Hacettepe University
Computer Science and Engineering Department

Name and Surname : Murat Celik
Identity Number : 21827263
Course : BBM104
Experiment : Assignment 3
Subject : Inheritance and Polimorphism
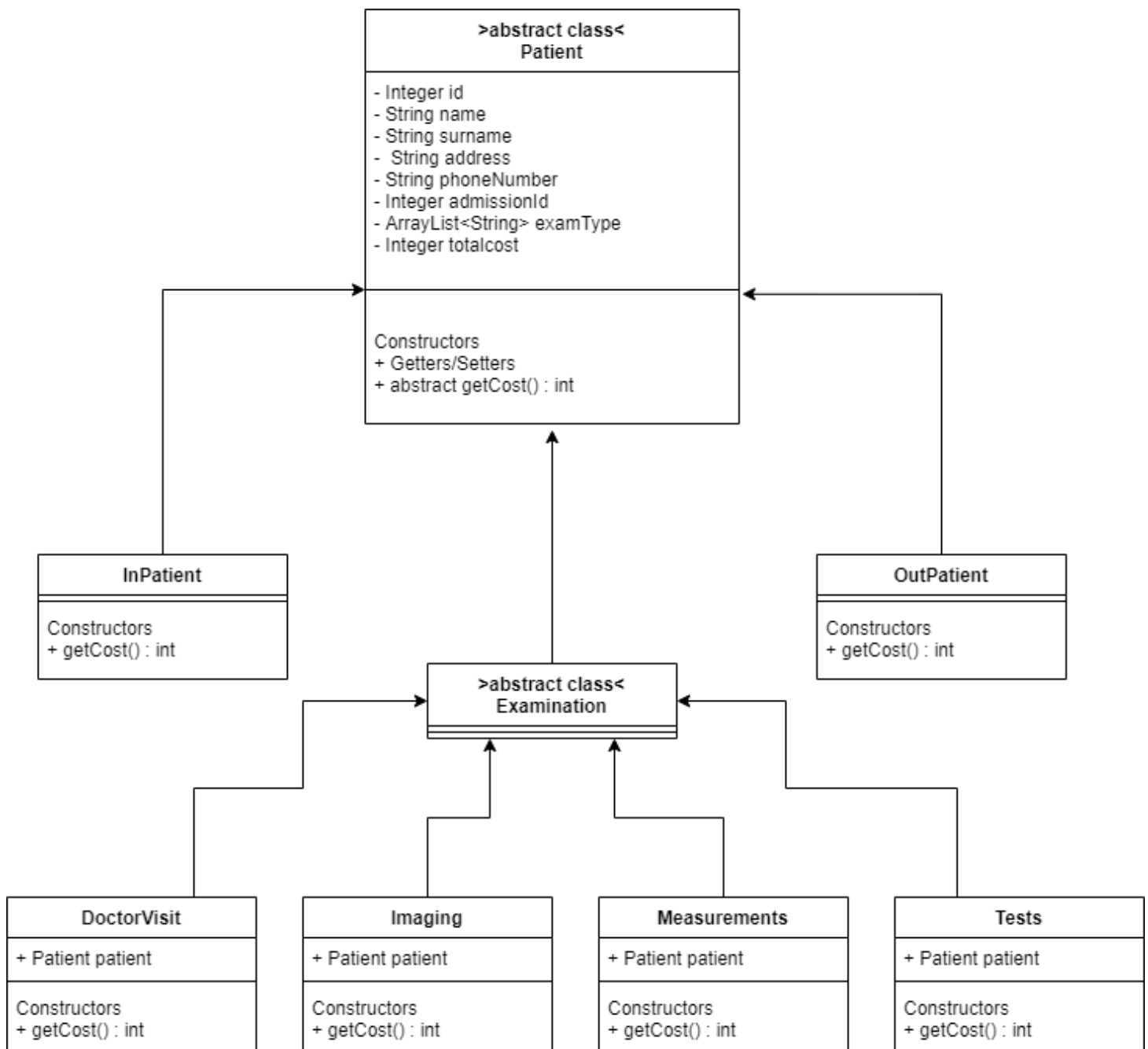**Data Due :** 22.05.2020 (23:59)

Problem

Develop a simple Hospital Management System.

- New Patient
- Remove Patient
- List Patients
- Create Admission
- Add Examination
- Total Cost

Decorator pattern

Decorator pattern allows a user to add new functionality to an existing object without altering its structure.

>abstract class<
Patient

- Integer id
- String name
- String surname
- String address
- String phoneNumber
- Integer admissionId
- ArrayList<String> examType
- Integer totalcost

Constructors
+ Getters/Setters
+ abstract getCost() : int

InPatient

Constructors
+ getCost() : int

OutPatient

Constructors
+ getCost() : int

>abstract class<
Examination

DoctorVisit

+ Patient patient

Constructors
+ getCost() : int

Imaging

+ Patient patient

Constructors
+ getCost() : int

Measurements

+ Patient patient

Constructors
+ getCost() : int

Tests

+ Patient patient

Constructors
+ getCost() : int

There are 2 types of examination. These are: a) Inpatient b) Outpatient

There are 4 types of operations: 1) doctor visit 2) imaging 3) tests 4) measurements

We create object from an examination type according to patient. If patient has operations, we add the object in a operations objects . So that, the object has a new functionality.
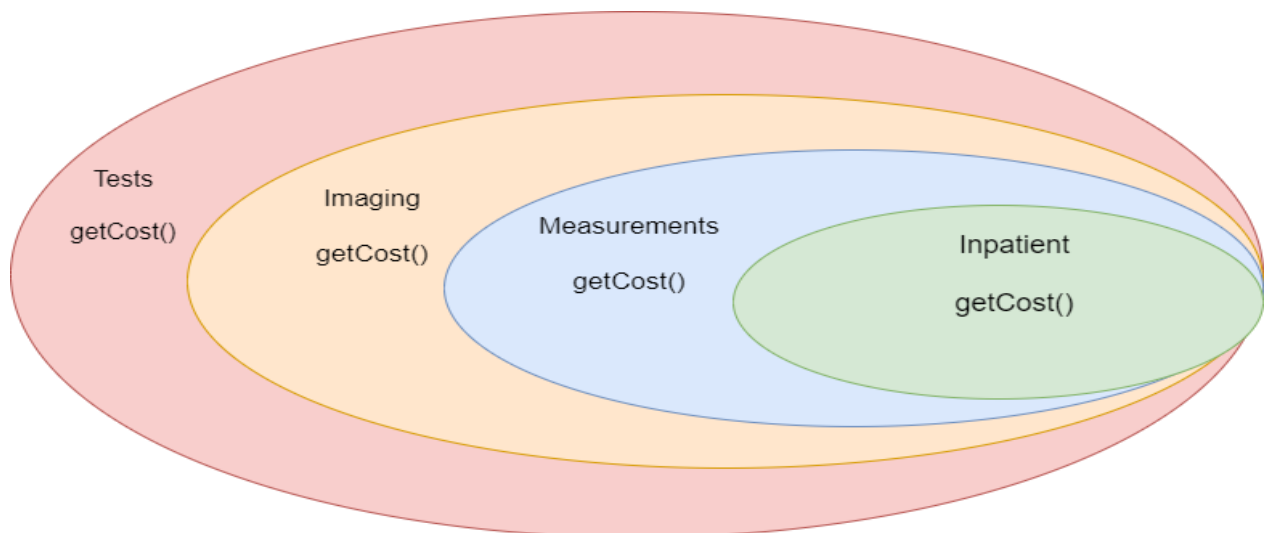
Example//

Patient has ➔ Inpatient   imaging,test,measurement

My Code :

Patient examination = new InPatient();

examination = new Measurements(examination);

examination = new Tests(examination);

examination = new Imaging(examination);

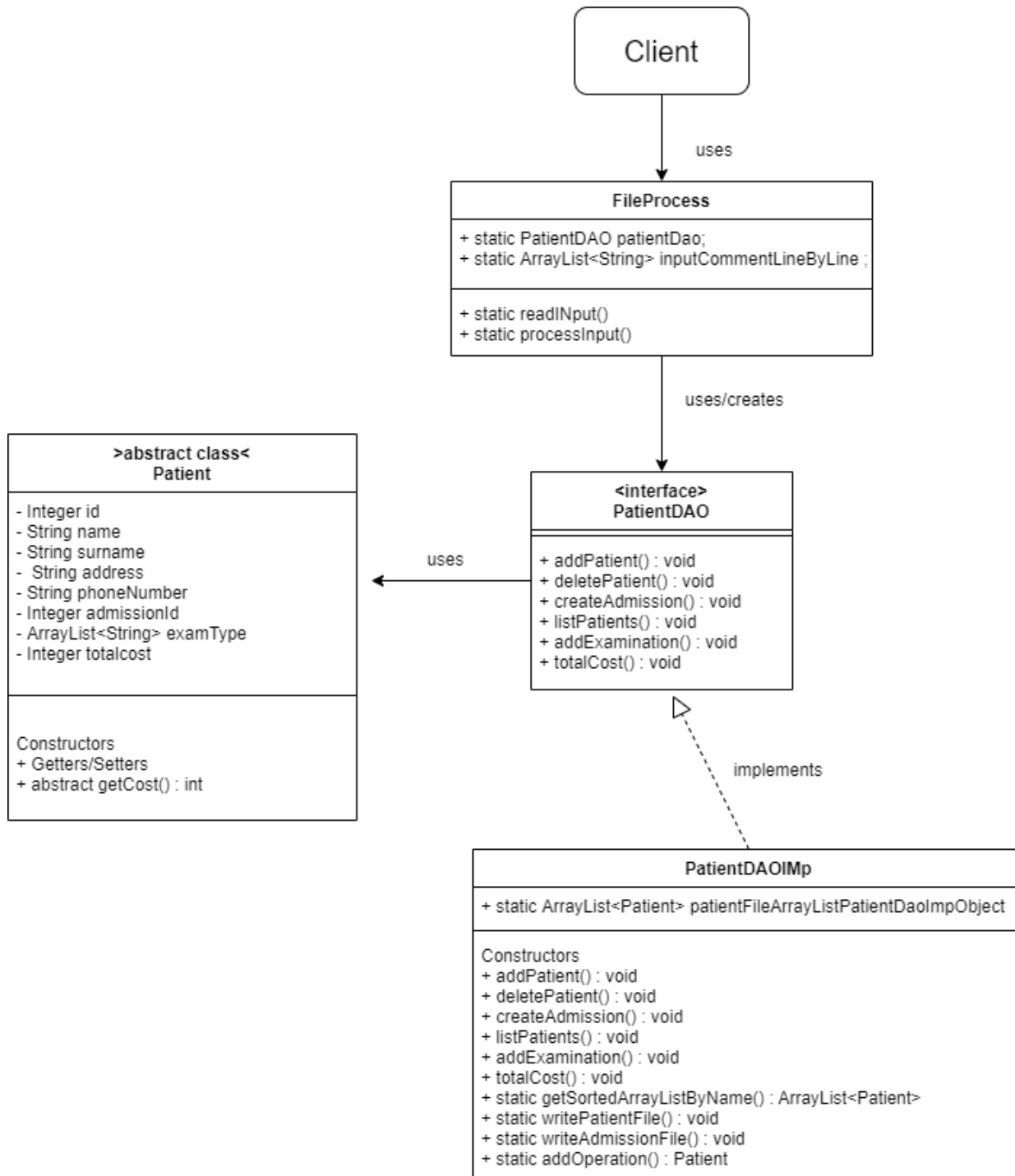If we call examination.getCost(), we obtain 32$

– Inpatient examination 10 $

– imaging 10 $

 – tests 7 $

– measurements 5 $

Data Access Object

The Data Access Object (DAO) pattern is a structural pattern that allows us to isolate the application/business layer from the persistence layer (usually a relational database, but it could be any other persistence mechanism) using an abstract API.

Client

uses

FileProcess

+ static PatientDAO patientDao;
+ static ArrayList<String> inputCommentLineByLine ;

+ static readINput()
+ static processInput()

uses/creates

>abstract class<
Patient

- Integer id
- String name
- String surname
- String address
- String phoneNumber
- Integer admissionId
- ArrayList<String> examType
- Integer totalcost

Constructors
+ Getters/Setters
+ abstract getCost() : int

uses

<interface>
PatientDAO

+ addPatient() : void
+ deletePatient() : void
+ createAdmission() : void
+ listPatients() : void
+ addExamination() : void
+ totalCost() : void

implements

PatientDAOIMp

+ static ArrayList<Patient> patientFileArrayListPatientDaoImpObject

Constructors
+ addPatient() : void
+ deletePatient() : void
+ createAdmission() : void
+ listPatients() : void
+ addExamination() : void
+ totalCost() : void
+ static getSortedArrayListByName() : ArrayList<Patient>
+ static writePatientFile() : void
+ static writeAdmissionFile() : void
+ static addOperation() : Patient

We have collected the orders that the client can give into the PatientDAO interface. We have written the contents of these processes in PatientDAOIMp. Thus, we concealed the backside from the user and we reduced the error rate of the user. We were able to easily use the Patient object in PatientDAOIMp. And we were able to process our data safely.

Conclusion

I learned how to use Decorator Pattern and Data Access Object Pattern. I practiced for algorithm again and again. I had the opportunity to think about how to use the exchange of information between files. I think my code would be more efficient if ı use SQL database.