# COMP2511 Sample Final Exam (19T2)

## Part 1 (of 3): Multiple Choice (20 marks)

There will be 15 to 20 multiple choice questions. The marks for each question may vary.

For the sample multiple choice questions, see the following three quizzes:

- Quiz 1 (click here)
- Quiz 2 (click here)
- Quiz 3 (click here)

---

## Part 2 (of 3): Short Answer (30 marks)

Note: The marks for each question varies.

### Part 2: Q1 (6 marks)

Examine the code and answer the following questions

```
import java.time.LocalDate;
public class Account {

      public int accountNumber;
      public double balance;

      public static String printStatementHeader() {
            return "Statement for " + LocalDate.now().getMonth();
      }
}
public class SavingsAccount extends Account {

      private static double monthlyAcessFee;
      public static String printStatementHeader() {
            return "Statement for " + LocalDate.now().getMonth() + "\n" +
"Month access fee is: " + monthlyAcessFee;
      }

      public static void main(String[] args) {
            Account a1 = new SavingsAccount();
            System.out.println(a1.printStatementHeader());
      }
}
```

(A) In the class `Account`, the instance variables are defined to be public. State which OO principle is violated here and why it is a bad practice for instance variables to be public . (2 marks)

(b) When the above code is run, the output produced is as follows:

"Statement for NOVEMBER"

Explain why the output above is produced. (2 marks)

(c) Describe what changes would you make to the code above to ensure that the classes `Account` and `SavingsAccount` are *immutable*? (2 marks)

## Part 2: Q2 (4.5 marks)

Name the design pattern suitable in each of the scenarios described below (1.5 marks)

(A) The `Collections.sort()` method takes in a `Comparator` object to specify how elements should be compared. By varying the comparator, you can sort by different criteria. Name the design pattern used in the implementation of the `Comparator`.

(B) Adding operations to classes without changing the class.

(C) Restricting the instantiation of a class to a single instance.

## Part 2: Q3 (3 marks)

The user of a class `Car` below wishes to maintain a collection of `Car` objects such that they can be iterated over in some specific order.

```
public class Car {
   private String manufacturer;
   private int age;
}
```

Open the eclipse project part2Q3 (click for sample code). The class `Car` has been defined for you. Complete the class `CarComparator` implementing a suitable `Comparator`, that enables the collection to be sorted by `{manufacturer, age}`. i.e. sort first by manufacturer, and sub-sort by age.

### Part 2: Q4 (4 marks)

Open Eclipse Project Part2Q4 (click for sample code). Open the file `Logger.java`. This source code represents a logger class which defines a method `log()` which writes to the console. The logger class can be instantiated only once in the application; it is to ensure that all components of the application makes use of that same logger instance.

Using an appropriate design pattern, write a Java class for this implementation in the file `Logger.java` (assuming the application is not multi-threaded environment).

### Part 2: Q5 (2 marks)

Provide one important difference between the access modifiers `private` and `protected`.

And few more questions like the above five questions ...

## Part 3 (of 3): Programming Questions (50 marks)

Note: The marks for each question varies.

### Part 3: Q1

Given a problem specification, provide your OO design for a possible solution (similar to Ass1/lab03). You need to provide the required:

- interfaces (with brief comments)
- classes (with brief comments) and
- method signatures (with brief comments)

For this question,

- you do NOT need to implement methods.
- you do NOT need to draw UML diagram(s).

### Part 3: Q2

Given a problem specification, **implement** a solution in Java. The question will be similar to Lab09 (on generics and programming by contract).

For this question,

- you need to implement methods such that they **pass the given Junit tests**.
- you do NOT need to draw UML diagram(s).

## Part 3: Q3

Given a problem specification, **implement** your solution using a suitable design pattern (one of the design patterns discussed in the course). You need to **implement** the required:

- interfaces
- classes
- methods

For this question,

- you need to design and implement interfaces, classes and methods such that **pass the given Junit tests**.
- We will test your solution using extensive test cases (not provided in the question).
- you do NOT need to draw UML diagram(s).

## Part 3: Q4

Given a problem specification, **implement** your solution using a suitable design pattern (one of the design patterns discussed in the course). You need to **implement** the required:

- interfaces
- classes
- methods

For this question,

- you need to design and implement interfaces, classes and methods such that **pass the given Junit tests**.
- We will test your solution using extensive test cases (not provided in the question).
- you do NOT need to draw UML diagram(s).

---

End

---