The University of Nottingham Ningbo China

SCHOOL OF COMPUTER SCIENCE

A LEVEL 2 MODULE, AUTUMN SEMESTER 2019-2020

SOFTWARE MAINTENANCE

Time allowed: **Sixty (60) Minutes (One Hour)**

Candidates may complete the front cover of their answer book and sign their desk card but must NOT write anything else until the start of the examination period is announced

Answer All FIVE questions

No calculators are permitted in this examination.

Dictionaries are not allowed with one exception. Those whose first language is not English may use a standard translation dictionary to translate between that language and English provided that neither language is the subject of this examination. Subject specific translation dictionaries are not permitted.

No electronic devices capable of storing and retrieving text, including electronic dictionaries, may be used.

DO NOT turn examination paper over until instructed to do so

INFORMATION FOR INVIGILATORS:
Collect both the exam papers and the answer booklets at the end of the exam.

SECTION A: Object-Oriented Concepts and Maintenance Principles

Question 1: Abstract class and interface are two mechanisms for supporting abstract definition in Java programming language, but there are big differences between the abstract class and interface.

a. How do you differentiate between both types?

[4 Marks]

b. Provide two examples in Java codes, one on abstract class and one on interface (provide only class signatures).

[4 Marks]

Question 2: Class diagrams are visual representations of the static structure and composition of a system using conventions set by the Unified Modeling Language (UML).

a. Classes in a class diagram are represented by boxes that are partitioned into 3 sections. Create an object called "User" and illustrate those sections with a simple class diagram. The object "user" has an id, name and email with respective setter and getter methods.

[4 Marks]

b. Classes are interrelated to each other in specific ways. Relationships in class diagrams include "Association", "Aggregation", "Generalization/Inheritance", "Dependency/Realization" and "Composition". Provide 4 examples of relationship of any 2 classes each in UML notation.

[8 Marks]

Question 3: Code refactoring refers to changes made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behaviors.

a. Identify 4 issues in the code snippets below that requires code refactoring, assuming codes are commented and documented appropriately.

[4 Marks]

b. Provide solutions to reflect the changes of above issues based on code refactoring methods you have learnt.

[4 Marks]

```
public class GalleryApp {
     public static void main(String[] args) {
          Art a1 = new Art();
          a1.setArtName("Mona Lisa");
          a1.setArtPrice(300000000);
          a1.setArtSizeX(77);
          a1.setArtSizeY(53);
          a1.setArtArtist("Leonardo Da Vinci");
```

```
        Gallery g = new Gallery();
        g.addArt(a1);
        g.displayArts(); }
}
```

```
public class Gallery {
     private Art[] arts;
     private int count;
     private ArrayList<String> employeeName;
     private ArrayList<Integer> employeeId;

     public Gallery() {
          arts = new Art[20];
          count = 0;
          employeeName = new ArrayList<>();
          employeeId = new ArrayList<>();
     }
     public void addArt(Art art) { … }
     public void displayArts() { … }
     public void addEmployeeName(String name) { … }
     public void addEmployeeId(int id) { … }
     public void printAllEmployeeInfo() { … }
}
```

```
public class Art {
     public String n, a;
     public double p;
     public int x, y;
     public static int numOfArt;

     public Art() { … }
     public void setArtName(String n) { … }
     public void setArtPrice(double p) { … }
     public void setArtSizeX(int x) { … }
     public void setArtSizeY(int y) { … }
     public void setArtArtist(String a) { … }
     public String getArtInfo() { … }
}
```

End of Section A: Total 28 marks

SECTION B: Design Principles

Question 4: We have the following `Dog` interface that contains a `speak()` method to be implemented. `GoldenRetriever` and `Husky` are two classes implement the Dog interface which have different speak behavior as shown in the code snippets below.

```
interface Dog {
      public void speak(String str);
}
```

```
public class GoldenRetriever implements Dog {
	@Override
	public void speak(String str) {
		System.out.println("I like pie and muffin.");
	}
}
```

```
public class Husky implements Dog {
	@Override
	public void speak(String str) {
		System.out.println("Dude, what's up?");
	}
}
```

a. Let's define a **factory class** that returns the corresponding `Dog` object according to their names input as string. This can be accomplished through a method named `getDog()`. Implement this method by fill in the blanks below.

Hint:

1. The `dogType` string should be either `GoldenRetriever` or `Husky`.
2. You can use `String.equalsIgnoreCase(String str)` for string comparison.

```
public class DogFactory {
	public static Dog getDog (String dogType) {
		if ( … )                                    [1 mark]
			…                                   [1 mark]
		else if (…)                                 [1 mark]
			…                                   [1 mark]
	}
	return null;
}
```

Now, we would like to create a cartoon dog that can speak like a Hero or a Villain. The `Hero` or `Villain` classes implement the `Cartoon` interface listed below. (*Definition of Villain: "A bad person who harms other people or breaks the law, or a cruel or evil character in a book, play or film" – dictionary.cambridge.org.*)

```
interface Cartoon {
	public void speakHero();
	public void speakEvil();
}
```

```
class Hero implements Cartoon {
	@Override
	public void speakHero() {
		System.out.println("I like to rescue!");
	}
	public void speakEvil(){} //do nothing
}
```

```
class Evil implements Cartoon {
	@Override
	public void speakEvil() {
		System.out.println("I like to destroy!");
	}
	public void speakHero(){}
}
```

a. We would like to make the Husky speak either "I like to rescue!", or "I like to destroy!", or its routine words "Dude, what's up?". Apply the **Adapter design pattern** to realize this requirement by fill in the blanks below.

```
public class CartoonAdapter implements Dog {
	Cartoon cartoon;
	public CartoonAdapter(String cartoonType) {
	// Hint: 1. cartoonType should be either "Hero" or "Villain".
	//       2. This constructor initialize objects/variables.
	//       3. You can use String.equalsIgnoreCase(String str) for
	//          string comparison.
		if (…) …                                          [1 mark]
		else if (…) …                                     [1 mark]
	}
	@Override
	public void speak(String cartoonType) {
	// Hint: 1. This method implements the Dog interface.
	//       2. Since it is defined in CartoonAdapter, it should let
	//          Cartoon speak. The cartoonType should be either
	//          "Hero" or "Villain".
	//       3. You can use String.equalsIgnoreCase(String str) for
	//          string comparison.
		if (…) …                                          [1 mark]
		else …                                            [1 mark]
	}
}
public class Husky implements Dog {
	CartoonAdapter cartoonAdapter;

	@Override
	public void speak(String cartoonType) {
	// Hint: 1. Either cartoonAdapter or Husky should speak.
	//       2. You can use String.equalsIgnoreCase(String str) for
	//          string comparison.
		if (…) …                                          [1 mark]
		else …                                            [1 mark]
	}
}
```

Question 5:

a. There are five main Object-Oriented Design Principles (SOLID). Define the Liskov's Substitution Principle. Give a counter example showing the case that violates the Liskov's Substitution Principle.

[4 Marks]

b. What is the purpose of "staging area" or "index" in Git? Through what command dose it interact with the local repository?

[2 Marks]

c. Which command is used to create a new branch named "dev" from the local repository AND then enter the working directory?

[2 Marks]

d. Discuss the key differences between MVC and MVP patterns?

[4 Marks]

End of Section B: Total 22 marks