200042105

B.Sc. in SWE 2nd Semester **13 April, 2022**

ISLAMIC UNIVERSITY OF TECHNOLOGY (IUT)

ORGANISATION OF ISLAMIC COOPERATION (OIC)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SEMESTER: FINAL EXAMINATION **SUMMER SEMESTER, 2020-2021**

DURATION: 3 HOURS **FULL MARKS: 150**

SWE 4201: Object Oriented Concepts I

Programmable calculators are not allowed. Do not write anything on the question paper.

Answer all 6 (six) questions. Marks of each question and corresponding CO and PO are written in the right margin with brackets.

---

1. A product management software for a superstore maintains a *List* of products to keep track of the items in the store. Pencils, notebooks, chocolates, and other items may be available in the super shop. Each type of product has its own set of attributes as well as some common ones like quantity, price, etc.

a) Create a *UpdateProductList* utility class comprising two methods: *AddProduct* and *RemoveProduct*, that adhere to the following criteria: 10 (CO2) (PO2)

- Any instance of a class that inherits the *Product* class will be allowed to be passed as a parameter to the methods. You should use *Generics* to implement this feature.
- The *AddProduct* method will increase the quantity of a product if it already exists in the *ProductList*. Otherwise, it will add a new type of product to the existing list.
- The *RemoveProduct* method will decrease the amount of an existing product by a certain quantity.
- You should be able to utilize these two methods without creating an instance of the *UpdateProductList* class.

b) The *RemoveProduct* method will not be able to remove a product from the list if it does not exist in the *ProductList*. In addition, if the number of items to be reduced exceeds the current quantity of a product, the system will be in a state of inconsistency. To prevent this from happening, the *RemoveProduct* method should throw exceptions in both of these scenarios. 10 (CO2) (PO2)
Create two custom exception classes, and update the *RemoveProduct* method's code to implement the required features.

c) "Having separate *AddProduct* and *RemoveProduct* methods rather than adding (*Array.Add*) to the *ProductList* directly from other classes is beneficial"- Do you agree with the statement? Justify your answer. 5 (CO1) (PO1)

2. Dr. Gregory House leads a team of diagnosticians as the Head of Diagnostic Medicine at the Princeton-Plainsboro Teaching Hospital in Princeton, New Jersey. His team comprises the following members:

- Dr. Eric Foreman, Neurologist.
- Dr. Allison Cameron, Immunologist.
- Dr. Robert Chase, Surgeon.

This hospital has a Hospital Management System in place. This system already has a *Physician* class for storing doctor information. *GroupLeader* and *Fellow* are the subclasses of the *Physician* class.

Dr. House belongs to the *GroupLeader* class, whereas the other doctors belong to the *Fellow* class. Each physician has a method called *CallForEmergency* that takes two arguments, *PatientID* and *WardName*, and returns *void*. If the *IsOccupied* attribute of a physician is *false*, only then he or she can respond to an emergency call.

a) Create a delegate that can refer to any physician's *CallForEmergency* method. Then create a method called *GetEmergencyDoctor* that will assign a method to this delegate while adhering to the constraints listed below. 2+5 (CO3) (PO3)

- You can check the availability of the fellow doctors of the Diagnostic Medicine department in any order. You can delegate the *CallForEmergency* method to the first doctor whose *IsOccupied* attribute is *false*.
- If none of them are available, you will assign Dr. House.

b) When a patient gets admitted to the Princeton-Plainsboro Hospital, the following information is stored in the system: 10 (CO2) (PO2)

- PatientID (auto-incrementing, starting from 0)
- Name
- Admitted Department (default value is 'Diagnostic Medicine')

Write the codes for the following classes:

- *Patient*
- *PatientAdmission*

The responsibility of the *PatientAdmission* class is as follows:

- Admitting a patient.
- Raising an *Event* when the patient gets admitted.
- Calling the *EventHandler*. The *EventHandler* should follow the Open-Close Principle.

c) Based on the scenario from Question **2.b)**, Write codes for a method that will handle the following responsibilities: 8 (CO2) (PO2)

I. Create a patient with arbitrary attribute values.
II. Create an instance of the *PatientAdmission* class to admit new patients.
III. Add a subscriber '*PagerService*' to the created *Event*.

The codes for the *Subscriber(PagerService)* should also be included. It will write a line to the console: "Informing Diagnostic Medicine Department … … "

3. ESILES Digital Platforms is a renowned software firm in Eriador, Middle-Earth. They are developing an employee management system. So far, they have come up with the following system architecture:
There is a base *Employee* class that will have two child classes: *BusinessEmployee*, and *DeveloperEmployee*. There is also another type of employee that handles client-side operations as well as development issues. So, they have created another class named, *ProductEngineer*. The *ProductEngineer* class inherits both from *BusinessEmployee* and *DeveloperEmployee* class. Some information regarding these four classes shown in Figure 1.

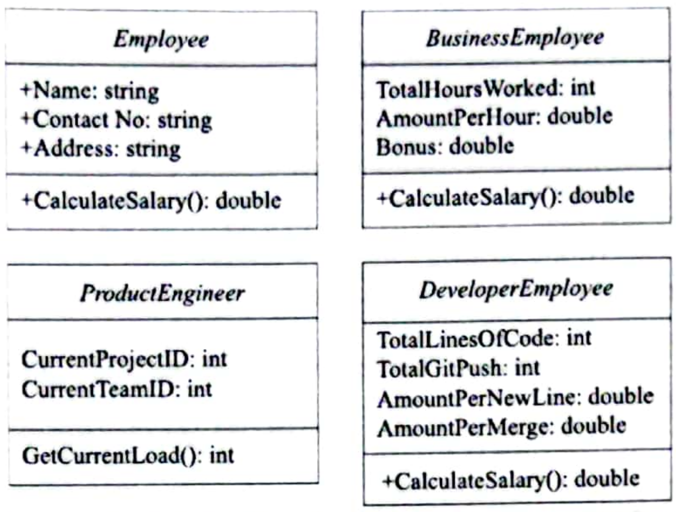

Figure 1: Classes associated with the Employee Management System

The *Employee* class is an abstract class since it contains an abstract method, *CalculateSalary*. *BusinessEmployee* and *DeveloperEmployee* classes override this method.

a) Is it possible to use C# to construct this same class architecture? Provide relevant explanations for your answer. 7 (CO2) (PO2)

b) Write necessary codes to implement the aforementioned scenario considering the four types of employees adhering to the following constraints: 12 (CO2) (PO2)

- Multiple inheritance is permitted only when there is at most one base class and the rest are interfaces.
- For the class methods (CalculateSalary, GetCurrentLoad), you can use any implementation that makes use of the existing class attributes.

c) Draw the final class diagram based on your solution. 6 (CO1) (PO1)

4. In the game "Guess It Right", the user is shown a random number and asked to guess whether the next number to be shown is greater or lesser than the current number. As long as the player's guess is correct, the game continues. The player receives one point for each correct guess. The numbers that appear are always even and between 10 and 100. The game also saves the player's name, score of the previous game, and the all-time highest score. If someone beats the highest score, the prior scorer's name is removed and replaced with the new scorer's name. A player's name cannot exceed 20 characters and must begin with an uppercase letter and end with a lowercase letter. The game also tracks how many times it has been played. After the game begins, it reads prior records from a file called "data.txt," and when the game is closed, it rewrites the new information into the same file. 25 (CO3) (PO3)

To imitate the game mentioned above, create a class called *GuessGame* and include the necessary and appropriate member variables and functions. In order to play the game, the following functionality must be implemented.

- Both parameterized and non-parameterized constructors should be available. The highest score as well as the name of the scorer will be passed to the parameterized constructor.

- There should be a *private* method with the method signature:
  private bool VerifyName (string name)
  This method will return *true* if the player's name is valid and *false* otherwise.
- There should be two *private* methods that will contain the following method signatures:
  private bool UpdateLastScore (string name, int score)
  private bool UpdateHighScore (string name, int score)
  These methods will be responsible for updating the score information in the text file.
- There should be a *public* method, *Play* that will coordinate all the functions and simulate the game.

Inclusion of any attributes or member methods to simulate the game is permitted.

5. Assume that the information shown in Table 1 is saved in your project directory as a CSV (comma-separated-value) file. Also, imagine that the tables have more rows than the four shown in the example. *PersonalInfo* and *ContactInfo* are the two classes you have previously built. The class attributes are similar to the respective table column names.

Table 1: Personal Information

ID	Name	Birthday	Age	Gender
1	Michael Scott	March 15, 1965	57	Male
2	Angela Martin	November 11, 1974	47	Female
3	Jim Halpert	October 1, 1978	43	Male
4	Pam Beesly	March 25, 1979	42	Female

Table 2: Contact Information

ID	Phone	City	State
1	(526) 251-2198	Scranton	Pennsylvania
2	(936) 238-2315	Dayton	Ohio
3	(875) 419-8651	Columbus	Ohio
4	(462) 668-3406	Harrisburg	Pennsylvania

a) Write the code for a method that reads data from the CSV files and stores it in two distinct Lists that correspond to the two classes you previously defined, *PersonalInfo* and *ContactInfo*. 10 (CO2) (PO2)

b) Write the code for a method to perform the following information operations: 15 (CO2) (PO2)

- Print the first name, age, gender, and city of the first person in the list who lives in 'Pennsylvania' state.
- Print the name of the state, gender, and average age of males and females for every state in descending dictionary order of the state name. An example is given below:

```
Pennsylvania   male   57
Pennsylvania   female   42
Ohio   female   47
Ohio   male   43
```

(Any type of query or iteration on the previously created *Lists* must be performed using *LINQ*.)

6. a) Consider the following code snippet: 10 (CO1) (PO1)

```
1.  class Ride {
2.      String vt;
3.      int distance;
4.      int nop;
5.
6.      int gFare() {
7.          int f;
8.          if (vt == "sedan") {
9.              f = (50 + distance * 30) / nop;
10.         } else if (vt == "motorBike") {
11.             f = Math.max(25, distance * 20) / nop;
12.         } else {
13.             if (distance < 10)
14.                 f = 300 / nop;
15.             else
16.                 f = distance * 30 / nop;
17.         }
18.
19.         return f - (f % 5);
20.     }
21.
22.     boolean isRideValid() {
23.         if (vt == "sedan") {
24.             return nop <= 4 && distance <= 25;
25.         } else if (vt == "sevenSeater") {
26.             return nop <= 7 && distance >= 10;
27.         } else {
28.             return nop == 1 && distance <= 10;
29.         }
30.     }
31. }
```

Code Snippet 1: Ride Class for question 6.

Your objective is to locate as many code smells as possible (At least 5). Mention the line numbers where there is a code smell.

b) Refactor the code by removing the code smells from Code Snippet 1. 8 (CO1) (PO1)

c) *"Any fool can write code that a computer can understand*
Good programmers write code that human can understand"
– Martin Fowler 7 (CO1) (PO1)

Give your opinion for or against this statement and provide necessary justification.