Assignment 1

Release Date: Week 4, 29/10/2024
Due Date: Week 8, 29/11/2024, 11 PM
Value: 15%
Assessment Mode: Individual Assignment

Rationale

This assignment has been designed to allow students to test and demonstrate their ability to write a Java program that uses a range of different concepts and facilities. This assessment relates to the following learning outcomes:

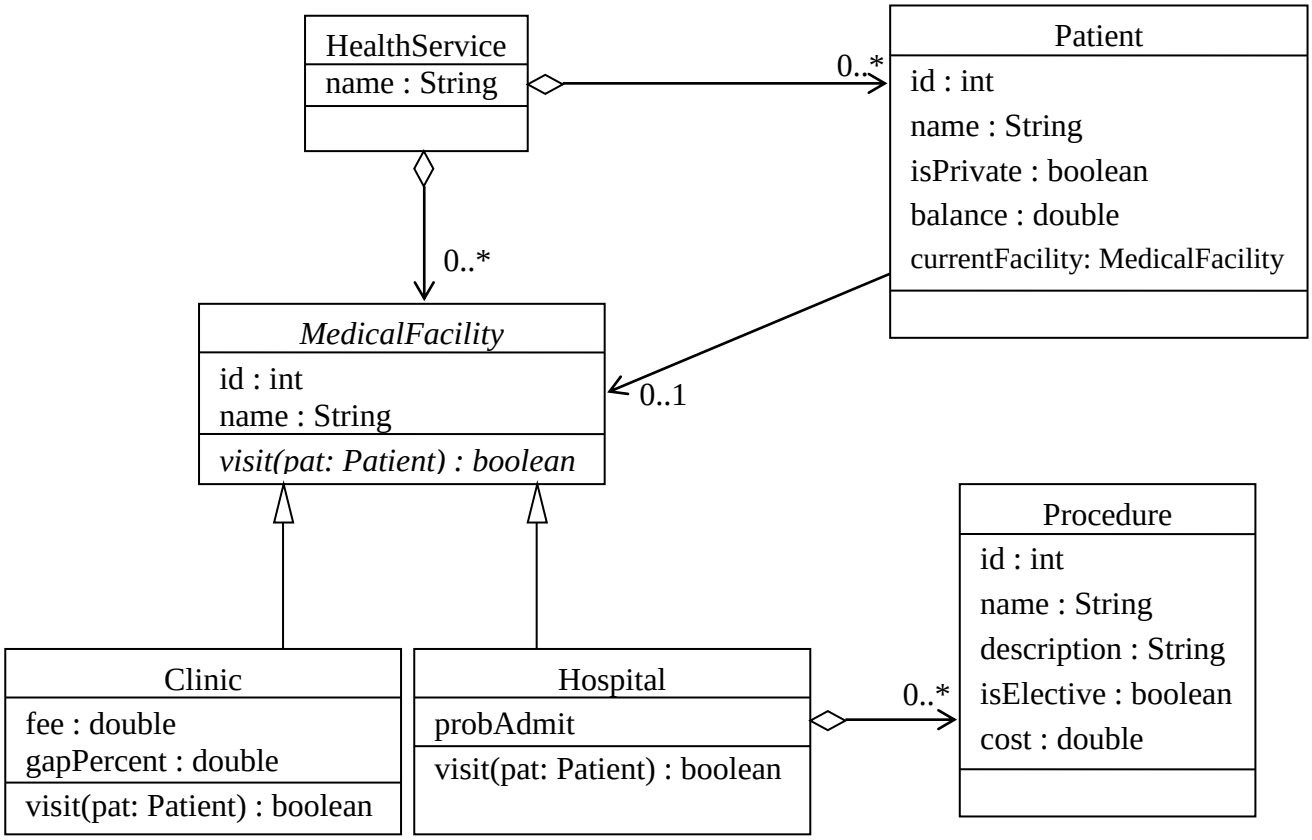
- design and write programs using several classes, including problem domain classes; and
- understand and implement object-oriented principles and key concepts in modern software design such as inheritance, association, encapsulation, and reusable software.

In particular, this assignment tests the students' ability to use appropriate class hierarchies and collection classes.

Instructions

You are required to implement in Java all the classes represented in a given conceptual model according the specifications provided. You are also required to develop a separate user interface which, when combined with the problem domain classes forms an executable application.

The conceptual model for use in this assignment is shown below as a class diagram rendered using UML. It depicts a model for a health service. The model is not intended to reflect actual practices within any existent health service.



The **health service** class is the anchor point for the model and it contains a collection of **patients** and a collection of generalised **medical facilities**. A specific medical facility can be either a **hospital** or a **clinic**. Each hospital contains a collection of **procedures**.

Patients **visit** medical facilities, however a visit to a hospital results in different functionality to a visit to a clinic. In a hospital a visiting patient is assessed and may or may not then be **admitted**. Procedures may be **performed** on admitted patients. In a clinic a visiting patient is **registered** on the first visit and can receive **consultations** on subsequent visits.

Specifications are provided below for each individual class in the diagram. Students are expected to implement the classes correctly according to these specifications although some of the programmatic techniques for doing so are left for individual choice. For example, extra attributes or methods may be added to any class as needed for sensible storage and functionality.

HealthService

This is the anchor class for the model and only one object will be expected to exist for the application. The object will have a `name` and will maintain and manage a collection of patients and a collection of medical facilities.

Patient

Each patient object has an `id` attribute, a read-only, automatically generated, unique sequential identifier, and a `name`. The `isPrivate` attribute indicates whether the patient has private or public status and this affects how they are charged when they visit clinics or undergo procedures in hospitals. The `balance` attribute holds the total cost of the patient's health needs and is increased as the patient incurs costs. It is initially zero. The `currentFacility` attribute can hold a reference to a medical facility. It is a constraint of the application that a patient can be associated with at most one facility at any given time and may be associated with no facility.

MedicalFacility

Medical facilities deal with patients. Each facility has an `id` attribute, a read-only, automatically generated, unique sequential identifier, and a descriptive `name`. Although every facility has a `visit` method, that method is impossible to implement for general medical facilities and is thus *abstract*. See Hospital and Clinic for separate specifications of how the `visit` method is implemented.

Hospital

A hospital is a specialised medical facility which receives visits by patients and can perform procedures on patients. In addition to the attributes inherited from its base class, a hospital has a `probAdmit` attribute which holds a probability (between 0 and 1) that a patient who visits will be admitted.

When a patient *visits* a hospital a random number is generated and compared to the value of `probAdmit` to determine whether the patient will be admitted or not. For example, if `probAdmit` is 0.5, and the random number generated (between 0 and 1) is greater that 0.5, then the patient is admitted. When a patient is admitted, the patient's `currentFacility` attribute is set to refer to the hospital.

Clinic

A clinic is a specialised medical facility which receives visits by patients and offers medical consultations for patients. In addition to the attributes inherited from its base class, a clinic has a `fee` attribute and a `gapPercent` attribute.

When a patient *visits* a clinic a check is done to see if the patient is registered with the clinic or not. If the `currentFacility` does not already contain a reference to this clinic, the patient becomes registered by setting their `currentFacility` attribute to refer to this clinic. A consultation can't be done until the next visit.

If the `currentFacility` attribute for the patient already refers to this clinic, the patient receives a consultation. The cost of a consultation depends on the patient's status (either private or public). If the patient is private, the cost is equal to the clinic's `fee` attribute. If the patient is public, the cost is the `fee` multiplied by the `gapPerCent` attribute. For example, if the `fee` attribute of the clinic is $50, and the `gapPerCent` is 10, then the cost is $5 (10% of $50). In either case, the cost is added to the patient's `balance` attribute.

Procedure

A procedure can be performed on a patient only when the patient is in a hospital. The patient's `currentFacility` attribute indicates this. Each procedure has an `id` attribute, a read-only, automatically generated, unique sequential identifier, and a descriptive `name`. The `isElective` attribute determines whether the procedure is elective or not and each procedure **has** a basic `cost`.

The cost of performing a procedure on a particular patient depends on both the patient and the procedure, according to the table below:

Patient	Procedure	Cost
public	not elective	zero
public	is elective	basic procedure cost
private	not elective	$1000
private	is elective	$2000

User interaction and output

It is a specific requirement of this assignment that **none** of the problem domain classes listed above may contain any user interaction code, including the reading of values from a keyboard or interactive input device and none of the problem domain classes may generate any output for the user, such as screen messages or prompts.

It is acceptable (encouraged) to have screen output messages generated by problem domain classes for **debugging** purposes. These can be very useful during the implementation and testing phases of development. Debugging code should be removed or commented out of all problem domain classes in the final version of the application.

User interface specifications

A Java application fulfilling the role of a user interface should be initiated in a class called `MedicalConsole`. This class is not shown in the diagram (it is not a problem domain class) but is the controlling class for the whole application. The class should provide a *console style* user interface. That is, all output for the user should be directed to standard output (and appears on the screen) and all input should be obtained from standard input (read from the keyboard). Do not create a Graphical User Interface. You will be expected to do that in assignment 2.

The MedicalConsole class must contain the application's **main method** so that the application can be launched with a command equivalent to

```
java MedicalConsole
```

When the application is launched, it should create a single **HealthService** object and should install some sample Patient, Hospital, Clinic and Procedure objects. The application should implement at least the interactions listed below and show them on a menu:

Add an object

This option allows the user to create an object to represent a medical facility, a patient, or a procedure. The user should choose which type of object to create, then should enter the details required, receiving appropriate prompts for all attributes. Use **one** menu option to allow access to all three types of objects, and use as much common code to deal with these three different types of object.

If the user chooses to add a medical facility, the user must choose between a hospital or clinic. If the user wishes to create a procedure object, a hospital must be chosen to add the procedure to. If there are no hospitals, the program must prompt the user to add one, then return to adding the procedure details.

If the user wishes to create a patient object, the *currentFacility* attribute should be set to null to indicate that the patient is not currently admitted to any hospital, and is not registered at any health clinic. The *id* attribute should be automatically generated, and the *balance* should be set to zero. The other attributes are entered by the user.

List objects

Allow the user to choose which type of object to list, then list the objects displaying all their attributes and values. When listing medical facilities, indicate the type, for example, whether a hospital or clinic.

Delete an object

Allow the user to choose which type of object to delete, then to enter a number identifying the object. After checking for a valid object, your program should display details of the object, then ask the user to confirm this is the correct one before deleting. If the object chosen for deletion contains other objects, provide a summary, such as "This hospital can perform 12 procedures - do you still wish to delete it?"

Visit

This option is to simulate a patient visiting a medical facility. Your program must provide an option to choose a patient and a hospital or a clinic, and check for valid entries, then invoke the `visit` method of the facility.

Operate

The user is prompted to specify which hospital, which patient, and which procedure at the hospital and check for valid entries. If all objects are valid the procedure should be performed. Otherwise the program should display a suitable message.

The amount charged based on the table given above is added to the *balance* attribute of the *Patient*. After which a random number is generated to check if the patient should remain in the hospital. If the random number generated is less than the *probAdmit* value, then the patient is discharged by setting the *currentFacility* attribute to *null*. Otherwise, the patient remains in hospital, and an additional procedure may be performed by selecting the Operate option again for the same patient.

Quit

Quit the program.

Extra features

In addition to the menu options above, you may provide extra functionality in order to gain additional marks. You may invent your own extra features which are sensible and add value to the application, or you may use some of those listed below.

- Include an edit feature to allow the user to change details of any object, including changing the name of the health service object. The user should be given the choice of which type of object, followed by which object.
- Display patients in order of name or in order of their balance.

Do not include saving data to a file as an extra feature. That will be expected in assignment 2.

Design and implementation

In completing this assignment you should carefully consider the design of each class. Once you have coded each class, test its functionality completely. Once you have the abstract `MedicalFacility` class constructed and tested, work on the derived sub classes. Then construct the other classes. Make use of 'getters' and 'setters' to access and alter attributes, and provide a `toString` method for each class. You should also consider the need to include overloaded or overriding methods in the various classes.

The object model used in this assignment will be further used and extended in the next assignment. It is important that you design carefully before implementing. A flawed implementation may create problems for you in the second assignment.

Coding style and comments

Your source code should be clear and readable, using correct indentation, meaningful identifiers and comments. Include javadoc comments and tags as follows:

- for public classes, to indicate their purpose;
- for public methods, to indicate their effect, parameters and return values;
- for public fields, to indicate their purpose.

You are expected to write Javadoc comments in your source code, but you need NOT generate and submit the javadoc documentation for your classes as part of this assignment.

SUBMISSION REQUIREMENT

Your assignment has to submit to TurnItIn:

1. Assignment cover sheet.
2. All your Java source files, printed in Word document format
3. Printouts of your source code using Courier-New 10 point size font. You may need to indent your code so that any long Java statements are nicely formatted, and not having the second line of a statement printed starting from the left-hand margin. **You must print your output in portrait orientation**. Printing in landscape mode will be penalised.
4. Printed output (showing your interactivity with your program) is to be included at the end of your Java source files, in Word document created in (2)
5. Generate a runnable jar file of your project, and compress the jar file together with all your java source files into ONE file.
6. Submit your solutions to the Turnitin link created in LMS:
   - The Word document created in (1) is uploaded to Part 1 in Turnitin.
   - The compressed file created in (3) is uploaded to Part 2 in Turnitin.

Marking criteria

Topic	Value
Executing application. Program launches, operates and exits correctly.	20
Inheritance. Correct pattern, design, implementation, coding and overriding for MedicalFacility, Hospital and Clinic classes.	8
Aggregation. Correct design, implementation and coding of collections of facilities, patients and procedures, including supporting methods.	12
Application. Core features present as specified. Sound design, implementation, coding and structure of user interface.	30
Extra. Implementation of extra features.	10
Submission. Includes readable media with class files and documentation. Includes printed source code and output.	10
Presentation. Overall readability, accessibility of assignment	10
Total	**100**