THE UNIVERSITY OF MELBOURNE
SCHOOL OF COMPUTING AND INFORMATION SYSTEMS

SUPPLEMENTARY EXAM

Semester 1, 2020

SWEN20003 Object Oriented Software Development

Exam Duration: 2 hours Total marks for this paper: 120

This paper has 8 pages

Authorised materials:
Students may NOT bring any written material into the room.
Students may NOT bring calculators into the room.

Instructions to invigilators:
Each student should initially receive a script book.
Students may NOT keep the exam paper after the examination.

Instructions to students:

- The exam has 5 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.
- At the end of the exam, submit the solution file, `<username>-final-exam.pdf`, as a Canvas Turnitin Assignment submission.

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

Continued overleaf . . .

1 Short Answer (24 marks)

Question 1. **(24 marks)**

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

a) List and explain **two** of the four symptoms of poor software design. (4 marks)

b) Explain the term *polymorphism*, and **list** the ways it is achieved in Java. (4 marks)

c) Describe the general type of problem solved by the *Strategy* design pattern; in your answer, describe the components of its design and how they work together. (4 marks)

d) Give **two** reasons (with **justification**) why *generically typed classes* are *more appropriate* than standard classes in some circumstances. (4 marks)

e) List and describe **two** benefits of using software design tools (such as UML). (4 marks)

f) 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. (4 marks)

```
List<Item> result = products.stream()
                        .filter(p -> p.containsCategory("Vegetable"))
                        .filter(p -> p.getCost() < 4.95)
                        .sorted((p1, p2) -> (int) (p1.getCost() - p2.getCost()))
                        .collect(Collectors.toList());
```


Continued overleaf . . .

2 System Design (30 marks)

Question 2. **(30 marks)**

You have joined budding games company Eleanorus as lead designer for their "Battle Royale" game *PUBx*.

PUBx has two main types of game assets: vehicles, and players. All game assets are defined by their position in the world, and the mesh (or 3D model) that represents them. All game assets can also move.

The two main vehicle types are planes and tanks. All vehicles have a maximum speed, and may be *active* or *destroyed*. Tanks also have fuel, and may have up to four players inside it. Planes have no further characteristics.

All players keep track of how many enemies they have knocked out, as well as how much ammo they have remaining, and each player can fire their weapon. Players also share a count of how many players are active in the game. Players can be either *human* or *AI*, and human players all have a username. AI players have no further characteristics.

Finally, tanks and human players can be *controlled* by the person playing the game.

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, draw a UML class diagram for *PUBx*. In your class diagram show the attributes (including type) and methods that are implied from the problem description. You must show class relationships, association directions and multiplicities. You do **not** need to show getters and setters, or constructors.

Note: You may assume that `Position`, `Mesh`, and `Input` are provided to you as libraries; you do not need to include them as separate classes in your UML diagram. (24 marks)

b) Describe **two** test cases you might write to test your design, 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. (6 marks)

Continued overleaf . . .

3 Java Development (66 marks)

Question 3. **(22 marks)**

For this question you will implement classes for a simple location search system. You may assume that the enum `LocationType` (e.g. Restaurant) exists.

a) Implement an **immutable** `Position` class with the following: (11 marks)

i. Two attributes: `x` and `y` position.

ii. Appropriate initialization, accessor, and mutator methods.

iii. A method to calculate the Euclidean (straight-line) distance between two `Position` objects.

iv. A `toString` method that returns the `x` and `y` position in "co-ordinate" form; for example, a `Position` at the origin would return `(0.0, 0.0)`.

v. A `compareTo` method that results in `Position` objects being arranged in *increasing* order of Euclidean distance from the origin; for example, if `c1` is at (2, 3) and `c2` is at (4.2, 5), `c1.compareTo(c2)` should be negative.

b) Implement a `Location` class with the following: (7 marks)

i. Two attributes: a `Position`, and a `LocationType`

ii. Appropriate initialization, accessor, and mutator methods.

iii. A method to calculate the distance between two locations.

iv. A `toString` method that returns the type of the `Location`, and the `x` and `y` position in "co-ordinate" form; for example, a restaurant at the origin would return `RESTAURANT at (0.0, 0.0)`.

v. A `compareTo` method that results in `Location` objects being arranged using their `Position`.

c) Implement a `Map` class with the following: (4 marks)

i. One attribute: a list of `Location` objects.

ii. A method to add `Location` objects to the `Map`.

iii. A method to sort the list of `Location` objects.

Continued overleaf . . .

Question 4. **(24 marks)**

In this question you will implement the method

```
public String filterAndConcat(String s1, String s2, String illegalChars, int minFreq)
```

that takes two strings and concatenates them together, where:

- `s1` - the first `String` to be concatenated
- `s2` - the second `String` to be concatenated
- `illegalChars` - a `String` that contains characters that **should not** appear in the final output
- `minFreq` - the minimum frequency required for a character to be included in the output

***Algorithm*:**
Concatenate ("add") the two inputs `s1` and `s2` into a single `String`, then count the occurrence of each character. The output should then contain **only** the characters that appear in the concatenated `String` with **at least** `minFrequency`.

Note 1: All spaces must be removed from the inputs before concatenation.

Note 2: If any illegal characters are found in the inputs, your method should create and throw an `IllegalCharacterException` exception; you may assume this class already exists.

Example 1 (Invalid Input):

Input: `filterAndConcat("Hello", "Hell", "Hll", 2)`

Output: `IllegalCharacterException: 'H' found in input string.`, since one of the illegal characters (`'H'`) was found in one of the inputs.

Example 2 (Valid Input): HelloHell

Input: `filterAndConcat("Hello", "Hell", "", 2)`

Output: `"HellHell"`, since every character but `'o'` appears at least as many times as `minFreq`.

Example 3 (Valid Input):

Input: `filterAndConcat("You know nothing", "John Snow", "", 9)`

Output: `""`, since none of the characters meet the frequency threshold.

Example 4 (Valid Input):

Input: `filterAndConcat("Android good", "Apple bad", "", 3)`

Output: `"dodoodd"`, noting that spaces have been removed from the inputs, and that neither `'a'` or `'A'` appears in the output as they are *different* characters.

Continued overleaf . . .

Question 5. **(20 marks)**

Hard Question! In this question you will implement a small object oriented system using generics. Assume the `Mathematical<T>` interface exists, which declares two abstract methods:

`T add(T item)`	Computes the "addition" of two objects of type `T`.
`T subtract(T item)`	Computes the "subtraction" of two objects of type `T`.

a) Implement the `MathsMap<K, V>` class. This class should have an instance variable `map` of type `HashMap`, where the types of the key and value are the same as those of the `MathsMap`. You should also implement appropriate getters, setters, and constructors, for this class.

Your class definition should begin with:

```
public class MathsMap<K, V> implements Mathematical<MathsMap<K, V>>
```

(6 marks)

b) Implement the following methods for the `MathsMap` class:

`void put(K key, V value)`	Inserts the argument `value` into the `HashMap` with the associated key `key`.
`void remove(K key)`	Removes the argument `key` from the `HashMap`.
`MathsMap<K, V> add(MathsMap<K, V> item)`	Returns a new `MathsMap` that contains a `HashMap` with the key-value pairs of the input added to the calling object's map. If both maps share a key, only the calling object's value should be retained.
`MathsMap<K, V> subtract(MathsMap<K, V> item)`	Returns a new `MathsMap` that contains a `HashMap` with any keys contained in the argument *removed* from the calling object's `HashMap`.
`String toString()`	Returns a `String` representing the contents of the `HashMap`.

Example: The code below is an example of how the `MathsMap` class could be used.

```
MathsMap<Integer, String> mathmap = new MathsMap<Integer, String>();
MathsMap<Integer, String> mathmap2 = new MathsMap<Integer, String>();

mathmap.put(0, "Apple");
mathmap.put(1, "Chrome");
mathmap2.put(0, "Android");
mathmap2.put(2, "Samsung");

System.out.println(mathmap);
System.out.println(mathmap2);
System.out.println(mathmap.subtract(mathmap2));
System.out.println(mathmap2.add(mathmap));
```

The output of each line would be `{0=Apple, 1=Chrome}`, `{0=Android, 2=Samsung}`, `{1=Chrome}`, and `{0=Android, 1=Chrome, 2=Samsung}`. The add method combines the two maps, with the calling map's keys taking preference. The subtract method removes all duplicate keys. (14 marks)

Continued overleaf . . .

4 Appendix

Hashmap

```
public class HashMap<K,V>
        extends AbstractMap<K,V>
        implements Map<K,V>, Cloneable, Serializable
```

The `HashMap` class, in the `java.util` package, implements the Map interface, which maps keys to values. Any non-null object can be used as a key or as a value.

`HashMap()`	Constructs an empty HashMap with the default initial capacity (16) and the default load factor (0.75).
`boolean containsKey (Object key)`	Returns `true` if this map contains a mapping for the specified key.
`boolean containsValue (Object value)`	Returns `true` if this map maps one or more keys to the specified value.
`Set<Map.Entry<K, V>> entrySet()`	Returns a `Set` view of the mappings in the map.
`V get(Object key)`	Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
`Set<K> keySet()`	Returns a `Set` view of the keys contained in this map.
`V put(K key, V value)`	Associates the specified value with the specified key in this map.
`void putAll(Map<? extends K, ? extends V> m)`	Copies all of the mappings from the specified map to this map.
`boolean remove(Object key)`	Removes the mapping for the specified key from this map if present.
`int size()`	Returns the number of key-value mappings in this map.

Continued overleaf . . .

ArrayList

```
public class ArrayList<E>
        extends AbstractList<E>
        implements List<E>, RandomAccess, Cloneable, Serializable
```

The `ArrayList` class, in the `java.util` package, a resizable-array implementation of the `List` interface.

`ArrayList()`	Constructs an empty list with an initial capacity of ten.
`boolean add(E e)`	Appends the specified element to the end of this list.
`void add(int index, E element)`	Inserts the specified element at the specified position in this list.
`boolean equals(E element)`	Compares the specified object with this list for equality.
`E get(int index)`	Returns the element at the specified position in this list.
`int lastIndexOf(Object o)`	Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
`E remove(int index)`	Removes the element at the specified position in this list.
`boolean remove(Object o)`	Removes the first occurrence of the specified element from this list, if it is present.
`E set(int index, E element)`	Replaces the element at the specified position in this list with the specified element.
`int size()`	Returns the number of elements in this list.

— End of Exam —