Programming Assignment 3

Submission Date : 23.11.2023
Due Date : 14.12.2023 (23:00)
Subject : Inheritance and Polimorphism
Advisors : R.A. Bahar GEZİCİ

1 Academic Course Management System

In this experiment, you have been tasked with developing an Academic Course Management System (ACMS). Your objective is to implement an APMS while applying inheritance and polymorphism concepts in object-oriented programming (OOP).

Inheritance is one of the core concepts of object-oriented programming (OOP). It is a mechanism where you can to derive a class from another class for a hierarchy of classes that share a set of attributes and methods. Implement a hierarchy of classes that represent various roles within the academic program system, sharing common attributes and methods.

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in object oriented design occurs when a parent class reference is used to refer to a child class object. Utilize polymorphism to enable different classes to exhibit unique behaviors through shared methods.

It is expected you to implement the program by using inheritence and polimorphism. Implementing the program without using the benefits of these concepts will be **penalized.**

Figure 1: Academic Course Management System

2 Problem

In this experiment, your task is to implement an Academic Course Management System (ACMS). The primary objective is to introduce object-oriented programming and design in Java, emphasizing class structure, interaction, inheritance, polymorphism, and basic input-output operations. There will be two main parts to this assignment.

- One part consists of creating the system's input-output interface and implementing the logic for this interaction. You should also use decorator design pattern in this part of the experiment.
- The other part is defining (and implementing) the persistence backend (i.e. how we store and retrieve the information our system uses).

Your program will take commands from an input file then it will print the results of these commands to an output file. You should use Data Access Objects (DAO) to manage your data.

2.1 Decorator Pattern

The Java language provides the keyword *extends* for subclassing a class. By extending a class, you can change its behavior. The Decorator Pattern is used for adding additional functionality to a particular object as opposed to a class of objects. With the Decorator Pattern, you can add functionality to a single object and leave others like it unmodified. Any calls that the decorator gets, it relays to the object that it contains, and adds its own functionality along the way, either before or after the call. This gives you a lot of flexibility since you can change what the decorator does at runtime, as opposed to having the change to be static and determined at compile time by subclassing. Since a Decorator complies with the interface that the object that it contains, the Decorator is indistinguishable from the object that it contains. That is, a Decorator is a concrete instance of the abstract class, and thus is indistinguishable from any other concrete instance, including other decorators. This can be used to great advantage, as you can recursively nest decorators without any other objects being able to tell the difference, allowing a near infinite amount of customization. Decorators add the ability to dynamically alter the behavior of an object because a decorator can be added or removed from an object without the client realizing that anything changed. It is a good idea to use a Decorator in a situation where you want to change the behavior of an object repeatedly (by adding and subtracting functionality) during runtime.

The dynamic behavior modification capability also means that decorators are useful for adapting objects to new situations without re-writing the original object's code.

2.2 Data Access Object

The Data Access Object (DAO) layer is an essential part of good application architecture. Business applications almost always need access to data from databases or data files. The Data Access Object design pattern provides a technique for separating object persistence (stored data) and data access logic from any particular persistence mechanism (writing to files or accessing a real database). There are clear benefits to this approach from an architectural perspective. The Data Access Object approach provides flexibility to change an

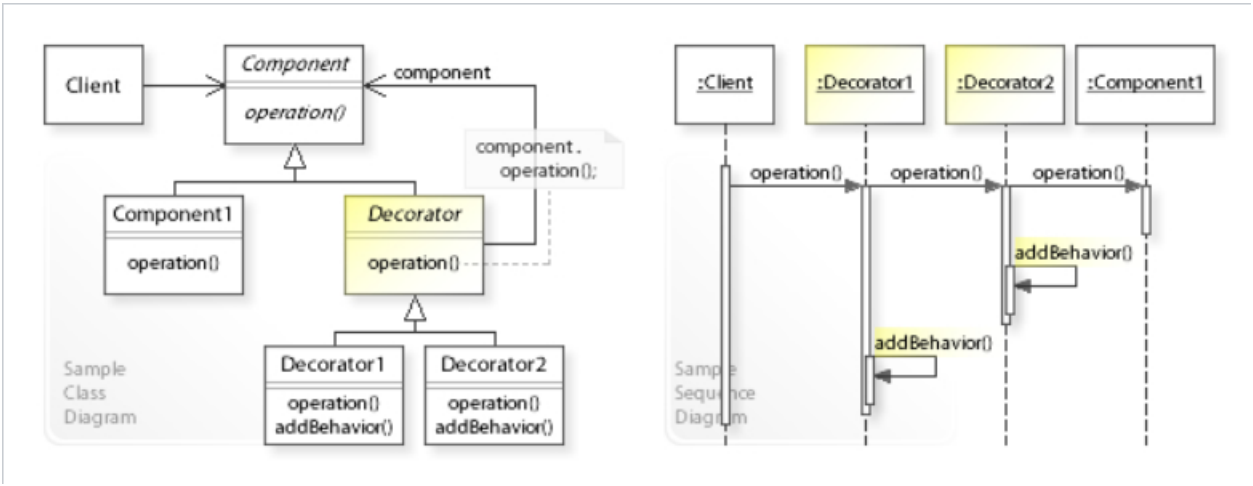

Figure 2: A sample UML class and sequence diagram for the Decorator design pattern

application's persistence mechanism over time without the need to re-engineer application logic that interacts with the Data Access Object tier. The Data Access Object design pattern also provides a simple, consistent API for data access that does not require knowledge of sub mechanism. A typical Data Access Object class diagram is shown below. (This is an example, your interface should look like different).

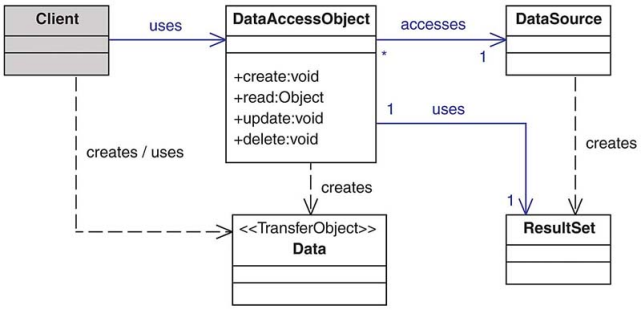

Figure 3: A sample UML class diagram

3 Experiment Details

In this experiment, you are expected to develop a simple Academic Course Management System. Your application should support the following features:

- **New Student:** Create a new student record. Attributes of a student are:
  - student id (Unique)
  - student surname
  - student name

  - student address
  - student phone number

- **Remove Student:** Remove a student from the database using their identification number.
- **List Students:**

  Print the list of students ordered by name.

- **Create Course Enrollment:**

  Create a new enrollment for an existing student and append it to enrollment data file. Each enrollment has a unique enrollment id.

- **Add Assessment:** Add an assessment to a given order with given types. Add an assessment to a specific course for a student. Assignments can be of different types, including: There are two types of assessments:
  - Essay-based Assessment 10 credits : Assigned for in-depth evaluation.
  - Multiple Choice Assessment 15 credits : A shorter form of assessment

  Each assessment can have up to three additional components or tasks, which vary based on the type of assessment:
  - Literature Review: 15 credits
  - Analysis: 10 credits
  - Question Set: 7 credits
  - Additional Tasks: 5 credits

  The idea here is to simulate different types of assessments within an academic environment, with each assessment type having specific components or tasks associated with it. This allows for the incorporation of various operations within the context of an educational examination, illustrating the concept of inheritance and polymorphism in managing diverse assessment types within a course.

- **Calculate Total Fees**

  In the Academic Program System, calculate the total fees for a student's course enrollment. List all assessments in the order then calculate the total fee of an assignment.
  - Total Course Fees Calculation: Calculate the total fees for a student's course enrollment based on the assessments and additional operations involved. Each assessment and its associated operations (additional tasks) contribute to the overall fees. The total cost is the sum of the course fee, assessment fees, and the fees for additional tasks.

Student information and course enrollment information will be stored in text files. The format of the files for persistence are shown below. You can reach sample student file, course enrollment data file and sample input-output files from the page of piazza site of the experiment.

3.1 Student Data File Format:

< **studentID** > *tab* < **studentName** > *space* < **studentSurname** > *tab* < **phone number** > *tab* < **address** > *newline*

Student name and surname consist of one string. The address can be made up of one or more strings. Entries in student data file should be ordered by ID. After add and remove operations, all changes should be reflected to data files (student.txt ve courseEnrollment.txt). In Figure 4, you can see an example of student.txt file.

```
5→Harun Gezici→536-8967812→Address: Tandogan 34 sk. 56/3
9→Ebru Yilmaz→312-4567890→Address: Maltepe 21 sk. 23/7
12→Orhan Seven→256-3456789→Address: Eryaman 3. Etap 12/7
23→Zeynep Atac→234-3456789→Address: Hacettepe University Department of Matemathics
35→Umit Gezen→236-3446789→Address: Dikmen 4. Cadde 34/2
42→Ismail Can→535-8969746→Address: Cebeci 56 cd. 34/5
45→Arif Batar→563-9563256→Address: Balgat 1321 cd. 12/5
46→Can Sever→856-4581256→Address: Kizilay 3 cd. 21/21
50→Meryem Atas→654-2122121→Address: Emek 19 sk. 12/5
```

Figure 4: A sample student.txt file

3.2 Course Enrollment Data File Format:

< **enrollmentID** > *tab* < **studentID** > *newline*
< **assessment type**($Essay - basedor MultipleChoice$) > *tab* < **tasks [1-4]** >

Entries in course enrollment data file should be ordered by ID. After add and remove tasks, all changes should be reflected data files. In Figure 5, you can see a sample courseEnrollment.txt file.

```
1→9
MultipleChoice→Analysis AdditionalTasks QuestionSet
Essaybased→Analysis LiteratureReview QuestionSet
2→35
Essaybased→Analysis
Essaybased→QuestionSet LiteratureReview
MultipleChoice→LiteratureReview
3→45
MultipleChoice→LiteratureReview
MultipleChoice→LiteratureReview
4→46
Essaybased→QuestionSet
```

Figure 5: A sample courseEnrollment.txt file

3.3 Input File Format

(Note: Number of tasks can vary from 1 to 3.)

- **AddStudent** space < **studentID** > *space* < **name** > *space* < **surname** >< **phone number** > *space* < **address** >
- **RemoveStudent** space < **student ID** >
- **CreateEnrollment** space < **EnrollmentID** > *space* < **StudentID** >
- **AddAssessment** space < **EnrollmentID** > *space* < **assessment type** > *space* < **task** >
- **TotalFee** space < **EnrollmentID** >
- **ListStudents**

```
AddStudent 33 Hakan Kanat 234-5677896 Ayranci 10. sok. 30/10
AddStudent 88 Beril Maral 568-8964142 Bilkent 32 cd. 72/3
RemoveStudent 46
CreateEnrollment 7 50
AddAssessment 7 Essaybased QuestionSet LiteratureReview
AddAssessment 7 MultipleChoice QuestionSet AdditionalTasks
TotalFee 7
ListStudents
```

Figure 6: A sample input.txt file

3.4 Output File Format

- Student < $StudentID$ >< $name$ > $added$
- Student < $StudentID$ >< $name$ > $removed$
- Course Enrollment < $EnrollmentID$ > created
- < $Assessmenttype$ > assessment added to enrollment < $EnrollmentID$ >
- Total cost for enrollment < $EnrollmentID$ >
  - < $assessmenttype$ >< $task[1-4]$ >< fee >
  - < $assessmenttype$ >< $task[1-4]$ >< fee >
  - Total: < $total\_fee$ >

4 Implementation Details

You have to use decorator pattern (20points) for adding operations mechanism. Add operation code should look like this as shown in Figure 8:

assessment.addtask(new Analysis(new QuestionSet(new AdditionalTasks())); This an assessment with Analysis, QuestionSet, and AdditionalTasks.

assessment.printtasks(); Output of this method call should be AdditionalTasks, QuestionSet and Analysis

```
Student 33 Hakan added
Student 88 Beril added
Student 46 Can removed
CourseEnrollment 7 created
Essaybased assessment added to enrollment 7
MultipleChoice assessment added to enrollment 7
TotalFee for enrollment 7
	Essaybased QuestionSet LiteratureReview 32$
	MultipleChoice QuestionSet AdditionalTasks 27$
	Total: 59$
Student List:
45 Arif Batar 563-95632569 Address: Balgat 1321 cd. 12/5
88 Beril Maral 568-8964142 Address: Bilkent 32 cd. 72/3
9 Ebru Yilmaz 312-4567890 Address: Maltepe 21 sk. 23/7
33 Hakan Kanat 234-5677896 Address: Ayranci 10. sok. 30/10
5 Harun Gezici 536-8967812 Address: Tandogan 34 sk. 56/3
42 Ismail Can 535-8969746 Address: Cebeci 56 cd. 34/5
50 Meryem Atas 654-2122121 Address: Emek 19 sk. 12/5
12 Orhan Seven 256-3456789 Address: Eryaman 3. Etap 12/7
35 Umit Gezen 236-3446789 Address: Dikmen 4. Cadde 34/2
23 Zeynep Atac 234-3456789 Address: Hacettepe University Department of Matemathics
```

Figure 7: A sample output.txt file

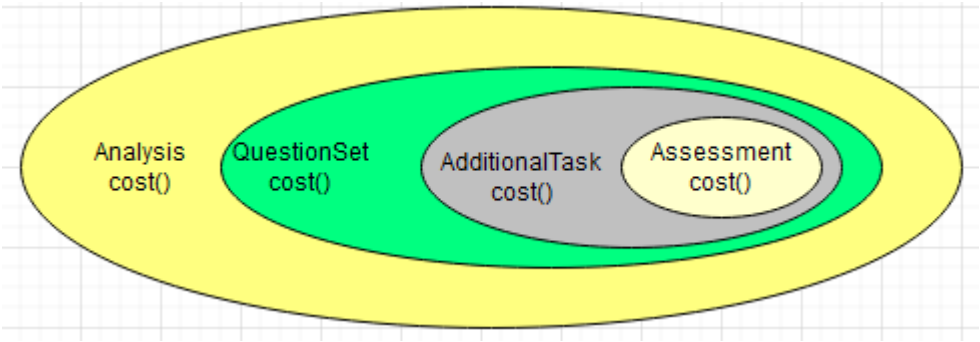

Figure 8: A sample decorator pattern representation

assessment.fee(); Output of this method call should return the fees of assessment and its tasks

For our system, we will store the data inside the individual files (one for enrollment and one for student class), and use Data Access Objects (DAO) to manage them (10points). Each DAO will have the basic operations to manage the data (defined by the DAO interface provided with the assignment). So basically what you need to do is implementing following abstract methods:

Object getByID(int ID) // read a single entry from the file

Object deleteByID(int ID) // delete a single entry from file

void add(Object object) // add or update an entry

ArrayList getALL() // get all entries

You could add additional methods to your DAOs and you can also make changes in DAO interface. Your application should terminate and save the files automatically when finish to read the input file.

5 Execution and Test

The input file (input.txt) is going to be given as program arguments. There are text files (student.txt, courseEnrollment.txt) in your working directory and when your program starts, you will read these files, then update them according to the input file. In order to test your program, you should follow the following steps:

- Upload your java files to your server account (dev.cs.hacettepe.edu.tr)
- Compile your code (javac *.java)
- Run your program (java Main input.txt)
- Control your output data (output.txt).

6 Submit Format

- File hierarchy must be zipped before submitted (Not .rar, only .zip files are supported by the system)

- $\langle studentid \rangle.zip$

    - src (Main.java, *.java)

7 Grading Policy

Task	**Point**
Compiled	10
Decorator Pattern	20
Data Access Object	10
Output	60
Total	100

8 Notes

- The assignment must be original, individual work. Downloaded or modified source codes will be considered as cheating. Also the students who share their works will be punished in the same way.
- We will be using the Measure of Software Similarity (MOSS) to identify cases of possible plagiarism. Our department takes the act of plagiarism very seriously. Those caught plagiarizing (both originators and copiers) will be sanctioned.

- You can ask your questions through course's piazza group and you are supposed to be aware of everything discussed in the piazza group. General discussion of the problem is allowed, but DO NOT SHARE answers, algorithms, source codes and reports .
- With this assignment which covers the subjects of classes, objects, polimorphism, and inheritance; you are expected to gain practice on the basics of object oriented programming (OOP). Therefore, you will not be graded if any paradigm other than OOP (i.e., structured programming) is developed!
- Don't forget to write comments of your codes when necessary.
- The names of classes', attributes' and methods' should obey to Java naming convention.
- Save all work until the assignment is graded.
- Do not miss the deadline. Submission will be end at 14/12/2023 23:00, the system will be open 23:59:59. The problem about submission after 23:00 will not be considered.
- There will be again 2 days extensions (each day degraded by 10 points) in this project.

9 References

1. *https://en.wikipedia.org/wiki/Data_access_object*
2. *https://en.wikipedia.org/wiki/Decorator_pattern*