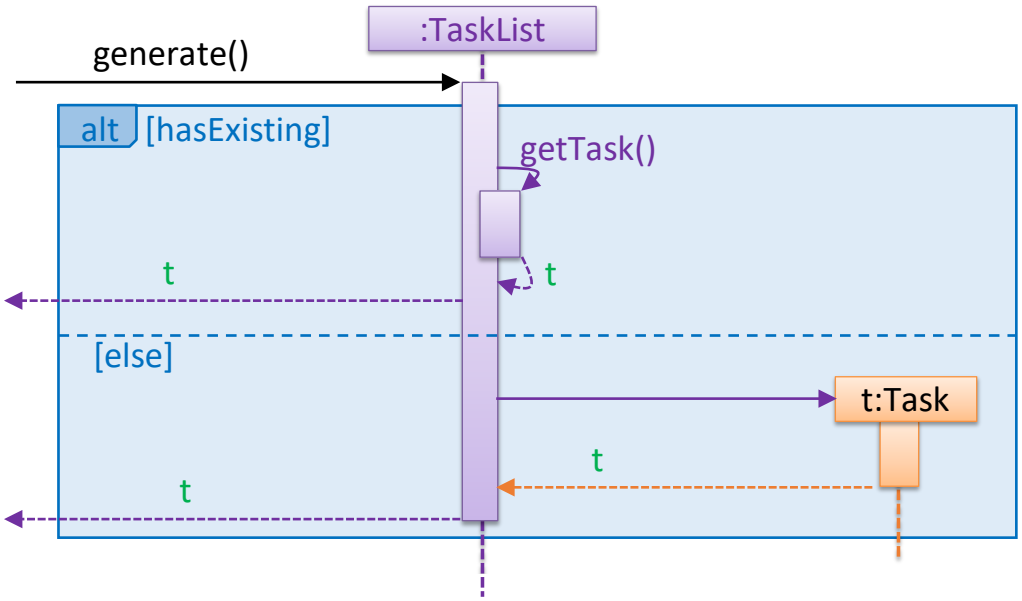
Answer: Command pattern.

Q2 [3+2=5 marks]

(a) [3 marks] Use a UML sequence diagram to illustrate the interactions caused by calling the `generate()` method given below. Ignore the `validate()` method.

```
class TaskList{
    Task generate(boolean hasExisting){
        if (hasExisting){
            return getTask();
        } else {
            return new Task();
        }
    }

    void validate(){
        check();
        confirm();
    }

    //...
}
```

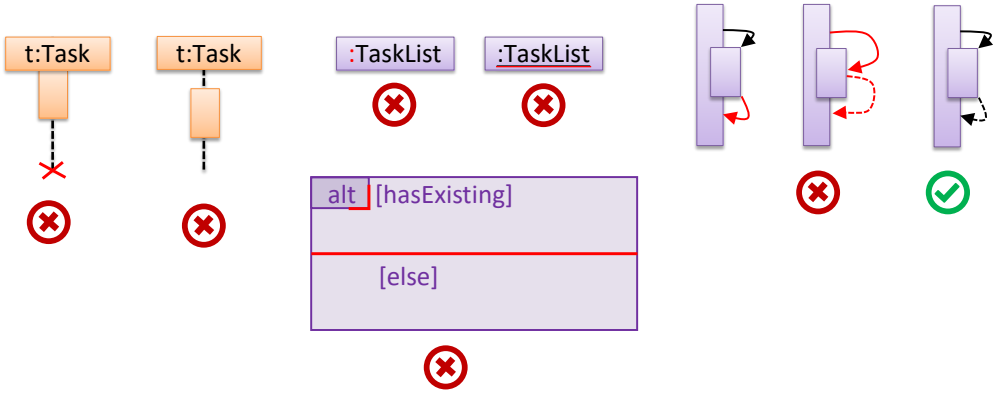
Suggested answer:



Common mistakes

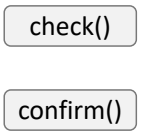
- Missing ':' in class name, underlining the class name
- Notation mistakes in the alt frame e.g., using solid line to divide the frame, not snipping the corner of alt label

- Showing an 'X' in the 'Task' lifeline. Reason: The created object is returned to the caller, which means the object lives on.



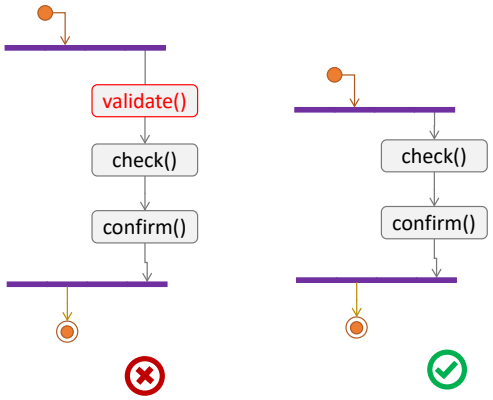
- Not showing `getTask()` as a self-call. Although the code of the `getTask()` method was not given in the question, it can only be a method in the same class.
- Showing return arrows as solid arrows.
- Not indicating that the created Task object is being returned (the suggested solution above uses `t` to indicate the returned object.

(b) [2 marks] Assume the `validate()` method and the `generate()` method of the a `TaskList` object -- code given in part (a) above -- are run in parallel using some mechanism. Complete the partial UML activity diagram below to Illustrate that scenario.



Common mistakes:

- Mistake: Showing validate() in the diagram. Reason: The activities check() and confirm() together represent the validate() method. If you show validate() in the diagram, check() and confirm() methods will be called twice each!



- If your diagram did not show the if-else branch that is given inside the generate() method, I had no way to judge whether you knew how to draw alternate paths. Although the question did not specify that you need to draw the internal steps of generate() method, it is implied because the partial diagram is already showing the internal steps of the validate() method.

Q3 [2+3 = 5 marks]

(a) [2 marks] The `isDigit(String s)` method returns `true` only if `s` is a single digit integer e.g., "4" or "9" (i.e., it should be a string of length 1). Suggest 5 more test inputs for this method where the expected result is **`false`**, in addition to the example test case given. Assume you are doing exploratory testing and give test inputs you think that are more likely to break the SUT i.e., this question evaluates your creativity as a tester, rather than your ability to apply theory.

	Input	Explanation
1	"10"	More than one digit
2		
3		
4		
5		
6		

Comments: The question specifies two conditions that should both hold for the method to return true: s should be a string of length 1 and should represent a single digit integer. You should look for test values representing conditions handling of which may have been overlooked by the developer. Some examples:

- Null
- An empty string of length 1: " "
- A string that represents a single digit integer but is not length 1: e.g., "05"
- A string of length 1 that is not an integer e.g., "x"

- A string that meets the criteria *after* trimming extra spaces e.g., “ 3 “

Values such as “1 + 3” is unlikely to discover bugs as it is unlikely the code evaluates the string as an equation.

It is also not possible to test a non-String (e.g., an int) as s because that would cause a compile error.

(b) [3 marks] The `isNormal(Status s, int t)` method returns true if both the following conditions are satisfied:

- s is either OK or FINE (Status is an enumeration with values OK, FINE, BAD)
- t is in the range -5 to 99, including both values

Three test case for testing this method are given below. Give no more than six more test cases (it total) that you will use to test this method in an efficient and effective manner.

Test cases that return **`true`**:

	s	t
1	OK	0
2		
3		
4		
5		

Test cases that return **`false`**:

	s	t
1	FINE	150
2	OK	-200
3		
4		
5		
6		
7		

Common mistakes:

- Giving more than the requested number of test cases
- Not considering null for s
- Providing incorrect type data that would result in compile errors
- Use -4 as a boundary value (-6 is the correct boundary value)
- Not testing boundary values -6, 100

Q4 [5 marks] For the following code, suggest how to fix at least 3 coding standard violations (w.r.t. the coding standard used in the module) and at least 3 other ways to improve the code quality. Use arrows and text, similar to the example given. It is not needed to indicate which suggestion is which type.

```
// Converts the given code to a properly formatted info message.
// @param code a string representing the status code.
// @return null if code is an empty string.
// @throws InvalidCodeException if code is null.
public String getinfo(String code) throws InvalidCodeException {

    log("In the getinfo method");

    if (code.isEmpty()) { return null; }

    if (code != null) {

        switch(code) {
            case "0" :
                code = CODE_ZERO_RECEIVED;
                break;
            case "1" :
                code = CODE_ONE_RECEIVED;
                break;
            default:
                code = CODE_TWO_RECEIVED;
                break;
        }

        code = INFO_PREFIX + code + INFO_SUFFIX;
        return code;

    } else {
    // invalid code!
        throw new InvalidCodeException();
    }
}
```

Should be indented at the same level as the throws clause

--- End of assessment paper (Part 2) ---

Comments:

- One doubtful suggestion was to provide a logging level for the log message. Reason: it is possible that the log method used by the code has a default log level specified inside it. A more useful suggestion is to include the parameter 'code' in the log message to help with troubleshooting later.
- The answer should indicate how to fix the problems, not simply point out the problem.
- Some issues missed by many students:
    - Happy path is indented. Can be fixed using guard clauses.
    - The header comment is not in JavaDoc format ('//…' instead of '/**…')
    - Case blocks should not be indented (as per our coding standard)
    - If condition does not follow Egyptian style braces
    - Method name is not in camelCase format