CSE 215 Assignment-2

Sec: 5 Faculty: SvA

Marks: 100 **Deadline: 20.12.22**

Submission Rules:

Answer all the questions following the given guidelines. Submit required .java files in a zip folder and a PDF file. Make sure to submit your assignment before the deadline 20/11/22 11:59PM to avoid penalty.

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. Once enrolled, they should be able to see the list of courses they are enrolled in. Similarly, teachers and TAs should be able to see the list of sections they are assigned to and a list of students present in those sections. However, only teachers can remove a student from their section.

a. Design a UML diagram to represent the classes and their relationships as given above. [15]
b. Develop a JAVA program that will handle the mentioned requirements following the guidelines below: [80]
(Note: You cannot use any other JAVA library classes except Array, ArrayList, Scanner and Date.)

- Use this **"Session"** class to store the database and invoke its methods when you need access to the database.

```
//A class to access the singular database and scanner instance
class Session{
   private static Session session = null;
   private User userList[] = new User[7];
   private Course courseList[] = new Course[6];
   public Scanner inputScanner = new Scanner(System.in);

   private Session(){
      createDatabase();
   }
    private static void createDatabase(){
       //place your database building code here
   }
   public static Session getSession(){
      if (session == null)
         session = new Session();
      return session;
   }
   public Course[] getCourseList(){
      return courseList;
   }

   public User[] getUserList(){
      return userList;
   }
```

- The execution class must have the following outline:

```
class CourseManagement {

   public static void main(String[] args) {
      while(true){

         //copy the following line wherever you need a session instance
         Session session = Session.getSession();

         System.out.print("Email:");
         String email = session.inputScanner.nextLine();
         System.out.print("Password:");
         String password = session.inputScanner.nextLine();

         //place rest of your code here

      }
   }

    public static User login(String email, String password) throws Exception{
        //place your login logic here
       //for wrong credential: throw new Exception("User not found");
       //call getMessage() on the exception instance to get the error message
   }

}
```

(**Note:** You don't need to include "Session" or "CourseManagement" class in your UML diagram.)

- Each role has their own sets of features they can access from the program. For example, a student can do three things, e.g. add or remove a course, view course list. After login, a function has to be called to view all of the actions and perform the action specific user requests from the system. Implement this by using the following interface with the appropriate class/classes:

```
interface Action{
   public void handleActions();
}
```

- **void addCourse(Student, Course), void removeCourse (Student, Course), void viewCourse/ Course[ ] viewCourse:** Invoke them according to the option selected by the user. They should all be static methods either in your execution class. Notice that viewCourse function is common for student, teacher and TA, however, it has different definitions. (Hint: You may achieve this by implementing polymorphism to the method argument, or better option, through method overloading)
- **void addStudent(Student), void removeStudent(Student), void viewStudentList()/ Student[ ] viewStudentList():** These functions should be present in "Course" class and must be utilized appropriately. The student capacity for a course can be 5.

- Address a few error cases. For example, making sure a student isn't being enrolled into different sections of the same course. Also, be careful about invoking static methods and make sure to avoid IndexOutofBoundException and NullPointerException.
- Put the entire program in an infinite loop and place option at every stage to go back to its previous stage. (Hint: place every stage inside a different function so that you may exit and enter the function as needed. Sometimes, recursion may work better than a loop. Note that, void function can also use return.)
- Use meaningful identifiers for methods and variables
- Use comments where necessary
- Avoid repetitive code or adding anything unnecessary. Bonus 5 marks for neat and concise code, and sticking to the guidelines.
- Your test data (database) should be as follows:

Email	Password	Student ID	Student Name	CGPA
student_a@northsouth.edu	password	S1	A	3.4
student_b@northsouth.edu	password	S2	B	3.6
student_c@northsouth.edu	password	S3	C	3.2

Email	Password	Teacher ID	Teacher Name
teacher_a@northsouth.edu	password	T1	A
teacher_b@northsouth.edu	password	T2	B

Email	Password	Assistant ID	Assistant Name
assistant_a@northsouth.edu	password	A1	A
assistant_b@northsouth.edu	password	A2	B

Index	Course Name	Section	Timing	Teacher	TA
1	Course A	1	A	T1	A
2	Course A	2	A	T2	A
3	Course A	3	B	T1	B
4	Course B	1	A	T1	A
5	Course C	1	B	T2	A
6	Course C	2	C	T1	B

- Sample input and output for Teacher:

```
1: View Courses   2: Logout
> 1
1: Course A.1   2: Course A.2   3:Course B.1    4:Back
>1
List of Student:
1. …
2. …
3. …
1: Remove Student   2:Back
> 1
Index of Student to be removed: > 1
Student "..." Removed From Your Course A.1 Successfully!
List of Student:
1.
2.
1: Remove Student    2:Back
>2
1: View Courses   2: Logout
>
```

- TA should also use the above program flow, but won't have the "Remove Student" option.
- For Student, follow this sample:

```
1: Add Course   2: View Courses   3:Back
>1
Press 0 to go Back.
Index of the Course to be added: > 4
You have been enrolled in "Course B.1" Successfully!
Press 0 to go Back.
Index of Course to be added: 0
1: Add Course   2:View Courses   3:Back
>2
1: Course B
1:Remove Course  2: Back
>1
Index of Course to be removed: > 1
You have dropped "Course B" Successfully!
No courses to show.
1: Back
>1
1: Add Course   2: View Courses   3:Back
>
```

c. Explain how and why the concept of inheritance, abstraction and polymorphism has been implemented within the program. [5]