Programming Assignment 3

Submission Date : 01.05.2020
Due Date : 22.05.2020 (23:59)
Subject : Inheritance and Polimorphism
Advisors : R.A. Bahar GEZİCİ, Nebi YILMAZ

1 Introduction

In this experiment, you have been commissioned to develop a hospital management system (HMS). You are expected to implement HMS with the given rule. The main focus of this experiment is to get you familiar with inheritence and polymorphism concepts in object oriented (OO) programming.

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.

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.

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: Hospital Management System

2 Problem

In this experiment you are expected to implement a hospital management system. The aim of this experiment is to introduce you object-oriented programming and design with Java. You will learn the structure of a class, how classes interact, inheritance and polymorphism and basic input-output operations in Java. 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.

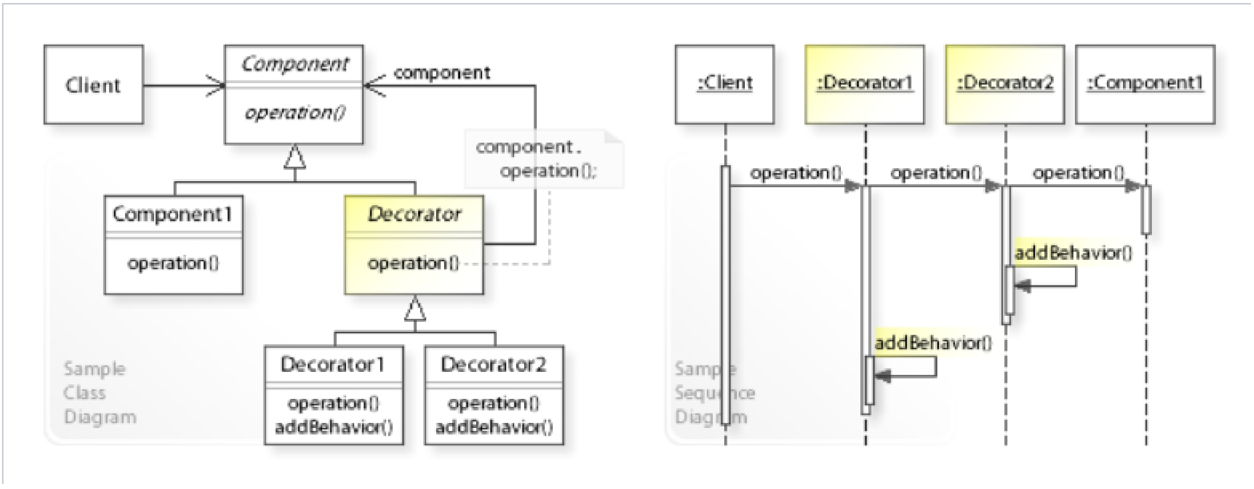


Figure 2: A sample UML class and sequence diagram for the Decorator design pattern

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 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).

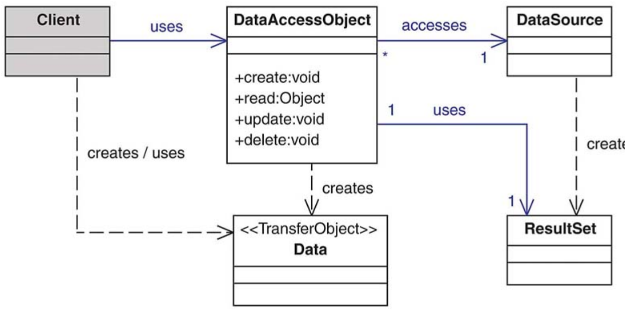


Figure 3: A sample UML class diagram

3 Experiment Details

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

- **New Patient:** Create a new patient. Attributes of a patient are:
  - patient id (Unique)
  - patient surname
  - patient name
  - patient address
  - patient phone number
- **Remove Patient:**

  Remove a patient from the database by using its identification number.
- **List Patients:**

  Print the list of patients ordered by name.
- **Create Admission:**

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

- **Add Examination:** Add an examination to a given order with given operations. An examination can include 3 operations at most. There are 2 types of examination. These are:
  - Inpatient examination 10 $
  - Outpatient examination 15 $

  An examination can include 3 additional operations at most. There are 4 types of operations:
  - doctor visit 15 $
  - imaging 10 $
  - tests 7 $
  - measurements 5 $
- **Total Cost**

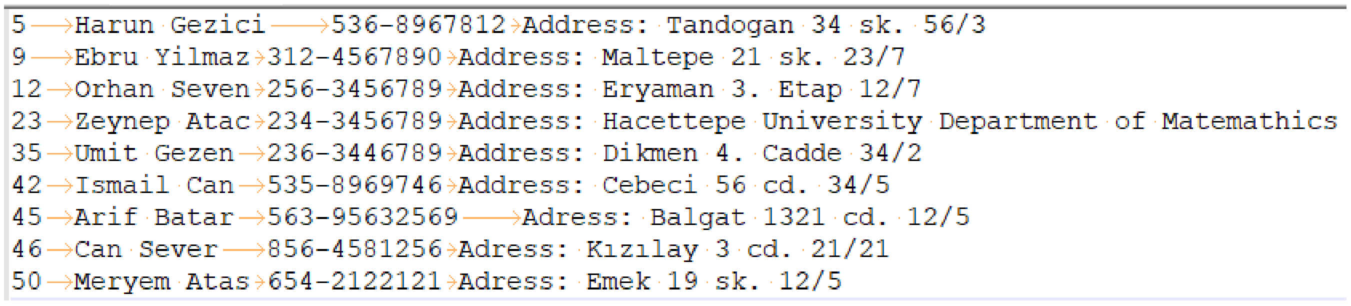
  Calculate the total cost of an admission. List all examinations in the order then calculate the total cost of an admission.

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

3.1 Patient Data File Format:

< **patientID** > *tab* < **patientName** > *space* < **patientSurname** > *tab* < **phone number** > *tab* < **address** > *newline*

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



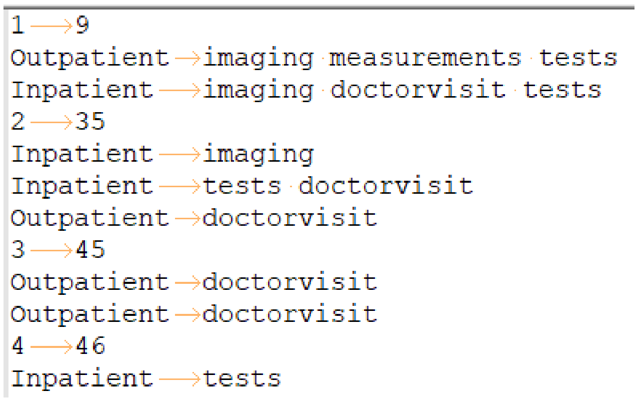
```
5	Harun Gezici	536-8967812	Address: Tandogan 34 sk. 56/3
9	Ebru Yilmaz	312-4567890	Address: Maltepe 21 sk. 23/7
12	Orhan Seven	256-3456789	Address: Eryaman 3. Etap 12/7
23	Zeynep Atac	234-3456789	Address: Hacettepe University Department of Matemathics
35	Umit Gezen	236-3446789	Address: Dikmen 4. Cadde 34/2
42	Ismail Can	535-8969746	Address: Cebeci 56 cd. 34/5
45	Arif Batar	563-95632569	Adress: Balgat 1321 cd. 12/5
46	Can Sever	856-4581256	Adress: Kızılay 3 cd. 21/21
50	Meryem Atas	654-2122121	Adress: Emek 19 sk. 12/5
```

Figure 4: A sample patient.txt file

3.2 Admission Data File Format:

< **admissionID** > *tab* < **patientID** > *newline*
< **examination type**(*InpatientorOutpatient*) > *tab* < **operation** [**1-4**] >

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



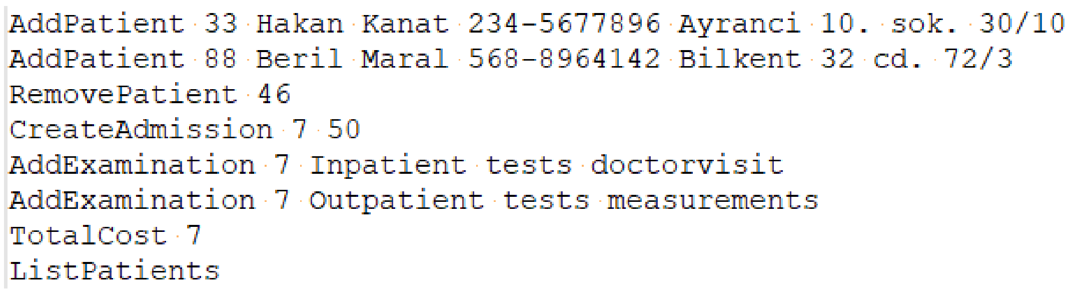
```
1	9
Outpatient	imaging measurements tests
Inpatient	imaging doctorvisit tests
2	35
Inpatient	imaging
Inpatient	tests doctorvisit
Outpatient	doctorvisit
3	45
Outpatient	doctorvisit
Outpatient	doctorvisit
4	46
Inpatient	tests
```

Figure 5: A sample admission.txt file

3.3 Input File Format

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

- **AddPatient** space < **patientID** > *space* < **name** > *space* < **surname** >< **phone number** > *space* < **address** >
- **RemovePatient** space < **patient ID** >
- **CreateAdmission** space < **AdmissionID** > *space* < **PatientID** >
- **AddExamination** space < **AdmissionID** > *space* < **examination type** > *space* < **operation** >
- **TotalCost** space < **AdmissionID** >
- **ListPatients**

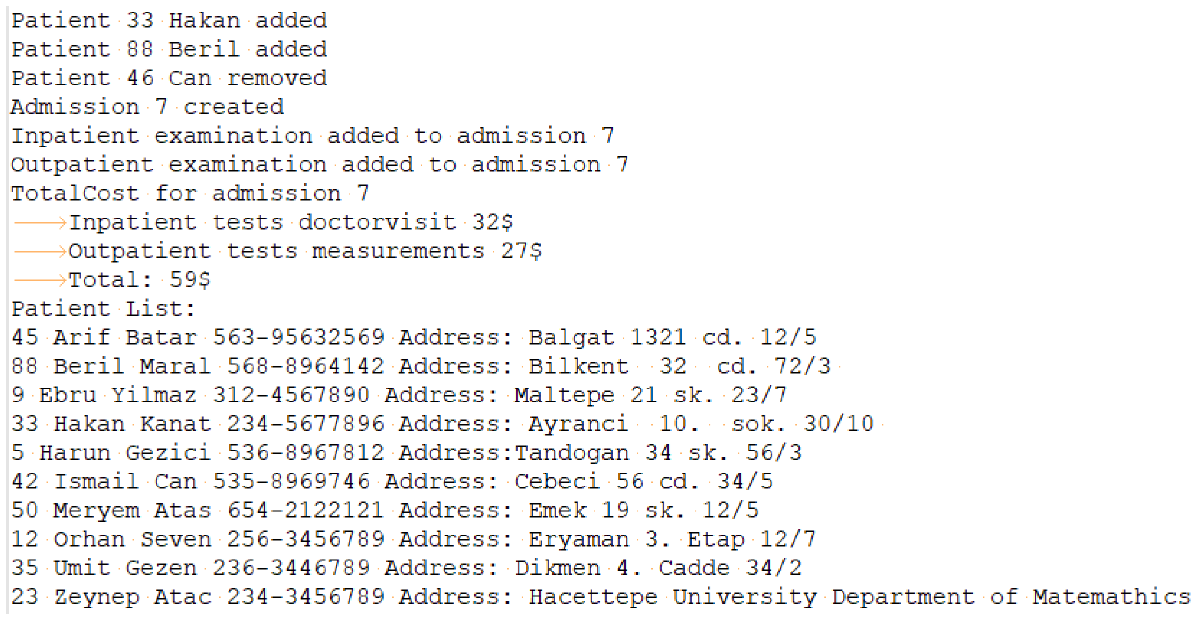


```
AddPatient 33 Hakan Kanat 234-5677896 Ayranci 10. sok. 30/10
AddPatient 88 Beril Maral 568-8964142 Bilkent 32 cd. 72/3
RemovePatient 46
CreateAdmission 7 50
AddExamination 7 Inpatient tests doctorvisit
AddExamination 7 Outpatient tests measurements
TotalCost 7
ListPatients
```

Figure 6: A sample input.txt file

3.4 Output File Format

- Patient $< PatientID >< name > added$
- Patient $< PatientID >< name > removed$
- Admission $< AdmissionID >$ created
- $< Examinationtype >$ examination added to admission $< AdmissionID >$
- Total cost for admission $< AdmissionID >$
  - $< examinationtype >< operation[1-4] >< cost >$
  - $< examinationtype >< operation[1-4] >< cost >$
  - Total: $< total\_cost >$



```
Patient 33 Hakan added
Patient 88 Beril added
Patient 46 Can removed
Admission 7 created
Inpatient examination added to admission 7
Outpatient examination added to admission 7
TotalCost for admission 7
	Inpatient tests doctorvisit 32$
	Outpatient tests measurements 27$
	Total: 59$
Patient 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

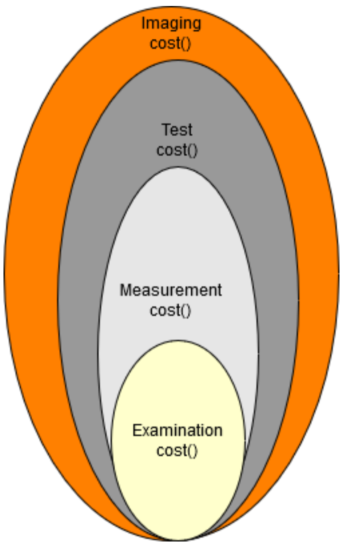
4 Implementation Details

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

examination.addoperation(new imaging(new test(new measurement()))); This an examination with imaging, test, and measurement.

examination.printoperations(); Output of this method call should be measurement, test and imaging

examination.cost(); Output of this method call should return the costs of examination and its operations



For our system, we will store the data inside the individual files (one for admission and one for patient 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 (patient.txt, admission.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)

- ⟨*studentid*⟩*.zip*

    - src (Main.java, *.java)

    - Report.pdf

7 Grading Policy

Task	**Point**
Compiled	10
Decorator Pattern	20
Data Access Object	10
Output	50
Report	10
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!
- You are responsible for a correct model design. Your design should be accurate. It is important to **draw a class diagram** to show the whole of the system that you have created.
- **In your REPORT,** it is important giving explanation about problem definition (limited to max. 3 sentences) and steps of algorithms that you followed (limited to max. 5 sentences). Also, you must add class diagram of your code in the report. Do not give so much unnecessary details in your report (totally max 2 pages written with Times New Roman 12).

- 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 22/05/2020 23:59, the system will be open 23:59:59. The problem about submission after 23:59 will not be considered.
- There will be again 3 days extensions (each day degraded by 10 points) in this project.

9 References

1. *https://en.wikipedia.org/wiki/Data*$_a$*ccess*$_o$*bject*
2. *https://en.wikipedia.org/wiki/Decorator_pattern*