The University of Melbourne
School of Computing and Information Systems

Final Exam

Semester 2, 2017

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 6 questions across 3 sections, and all questions must be attempted. Questions should all be answered in the script books provided, **not** the exam paper. Start the answer to each question on a new page in the script book.
- Answer all questions on the right-hand lined pages of the script book. The left-hand unlined pages of the script book are for draft working and notes and will **not** be marked.
- Ensure your student number is written on all script books during writing time.
- The marks for each question are 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.
- The section titled "Appendix" gives the documentation for several Java classes that you **can** use in your questions. You are not required to use all the listed classes and methods.
- Worded questions must all be answered in English, and code questions must all be answered in Java.

This paper will 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. Note the amount of time associated with each question; your answers should contain no more than **four** dot point, **not** essays.

a) Explain the term *encapsulation*, and how it relates to privacy and information hiding. Include details of how you can achieve encapsulation in Java in your answer. (4 marks)

b) Give **two** reasons (with **justification**) why tools like UML are useful to developers when designing software. (4 marks)

c) Explain the concept of *design patterns*, and provide **two** reasons why we would use a design pattern when writing software. (4 marks)

d) Define the terms *cohesion* and *coupling* in relation to software development. (4 marks)

e) Explain the term *automated software testing*, and give **two** reasons why we prefer automated testing over manual testing. (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<Member> result = members.stream()
                        .filter(m -> m.isActive())
                        .filter(m -> m.getFollowers() > 100000)
                        .sorted((m1, m2) -> m1.getFollowers() - m2.getFollowers())
                        .collect(Collectors.toList());    .
```


Continued overleaf . . .

2 Design (40 marks)

Question 2. **(16 marks)**

Figure 1 below shows a UML class diagram of the Strategy design pattern. You must *analyse* and *interpret* the diagram, and answer the following questions.

A motivating example for the Strategy pattern:
When using a spreadsheet or database (like Excel) it is highly likely that the user will want to sort *their data. However, given the wide variety of data types that may be used, it is difficult to develop a single, universal approach to comparing and arranging data.*

Critique the use of the Strategy pattern on the database example above. In your answer...

a) Explain the Strategy design pattern; specifically, describe what *type* of problem the Strategy pattern solves, and how. (4 marks)

b) Describe **one** benefit of using this design pattern. In your answer, compare this design with a solution that does **not** use the strategy pattern. (4 marks)

c) Describe the role of the Strategy pattern's participants, and how they collaborate; specifically, what purpose do the *Application*, *Strategy*, *Strategy1*, and *Strategy2* classes/interfaces have, and how do they work together? (4 marks)

d) Describe a real-world application of the Strategy pattern; your answer must be **unrelated** to comparing and sorting data. Make sure to highlight *why* the Strategy pattern is appropriate for your example. (4 marks)

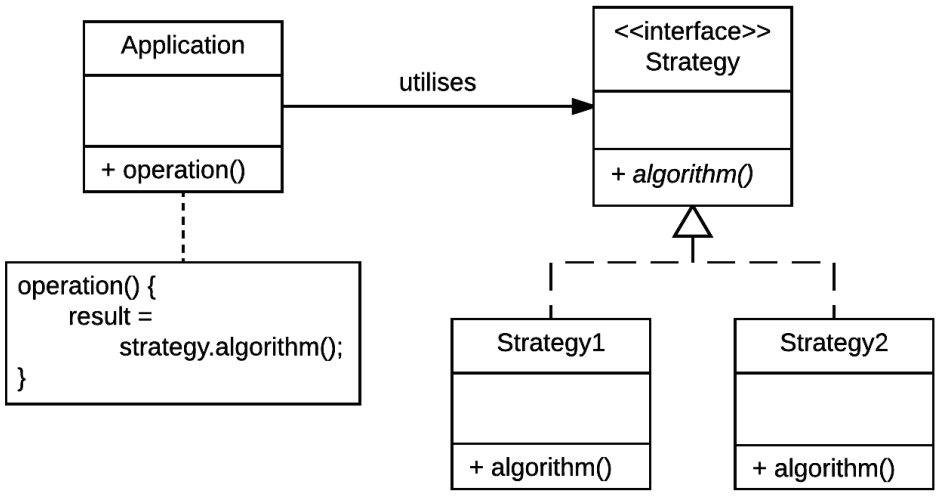

Figure 1: UML class diagram of the Strategy Pattern

 Continued overleaf . . .

Question 3. **(24 marks)**

You have been "hired" by the First Order to design a software system for managing the new Death Star (because the first three were total successes...) as part of their plot for galactic domination.

The Death Star employs two types of staff: soldiers and workers. The system records the name, seven character identification tag, and the rank of each staff member, as well as the room number they have been allocated. Workers are also tracked by the name of the role they have on the Death Star, while soldiers are tracked by the number of lasers they've fired, and the number of targets they hit (which is approximately zero for most Stormtroopers...).

Since the first three Death Stars were destroyed, the First Order is adding a number of defence items to version four, that the system must track. The system records the 3D location (X, Y, Z) of each item, as well as its cost. Each defence item also has a *minimum* number of soldiers required, and a list of all the soldiers currently on staff.

In addition, space defences like starships and fighters have an altitude and size, while ground defences like shields and lasers record an angle of orientation, and the amount of energy they have stored.

While tracking all these objects, the system maintains a count of the total number of staff employed, and the total cost of all its defences.

a) Using **only** the description given above, draw a UML class diagram for the Death Star. In your class diagram show the attributes (including type) 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. (18 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 (56 marks)

Question 4. **(20 marks)**

For this question you will implement classes for a basic card game. You may assume that the enums `Rank` and `Suit` exist, as defined in lectures. In addition, the `Rank` class has the following method:

`int getPoints()`	Returns the number of points a given `Rank` would score.

a) Implement an **immutable** `Card` class with the following: (6 marks)

i. Two attributes: a `Rank` and a `Suit`.

ii. Appropriate initialization, accessor, and mutator methods.

iii. A `compareTo` method that results in `Card` objects being arranged in *decreasing* order of the number of points given by the `Rank`; for example, if `c1`'s rank is worth less points than `c2`'s, `c1.compareTo(c2)` should be negative.

b) Implement a `Player` class with the following: (14 marks)

i. Two attributes: a numeric `playerID`, and a list of `Card` objects (the player's hand)

ii. Appropriate initialization, accessor, and mutator methods.

iii. A method to add a `Card` to the player's hand.

iv. A method to calculate the total score of the player's hand.

v. A method to sort the player's hand.

vi. A method to *play* the best `Card`; the best `Card` is the one with the highest score. The `Card` played should be returned, and removed from the player's hand as a result.

Question 5. **(16 marks)**

In this question you will implement the `rectEncryption` encryption algorithm. The algorithm takes a line of text and converts it to a *rectangle* of width `n` characters, as below. All characters are converted to upper case, with spaces replaced by the `#` symbol, and `*` characters added to fill up the rectangle. The text is then encrypted by reading down the *columns* from top to bottom.

Write the `public String rectEncryption(String text, int n)` method that implements this algorithm. You may define any additional classes/methods you feel you need.

Example:

Input: `rectEncryption("We have to wait so long for Game of Thrones!", 8)`

Rectangular form:

```
WE#HAVE#
TO#WAIT#
SO#LONG#
FOR#GAME
#OF#THRO
NES!****
```

Output: `WTSF#NEOOOOE###RFSHWL##!AAOGT*VINAH*ETGMR*###EO*`

Continued overleaf . . .

Question 6. **(20 marks)**

Hard Question! In this question you will implement a small object oriented system using generics. Assume the `Categoriser<In, Out>` interface exists, which declares one abstract method:

`Out categorise(In input)`	Computes a category of type `Out` for an argument of type `In`.

a) Implement the class `StringCategoriser`, including the `categorise` method. This method categorises non-empty `String` objects (i.e. not `""`) by returning the *first letter* of the input variable (as a `String`).
Your class definition should begin with:

```
public class StringCategoriser extends Categoriser<String, String>
```

(4 marks)

b) Implement the `CategorisedMap<K, V, C extends Categoriser<V, K>>` class where `K` and `V` are types for the key and value of the map, respectively, and `C` is a type that can categorise `V` into type `K`. This class should have two instance variables: a `HashMap` where the key is of type `K`, and the value is of type `ArrayList<V>`; and a variable `categoriser` of type `C` to categorise values. Implement an appropriate constructor for this class.
Your class definition should begin with:

```
public class CategorisedMap<K, V, C extends Categoriser<V, K>>
```

For example, `CategorisedMap<String, String, StringCategoriser>` stores lists of `String` objects, categorised by the first letter of the `String`. (4 marks)

c) Implement the following methods for the `CategorisedMap` class:

`void put(V value)`	Inserts the argument `value` into the `HashMap`, in the `ArrayList` associated with the *category* (key) computed by `categoriser`.
`int getCategoryCount(V value)`	Returns the number of elements in the category where `value` *would be* stored, or 0 if the category isn't present.

`String toString()`	Returns a `String` representing the contents of the `HashMap`. The `String` should be in the form
	`Key1: Value1, Value2, Value3, ...`
	`Key2: Value1, Value2, Value3, ...`
	Note: Extra commas will **not** be marked down.

Example: The code below is an example of how the `StringCategoriser` and `CategorisedMap` classes could be used.

```
CategorisedMap<String, String, StringCategoriser> categories
                = new CategorisedMap<>(new StringCategoriser());

categories.put("Hulk");
categories.put("Hawkeye");

System.out.println(categories);
System.out.println(categories.getCategoryCount("Hammerhead"));
```

The output of this code would be `"H: Hulk, Hawkeye,"`, followed by `"2"`: both "Hulk" and "Hawkeye" start with "H", so they are in the "H" category; "Hammerhead" would be added to the "H" category, so `getCategoryCount` returns 2. (12 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 a new, empty HashMap with a default initial capacity (11) and load factor (0.75).
`HashMap(int initialCapacity)`	Constructs a new, empty HashMap with the specified initial capacity and default load factor (0.75).
`boolean contains (Object value)`	Tests if some key maps into the specified value in this hashmap.
`boolean containsKey (Object key)`	Tests if the specified object is a key in this hashmap.
`boolean containsValue (Object value)`	Returns true if this hashmap maps one or more keys to this value.
`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.
`Set<Map.Entry<K, V>> entrySet()`	Returns a Set view of the mappings in the map.
`V put(K key, V value)`	Maps the specified key to the specified value in this hashmap.
`V remove(Object key)`	Removes the mapping for the specified key from this map if present.
`int size()`	Returns the number of elements in this list.

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.
`E remove(int index)`	Removes the element at the specified position in this list.
`E set(int index, E element)`	Replaces the element at the specified position in this list with the specified element.
`boolean remove(Object o)`	Removes the first occurrence of the specified element from this list, if it is present.
`Iterator<E> iterator()`	Returns an iterator over the elements in this list in proper sequence.
`int lastIndexOf(E e)`	Returns the index in this list of the last occurence of the specified element, or -1 if the list does not contain this element.
`ListIterator listIterator()`	Returns an iterator of the elements in this list (in proper sequence).
`ListIterator listIterator(int index)`	Returns a list iterator of the elements in this list (in proper sequence), starting at the specified position in the list.
`int size()`	Returns the number of elements in this list.

— End of Exam —

Library Course Work Collections

Author/s:
Computing and Information Systems

Title:
Object Oriented Software Development, 2017, Semester 2, SWEN20003

Date:
2017

Persistent Link:
http://hdl.handle.net/11343/212975

File Description:
SWEN20003