The University of Melbourne
School of Computing and Information Systems

Practice Exam

Semester 1, 2020

SWEN20003 Object Oriented Software Development

Exam Duration: 2 hours

Total marks for this paper: 100

This paper has 7 pages

Continued overleaf ...

Instructions to students:

- The exam has 4 questions across 3 sections, and all questions must be attempted.
- This is an online, open-book exam and you will have access to your subject material and Internet resources.
- You are required to provide your own answers to questions; copying directly from subject material or Internet sources is not acceptable and will be considered plagiarism.
- The mark allocation for each question is listed along with the question. Please use the marks as a guide to the detail required in your answers while keeping your answers concise and relevant. Point form is acceptable in answering descriptive questions. Any unreadable answers will be considered wrong.
- Worded questions must all be answered in English, and code questions must all be answered in Java.
- Please follow the instructions below to complete the exam:
  - Clone the solution template from the repository `<username>-practice-exam` in `gitlab.eng.unimelb.edu.au` using the appropriate: `git clone` command.
  - In this repository you will find a single file named `<username>-practice-exam.txt`. You are required to provide the answers to questions in the appropriate section of this file, through typing them in; all answers must be in this file and no other files should be added to the repository. You are not permitted to upload hand-written scanned files, and such solutions will receive zero marks.
  - Upon attempting each sub-question commit the changes to your files (answers) using:
    `git add .` and
    `git commit -m '[commit message]'`
    performed from the cloned folder in the above step.
    e.g. Once you complete Question 1(a) you can execute the following commands to commit the changes:
    ```
    git add .
    git commit -m 'completed question 1(a)'
    ```
  - You may choose to do `git push` as often as you would like as a way of backing up, but are not required to push until you complete the exam.
  - At the end of the exam, you are also required to submit the solution file, `<username>-practice-exam.txt`, as a Canvas Turnitin Assignment submission.
    Note: It is important to carefully follow the instructions above. Abnormal commits will receive scrutiny because it could indicate possible plagiarism.
- [**This instruction is only applicable to the final exam and not this practice exam**] Your lecturers will be available online via Zoom, during the first 15 minutes of the exam (Zoom link will be made available via Canvas). During this time you may ask clarification questions. You will be allowed to start answering questions during this time if you choose to do so (this is not a strict reading time as in normal written exams)

This paper will not be reproduced and lodged with Baillieu Library.

1 Short Answer (20 marks)

Question 1. **(20 marks)**

Answer the following questions with **brief**, **worded** responses. Your answers should contain no more than **four** dot points, **not** essays.

a) Define the term **abstraction** in its use as a fundamental technique in programming paradigms and explain how abstraction achieved in **object-oriented** programming. (4 marks)

b) Explain what it means for a class to be **immutable**. In your answer, explain how to **make** a class immutable in Java. (4 marks)

c) Describe the general type of problem solved by the **Factory Method** design pattern. Your answer should describe the components of the pattern and how they work together, and give a **real-world** example of how the Factory Method pattern is used. (6 marks)

d) Describe the **purpose** and **behaviour** of the following stream pipeline. Give a **real-world example** where you might use this code. Be sure to address each line of code in your answer. (6 marks)

```
List<String> deadlyPlagueNames = people.stream()
                                        .map(person -> person.getPlague())
                                        .filter(plague -> plague.getNumDeaths() > 1000000)
                                        .map(plague -> plague.getName())
                                        .collect(Collectors.toList());
```

2 System Design (30 marks)

Question 2. **(30 marks)**

The Lannisters have finally joined the 21st Century, and have asked you to design an inventory system to track all their possessions.

The system tracks all the people of Westeros, and keeps track of their name and age. People are either considered royalty, or slaves, and slaves can be used as either guards, or servants.

All slaves have a cost, and ***every*** *slave shares whether they are passive (i.e. following orders), or rioting/fighting back against their masters; all slaves are initially passive. Servants can also be commanded to serve any person. A royal can calculate their total wealth, and is served by a number of servants.*

There are also a number of castles in Westeros, and each castle has a value. A castle is also protected by a number of guards, and is owned by a single member of the royalty. However, a royal can only own up to three castles before someone tries to... acquire the castles from them.

For the questions below, you must rely **only** on the specification provided; you may make design decisions about method arguments, but do **not** make assumptions about behaviours that haven't been specified.

a) Using **only** the description given above, design a class diagram for the Lannister system and write Java class skeletons for your class design that show the attributes (including type) and methods signatures that are implied from the problem description (you must only submit the skeleton code for your classes not the class diagram).
Your classes must show privacy, class relationships, associations and have the correct data types to handled the multiplicities indicated in the class design. You do **not** need to show getters and setters, constructors (24 marks)

b) Describe **two** test cases you might write to test your implementation, stating specifically what *behaviour/component* you are testing, what an *input* might be, and the expected *output/result.* Do **not** write any Java code for this question.

Invalid example: *Test whether X object is created correctly/test whether Y variable is given a value.*
These tests are not related to your design or implementation, they are testing your ability to write code that works.

Invalid example: *Test that a castle can't be given a new owner if it is already owned.*
This is not specified, or even vaguely suggested by the description provided, nor is it a logical assumption.

Valid example:
Description (1 mark): ensure a person can't be instantiated without a name.
Input (1 mark): construct a person with an empty string as a name.
Output (1 mark): throw an exception.

This behaviour is not specified, but it is a **logical** and **sensible** assumption.
You cannot use this example in your answer. (6 marks)

3 Java Development (50 marks)

Question 3. **(25 marks)**

For this question you will implement classes for a simple self-driving car.

a) Implement an abstract `Sensor` class with the following: (6 marks)

i. One attribute: an `id` (identifies the sensor).

ii. Appropriate constructor(s), getter(s) and setter(s).

iii. A `measure` method that returns a numeric measurement from the world; *how* this might happen is not specified.

iv. A `toString` method that returns the sensor's `id`.

b) Implement a `Car` class with the following: (19 marks)

i. Three attributes: `poweredOn` (indicates whether the car is on or off; the car is initially off), `sensors` (a list of all sensors on the car), and `dangerThreshold` (the value above which the sensors are activated). You class should include a constructor, but you do not need to write getters and setters.

ii. A `measureSensors` method that gets measurements from every sensor on the car. This method should:

- **Print** the value of the measurement from each sensor on a new line, in the form `'Sensor [<id>]: <measurement>'`
- **Print** the number of sensors that measure above the `dangerThreshold` in the form `'<count> sensors above threshold'`
- **Return** the number of sensors that measure above the `dangerThreshold`

iii. A `dangerImminent` method that returns true when:

- The number of sensors on the vehicle is less than 4, OR
- The number of sensors that measure **above** the danger threshold is 7 or more

iv. A `run` method that:

- Turns **on** the car
- Continually `operate`s the car while it is on
- Turns **off** the car when it is in imminent danger

Note: you may assume the `void operate()` method exists; you do not have to write it
Note: you do not need to implement behaviour to add `Sensor`s to the `Car`

Question 4. **(25 marks)**

a) Implement the method with the following signature: (15 marks)

```
public String decode(String message, String language,
                                Map<String, Map<String,String>> map)
```

that takes a message and attempts to *decode* its characters, where:

- `message` - the message to be decoded
- `language` - the *output* language of the message
- `map` - a dictionary mapping a *language* to another dictionary, which maps the *encoded words* to their *decoded equivalent* for that language.

***Algorithm*:**
The encoding of the message represents each *word* with **four** integers from 0-9; for example, the word `class` may be represented by the code `0203`.
Your method should retrieve the correct language dictionary from `map`. It should then break the input message into substrings of length four, use the dictionary to *decode* the substring and add it to the output, and finally return the complete decoded message.

Note 1: If `language` is not contained in `map`, your method should create and throw a `LanguageNotFoundException`; you may assume this class exists.

Note 2: If any code is not present in the dictionary, you should create and throw an `InvalidCodeException` exception; you may assume this class already exists.

Note 3: If `message` does not contain a multiple of four characters, you should **add zeroes** to the front of the message. For example, if the message was `11`, your method should modify it to be `0011`.

Example 1 (Invalid Language):
Input: `decode("", "Klingon", map)`
Output: `LanguageNotFoundException: 'Klingon' language not found`

Example 2 (Invalid Code):
Input: `decode("10100101", "Spanish", map)`
Output: `CodeNotFoundException: '1010' not found in Spanish dictionary`

Example 3 (Valid Input):
Input:

```
english.put("0001", "Hello");
english.put("0002", " ");
english.put("0003", "World");
map.put("English", english);
decode("0100020003", map);
```

Output: `"Hello World"`. Note that extra zeroes were added to the front of the message to make it a multiple of four characters, allowing us to get the code `0001` for `Hello`.

b) Write a Java program with the following specification: (10 marks)

Your program, named `WordCount.java` will:

i. read data from a file named `codedString.dat` in directory `/home/ubuntu/data`, which contains encoded sentences suitable to be decoded according the algorithm in part (a);
ii. decode the sentences using the method you implemented in part (a);
iii. count the number of occurrences (frequency) of each word in the file; and
iv. display each word and its frequency sorted from highest to lowest frequency.

The sentences in the file are separated by newline characters (i.e. each sentence is a new line in the file).

— End of Exam —