North South University

Department of Electrical and Computer Engineering

CSE215 Assignment-1

Spring 2023

Name: Shaheer Farrubar Shamsi

ID: 2222127642

Section: 1

Date of submission: 07.04.2023 (Date Extended by SfR1 sir)

Faculty: Dr. Shafin Rahman (SfR1)

CSE 215 Assignment-1

Sec: 1, 2 Faculty: SfR1

Marks: 100 **Deadline: 07.04.23**

Extended to 07 April by SfR1

Consider a class management system where courses offered for enrollment each have one or more sections. Students can be registered in only one section of a particular course, given that the timing does not overlap with any previously registered course. After the enrollment is done successfully, they should be able to see the list of courses they are enrolled in. Similarly, teachers should be able to see the list of sections they are assigned to and a list of students present in those sections.

a. Develop a JAVA program that will handle the mentioned requirements following the guidelines below: **(Note: You cannot use OOP concepts to accomplish this task. Don't use any other JAVA library classes except Array, Scanner and Date.)**
[90]

Answer:
Sourcecode(.java) attached in canvas submission.

b. Do you think any aspect of the program could be done differently if OOP concepts were to be introduced? Explain briefly and specific to your code. [10]

Answer:
If object-oriented programming (OOP) concepts were to be introduced, the entire program could be structured differently. There are many improvements that could have made the program memory-efficient and much more concise. Here are some aspects that could be implemented better using OOP concepts:

1. Encapsulation: Encapsulation refers to the practice of hiding the internal details of an object and exposing only what is necessary. In the given code, data fields such as id, name, password, coursesEnrolled, coursesTeaching in the Student and Teacher classes are currently public, which means they can be accessed and modified from any part of the code. However, it is considered a best practice to make these data fields private and provide public getter and setter methods to access and modify them. For example, in the Student class, the id, name, and password fields could be made private, and corresponding getter and setter methods could be added to provide controlled access to these fields.

e.g.,

```
class Student {
   private int id;
   private String name;
   private String password;
   private String[] coursesEnrolled;
         // Getter Setter
}
```

2. Inheritance: Inheritance is a mechanism that allows a class to inherit properties and methods from a parent class. In the given code, the Student and Teacher classes could potentially inherit from a common User class, which could contain common data fields and methods such as id, name, password, and login/logout functionality. This would allow for better code reuse and maintainability.

e.g.,

```
class User {
   private int id;
   private String name;
   private String password;

   // Getter and Setter
   // .. login/logout functionality
}
class Student extends User {
   private String[] coursesEnrolled;
   // .. rest of the code for Student class
}
class Teacher extends User {
   private Course[] coursesTeaching;
   // .. rest of the code for Teacher class
}
```

3. Polymorphism: Polymorphism allows objects of different classes to be treated interchangeably, providing greater flexibility and extensibility. In the given code, a common interface or abstract class could be introduced for the Student and Teacher classes to define common methods, such as listCourseStudents() for the Teacher class and viewCoursesEnrolled() for the Student class. This would allow for polymorphic behavior.

e.g.,

```
interface CourseManager {
   void viewCoursesEnrolled();
}
```

```
class Student implements CourseManager {
    // ... rest of the code for Student class
    @Override
    public void viewCoursesEnrolled() {
        // ... implementation of viewCoursesEnrolled() for Student
    }
}

class Teacher implements CourseManager {
    // ... rest of the code for Teacher class
    @Override
    public void viewCoursesEnrolled() {
        // ... implementation of listCourseStudents() for Teacher
    }
}
```

4. ArrayList: Collections like ArrayList provide dynamic data structures that can be used for better data management, manipulation, and scalability. In the given code, instead of using arrays and primitive arrays for storing courses and enrolled students, ArrayList could be utilized for better flexibility and ease.

The Java program could be more structured, modular, and extensible if we introduce OOP concepts, with better encapsulation, inheritance, polymorphism, and use of collections, leading to improved code maintainability, reusability, and scalability.