**ASSIGNMENT 1 FRONT SHEET**

|  |  |  |  |
| --- | --- | --- | --- |
| **Qualification** | **BTEC Level 5 HND Diploma in Computing** | | |
| **Unit number and title** | Unit 20: Advanced Programming | | |
| **Submission date** |  | **Date Received 1st submission** |  |
| **Re-submission Date** |  | **Date Received 2nd submission** |  |
| **Student Name** | Phan Minh Tri | **Student ID** | GCC18015 |
| **Class** |  | **Assessor name** | Trung-Viet Nguyen |
| **Student declaration**  I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I understand that making a false declaration is a form of malpractice. | | | |
|  |  | **Student’s signature** |  |

**Grading grid**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P1 | P2 | M1 | M2 | D1 | D2 |
|  |  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **❒ Summative Feedback: ❒ Resubmission Feedback:** | | |
| **Grade:** | **Assessor Signature:** | **Date:** |
| **Lecturer Signature:** | | |

**ASSIGNMENT 1 BRIEF**

|  |  |
| --- | --- |
| **Unit Number and Title** | **20: Advance Programming** |
| Academic Year | 2018 |
| Unit Tutor | Hoàng Đức Quang |
| **Assignment Title** | **Assignment 1** |
| **Issue Date** |  |
| Submission Date |  |
| IV Name & Date |  |

|  |  |  |
| --- | --- | --- |
| **Pass** | **Merit** | **Distinction** |
| **LO1** Examine the key components related to the object-orientated programming paradigm, analysing design pattern types | | **D1** Analyse the relationship between the object-orientated paradigm and design patterns. |
| **P1** Examine the  characteristics of the object-orientated paradigm as well as the various class relationships. | **M1** Determine a design pattern from each of the creational, structural and behavioural pattern types. |
| **LO2** Design a series of UML class diagrams | | **D2** Define/refine class diagrams derived from a given  code scenario using a UML tool. |
| **P2** Design and build  class diagrams using a UML tool. | **M2** Define class diagrams for specific design patterns using a UML tool. |

|  |  |
| --- | --- |
| **Specific requirements**  *(see Appendix for assessment criteria and grade descriptors)* | **Scenario**  You’ve just made a contract with FPT Academy International, and are about to be appointed as a Project Leader for a group of programmers to develop its Student Management System.  In this Student Management System, you are required to create an application to store list of students and list of lecturers. Following information are to be stored for each student:  - stdId: The student ID of the form like GTxxxxx or GCxxxxx (x: is a digit)  - stdName: Student name  - stdDoB: Student date of birth  - stdEmail: Student email  - stdAddress: Student address  - stdBatch: The batch (class) of the student  Following information are to be stored for each lecturer:  - lecId: Lecturer ID with 8 digits (fixed)  - lecName: Lecturer  - lecDoB: Lecturer date of birth  - lecEmail: Lecturer email  - lecAddress: Lecturer address  - lecDept: Lecturer department (e.g., Computing, Business, etc)  This application will need to provide following functionalities via a menu  =======================   1. Manage Students 2. Manage Lecturers 3. Exit   =======================  Please choose:  When user selects 3, the program will exit.  When user selects 1, the program will display submenu for managing students:  =======================   1. Add new student 2. View all students 3. Search students 4. Delete students 5. Update student 6. Back to main menu   =======================  Please choose:  When user selects 2, the program will display submenu for managing lecturers:  =======================   1. Add new lecturer 2. View all lecturers 3. Search lecturers 4. Delete lecturers 5. Update lecturer 6. Back to main menu   =======================  Please choose:  For the submenu:  When user chooses 1, program will prompt user to input student’s/lecturer’s information (specified previously). After that, program will validate the input data and if they are all valid, program will add a new student/lecturer to the current list of students/lecturers. Program should inform to the user corresponding messages.  When user chooses 2, the program will list all the students/lecturers to the screen, each student/lecturer in a row and student’s/lecturer’s data fields are separated by ‘|’.  When user chooses 3, the program will ask user to input student’s/lecturer’s name to search for, the user can just type part of the name in order to search for complete student/lecturer information.  When user chooses 4, program will ask user to input student/lecturer id to delete the student/lecturer with the specified id if it exists, otherwise, it will display a message to inform users that the student/lecturer with such id doesn’t exist.  When user chooses 5, program will first ask user to input student/lecturer id to update, once inserted and a student/lecturer with the inserted id exists, it will display current data for each field of the student/lecturer and user can type in new data to update or just press enter to keep the current data for the field.  When user chooses 6, program will back to the main menu.  **Task 1**  Produce a written, self-learning course for managers/senior developers that explain the principles and features of OOP and that show how OOP is good for code re-use. You can include appropriate sample Java code where it aids your points and/or provides further clarification. Ensure that any diagrams that are included have captions and are referenced in the text. Hint: You must include the following terms  - Object/Class,  - Abstraction,  - Encapsulation,  - Inheritance,  - Polymorphism (Overloading and Overriding),  - Abstract classes,  - Interfaces |
| **Task 2**  Problem Analysis, for the scenario above. You need to produce a full design for the requirements given. The design must include  - Use-case diagrams for the most important features;  - Class diagrams for all objects identified as well as class relationships.  - Pseudo-code for the Algorithms for the main functionalities (3 flowcharts for most complex functions).  Example code can be used to help clarify OOP features. |
| **Submission requirements** | Students are expected to submit hard copy of assignment |

Contents

[**ASSIGNMENT 1 BRIEF** 3](#_Toc70287158)

[Part 1 10](#_Toc70287159)

[Produce a written, self-learning course for managers/senior developers that explain the principles and features of OOP and that show how OOP is good for code re-use. You can include appropriate sample Java code where it aids your points and/or provides further clarification. Ensure that any diagrams that are included have captions and are referenced in the text. 10](#_Toc70287160)

[1. Procedural-oriented Programming 10](#_Toc70287161)

[2. Object-oriented programming 10](#_Toc70287162)

[3. Features of object-oriented programming 11](#_Toc70287163)

[a) Class 11](#_Toc70287164)

[b) Encapsulation 12](#_Toc70287165)

[c) Abstraction 13](#_Toc70287166)

[d) Interface 15](#_Toc70287167)

[e) Polymorphism 16](#_Toc70287168)

[f) Inheritance 17](#_Toc70287169)

[Part 2 19](#_Toc70287170)

[Problem Analysis, for the scenario above. You need to produce a full design for the requirements given. The design must include 19](#_Toc70287171)

[1. Use-case diagram 19](#_Toc70287172)

[2. Class diagram 21](#_Toc70287173)

[3. The Algorithms for the main functionalities 24](#_Toc70287174)

[3.1. Show Menu 25](#_Toc70287175)

[3.2. Student list 26](#_Toc70287176)

[a) Function to add students. 26](#_Toc70287177)

[b) Function to show student list. 27](#_Toc70287178)

[c) Function to search student information. 28](#_Toc70287179)

[d) Function to delete student information. 28](#_Toc70287180)

[e) Function to update student information. 28](#_Toc70287181)

[3.3. Lecturer list 28](#_Toc70287182)

[a) Function to add lecturer. 28](#_Toc70287183)

[References 29](#_Toc70287184)

**Table of figure**

[Figure 1: Use-case diagram 20](#_Toc70287215)

[Figure 2: Class diagram 22](#_Toc70287216)

[Figure 3: Flowchart menu diagram 25](#_Toc70287217)

[Figure 4: Add to student list 26](#_Toc70287218)

[Figure 5: Show student 27](#_Toc70287219)

[Figure 6: Add to lecturer list 28](#_Toc70287220)

# Part 1

# Produce a written, self-learning course for managers/senior developers that explain the principles and features of OOP and that show how OOP is good for code re-use. You can include appropriate sample Java code where it aids your points and/or provides further clarification. Ensure that any diagrams that are included have captions and are referenced in the text.

# Procedural-oriented Programming

* Procedural-oriented programming is a programming style that is based on the concept of the calling process, which is derived from structured programming. Procedures, also known as procedures, subroutines, or functions, are basically a collection of procedural acts that must be performed in a certain order. Any method may be called at any point during the program's execution, either by other protocols or by itself. (geeksforgeeks, n.d.)
* Advantages:
* Procedural-oriented programming is great for strategic planning.
* Written flexibility together with ease of compiler and interpreter implementation.
* A broad range of books and online training content based on validated algorithms to make learning simpler along the way.
* The source code is adaptive, since a certain Processor may also be exploited.
* The algorithm can be reused in certain areas of the program, without the need to copy it.
* The memory level also slashes through the technique of Procedural Programming.
* Effective analysis of the program flow.
* Disadvantages:
* The program code is harder to write when Procedural Programming is employed.
* The Code of Procedure is also not reusable, which might necessitate the reconstruction of the code if it is used in another application.
* Difficult to relate with real-world objects.
* The importance is given to the operation rather than the data, which might pose issues in some data-sensitive cases.
* The data is revealed to the whole software and does not render it secure.

# Object-oriented programming

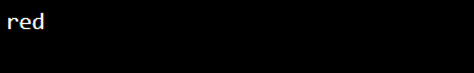
* Object-oriented programming is a programming style that is based upon the concept of objects. Objects contain data in the form of attributes as well as method-shaped code. Computer programs are built-in object-oriented programming that makes use of the concept of objects that interact with the outside world. The most popular object-oriented programming languages are class-based, which means that classes are entity instances that often describe their types.
* Advantages:
* OOP provides ease of operation due to its modularity and encapsulation.
* OOP mimics the real world, which makes things easy to comprehend.
* Since objects are whole within themselves, they can be reused in other programs.
* Disadvantages:
* Object-oriented programs appear to be sluggish and have a large memory consumption.
* Overexploitation.
* Programs constructed using this model can take more time to build. (techdifferences.com, n.d.)

# Features of object-oriented programming

# Class

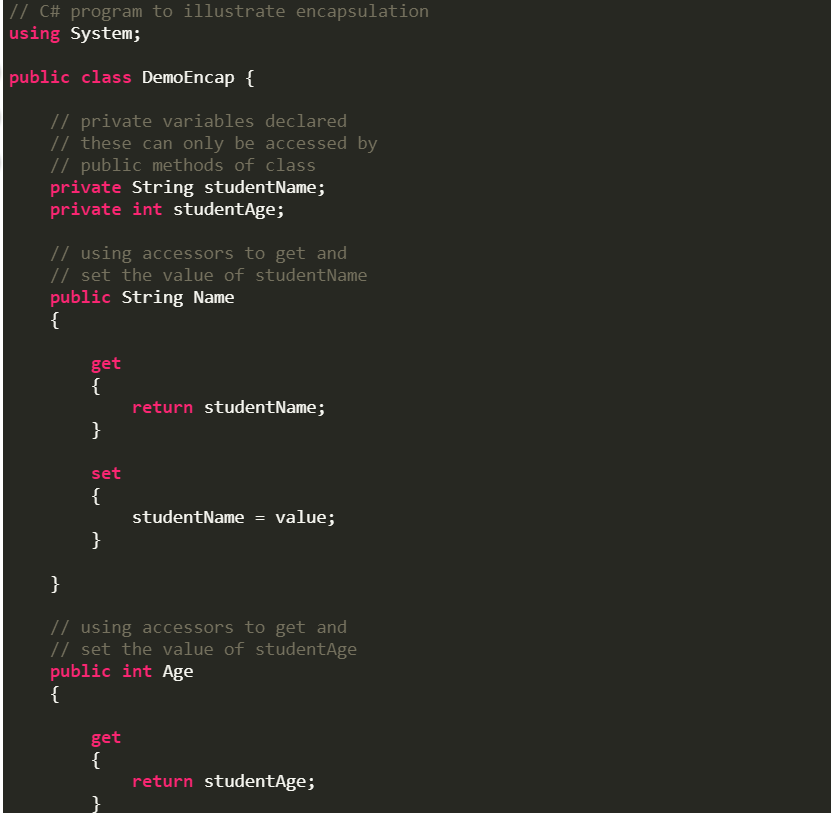
* The fundamental principles of Object-Oriented Programming that revolve around real-life entities are class and object. A class is a pre-defined blueprint or template from which objects can be produced. A class essentially incorporates fields and methods (member functions that describe actions) into a single object. Objects in C# support polymorphism, inheritance, and the concepts of derived classes and base classes.

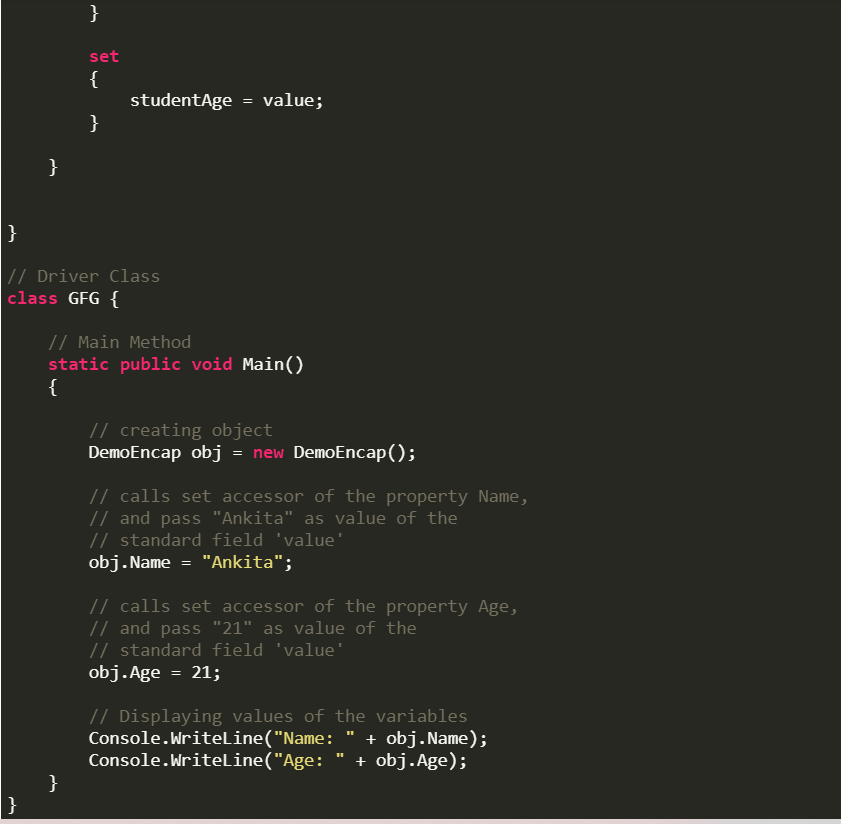




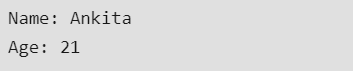
# Encapsulation

* Encapsulation is described as "the process of assembling one or more objects into a physical or logical package." Encapsulation limits access to knowledge about implementation in object-oriented programming theory.
* Encapsulation assists a programmer in enforcing the abstraction level of choice.
* Encapsulation is another essential function of OOP. Encapsulation is the phase where the data are not allowed clear access; instead, it is hidden. If you want to view the data, you must communicate with the data entity responsible.
* Think about when you send a letter in the mail. You request the post office to deliver the letter. How the post office accomplishes this is not exposed to you. If it changes the route it uses to mail the letter, it does not affect how you initiate the sending of the letter. You do not have to know the post office’s internal procedures used to deliver the letter.





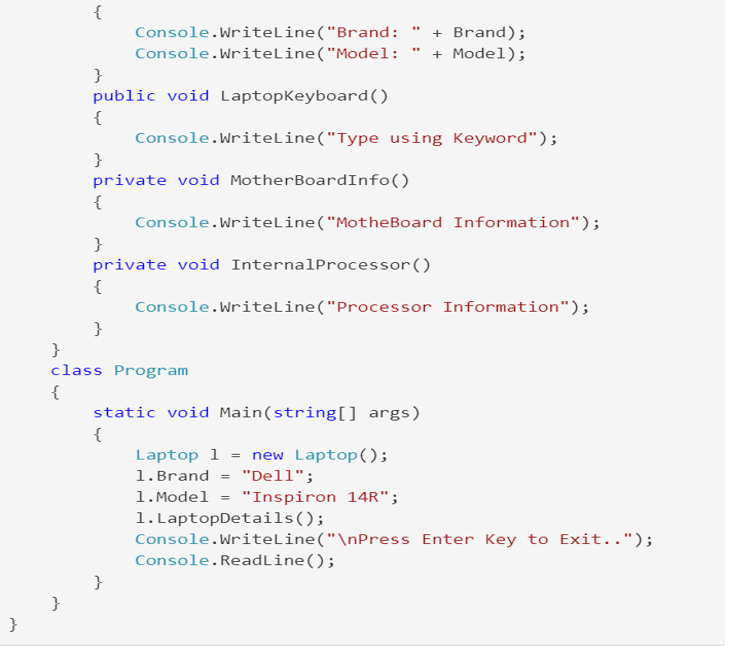
* OUT PUT:



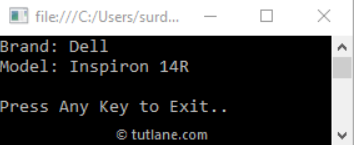
# Abstraction

* Data abstraction is a feature for which the user is only provided specific details. Unnecessary information would be hidden from the recipient. An object's properties and behavior of an object distinguish it from other similar-type objects and aid in classifying / grouping the objects.
* Abstraction is an object-oriented programming language (OOP) principle used in C# to cover design specifics and view only basic features of the object. A class is the best representation of abstraction in object-oriented programming. In c#, we can create a class with required methods, properties and we can expose only necessary methods and properties using access modifiers based on our requirements. (tutlane.com, n.d.)



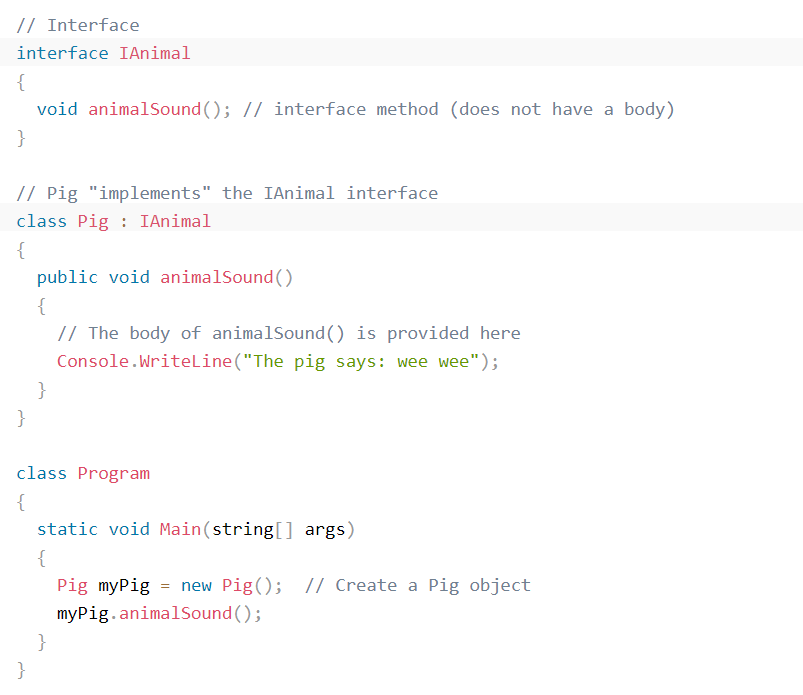


* OUT PUT

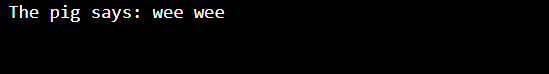


# Interface

* Interface, like a class, may have processes, properties, occurrences, and indexers as members. Interfaces, on the other hand, can only hold participant comments. The class that implicitly or expressly implements the interface can have an implementation of the interface's members.
* Members of the interface may be processes, properties, events, and indexers. However, interfaces can only include the declaration of the representatives. The class that implements the interface, either implicitly or directly, will include the implementation of the interface's participants.
* The configuration keyword in C # can be used to define an interface. Statements concerning structures, properties, indexers, and events can all be found in an app. It cannot contain fields, but it does contain auto-implemented properties. (tutorialsteacher.com, n.d.)

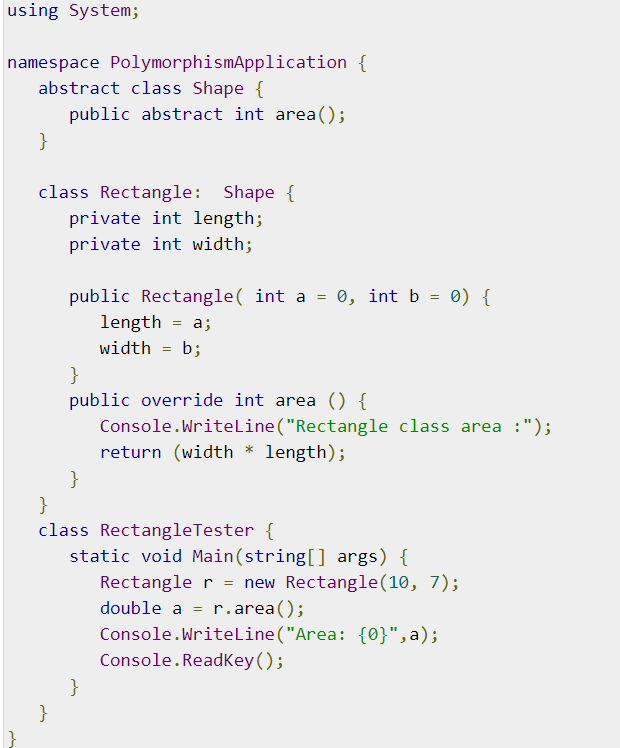


* OUT PUT:



# Polymorphism

* The term polymorphism denotes various types. Polymorphism is also defined as 'one interface, many features in the object-oriented programming framework.
* It can be static or interactive to polymorphism. At compile time, the answer to a function is defined in static polymorphism. On run-time, it is determined in dynamic polymorphism.



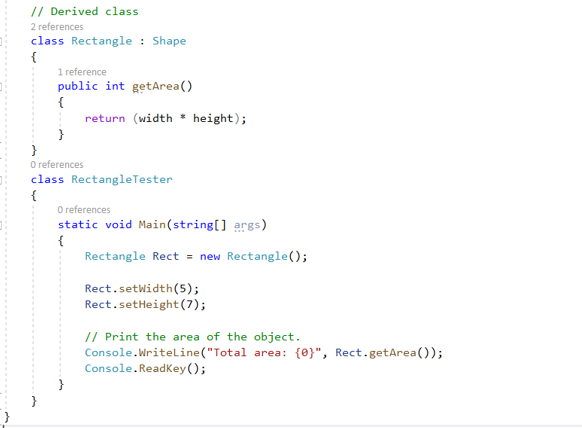
* OUT PUT:



# Inheritance

* One of the most important concepts of object-oriented programming is inheritance. Inheritance facilitates the creation and management of an application by allowing one to define a class as another class. It also allows for the reuse of programming elements, which reduces production time.
* When constructing a class, the author may choose to inherit the members of a current class instead of writing completely new data members and member functions. The existing class is referred to as the base class, and the new class is referred to as the derived class.
* The concept of succession puts the IS-A relationship into action. For example, a mammal IS an animal, a dog IS a mammal, and therefore a dog IS an animal, and so on.





* OUT PUT:



# Part 2

# Problem Analysis, for the scenario above. You need to produce a full design for the requirements given. The design must include

# Use-case diagram

|  |  |
| --- | --- |
| **Icon** | **Visibility** |
|  | Associations are often used to describe the relationship between Actor and Use Cases and between Use Cases. |
|  | The term "extend" was used to explain the relationship between the two Use Cases. Extend is used where a Use Case is developed to apply features to an existing Use Case which is only used under such circumstances. |
|  | Use Case is the function that Actors will use. |
|  | The "actor" refers to a user or an external entity that interacts with the device under consideration. |

Table : Annotate symbols

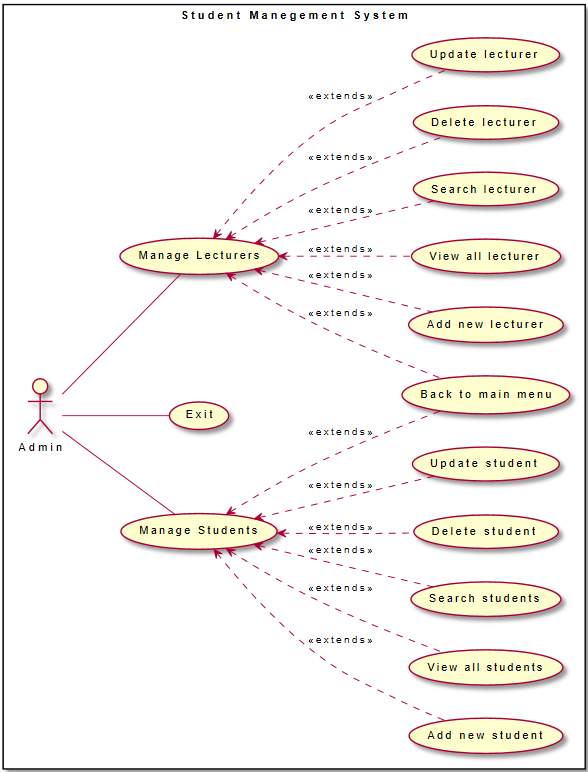


Figure : Use-case diagram

* Customers communicate with the system mostly through two functions: student management and faculty management. The list can be added to, modified, deleted from, searched for, and shown.
* @startuml
* left to right direction
* skinparam packageStyle rectangl
* rectangle "Student Manegement System" {
* Admin -- (Manage Students)
* Admin -- (Exit)
* Admin -- (Manage Lecturers)
* (Manage Students) <.- (Add new student) : <<extends>>
* (Manage Students) <.- (View all students) : <<extends>>
* (Manage Students) <.- (Search students) : <<extends>>
* (Manage Students) <.- (Delete student) : <<extends>>
* (Manage Students) <.- (Update student) : <<extends>>
* (Manage Students) <.- (Back to main menu) : <<extends>>
* (Manage Lecturers) <.- (Add new lecturer) : <<extends>>
* (Manage Lecturers) <.- (View all lecturer) : <<extends>>
* (Manage Lecturers) <.- (Search lecturer) : <<extends>>
* (Manage Lecturers) <.- (Delete lecturer) : <<extends>>
* (Manage Lecturers) <.- (Update lecturer) : <<extends>>
* (Manage Lecturers) <.- (Back to main menu) : <<extends>>
* }
* @enduml

# Class diagram

|  |  |
| --- | --- |
| **Icon** | **Visibility** |
|  | Class |
|  | Interface |
|  | private |
|  | public |
|  | Object |
|  | Extension |

Table : Class diagram annotation icon

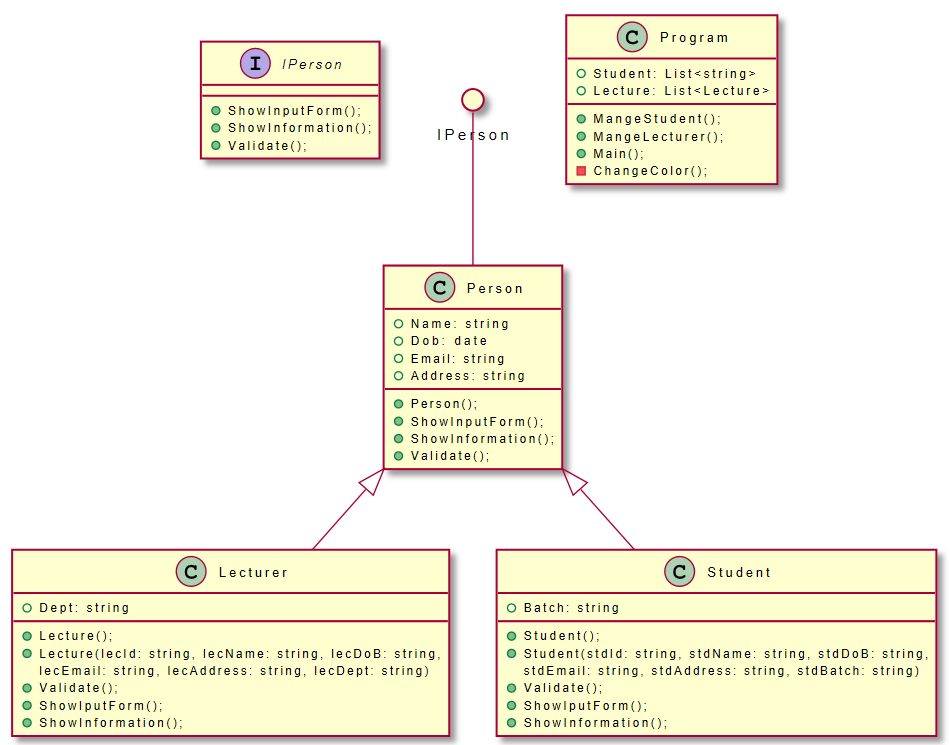


Figure : Class diagram

@startuml

'' Class

skinparam linetype orth

interface "IPerson" as 2 {

+ShowInputForm();

+ShowInformation();

+Validate();

}

circle IPerson

IPerson -- Person

Person <|-- Lecturer

Person <|-- Student

class Person {

+Name: string

+Dob: date

+Email: string

+Address: string

+Person();

+ShowInputForm();

+ShowInformation();

+Validate();

}

class Student {

+Batch: string

+Student();

+Student(stdId: string, stdName: string, stdDoB: string,

stdEmail: string, stdAddress: string, stdBatch: string)

+Validate();

+ShowIputForm();

+ShowInformation();

}

class Lecturer {

+Dept: string

+Lecture();

+Lecture(lecId: string, lecName: string, lecDoB: string,

lecEmail: string, lecAddress: string, lecDept: string)

+Validate();

+ShowIputForm();

+ShowInformation();

}

class Program {

+Student: List<string>

+Lecture: List<Lecture>

+MangeStudent();

+MangeLecturer();

+Main();

-ChangeColor();

}

@enduml

# The Algorithms for the main functionalities

|  |  |
| --- | --- |
| **Icon** | **Visibility** |
|  | Used to represent the Start and end of a program with the Keywords BEGIN and END. |
|  | Used to represent data entry by a user or the display of data by the program. |
|  | An instruction that is to be carried out by the program. |
|  | Used to split the flowchart sequence into multiple paths in order to represent SELECTION and REPETITION. |
|  | Indicates the flow of the algorithm pathways. |

Table : Flowchart annotation icon

# Show Menu

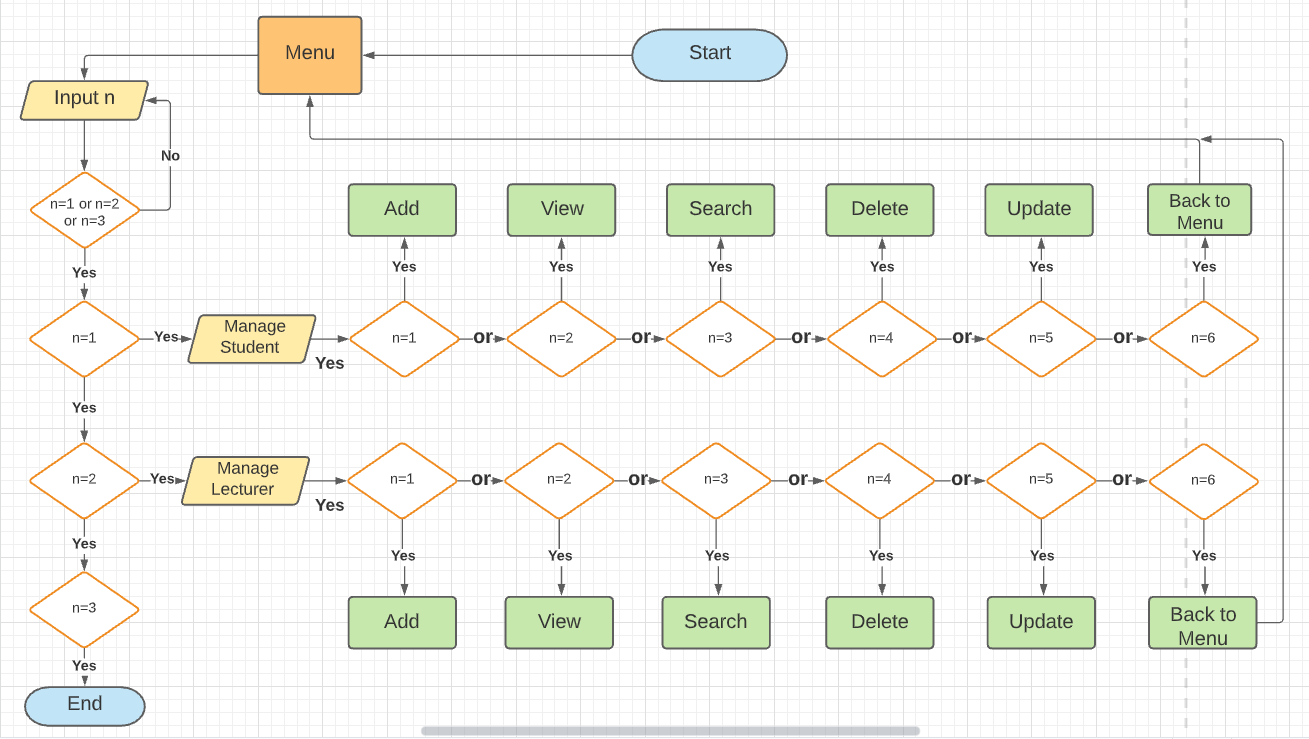


Figure : Flowchart menu diagram

* When a customer enters a value from the keyboard, the system will only accept n values between 1 and 3. If the value is outside of this range, the system will allow the customer to re-enter n new.
* If n equals one, the system redirects to student management; if n equals two, the system redirects to lecturer management. Otherwise, n = 3 customers will leave the system.
* Both the student management and lecturer management websites support values ranging from 1 to 6, and if the value falls outside of this range, the user will be prompted to insert a different value with similar functions.
* If n = 1, the value redirects to adding a new audience tab to the list, n = 2 to showing all objects in the list, n = 3 to searching for objects in the list, n = 4 to deleting objects in the list, and n = 5 to updating object information in the list. The system would then revert to the previous page if this is not done.

# Student list

# Function to add students.

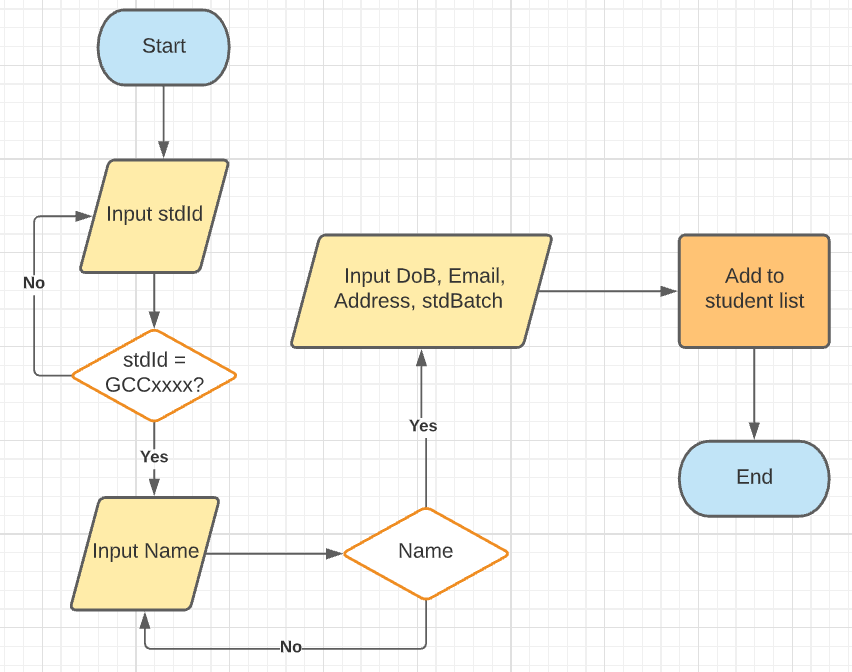


Figure : Add to student list

* The customer enter stud, if stdId starting is GCC or study does not exist in the student list, the system will give enter the student’s name. Otherwise, the customer will give re-enters stdId.
* If the student’s name already exists in the student list, the system will let the customer re-enter the student’s name. In contrast, the student’s name validates the system for the date of birth, email, address, and class then saves the student information in the student list.

# Function to show student list.

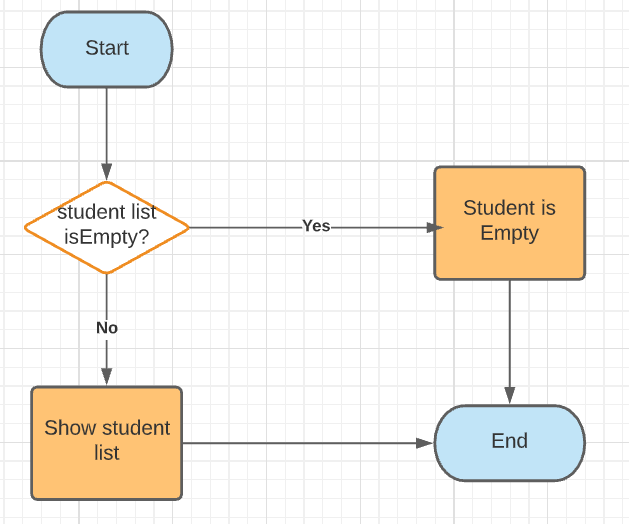


Figure : Show student

# Function to search student information.

# Function to delete student information.

# Function to update student information.

# Lecturer list

# Function to add lecturer.

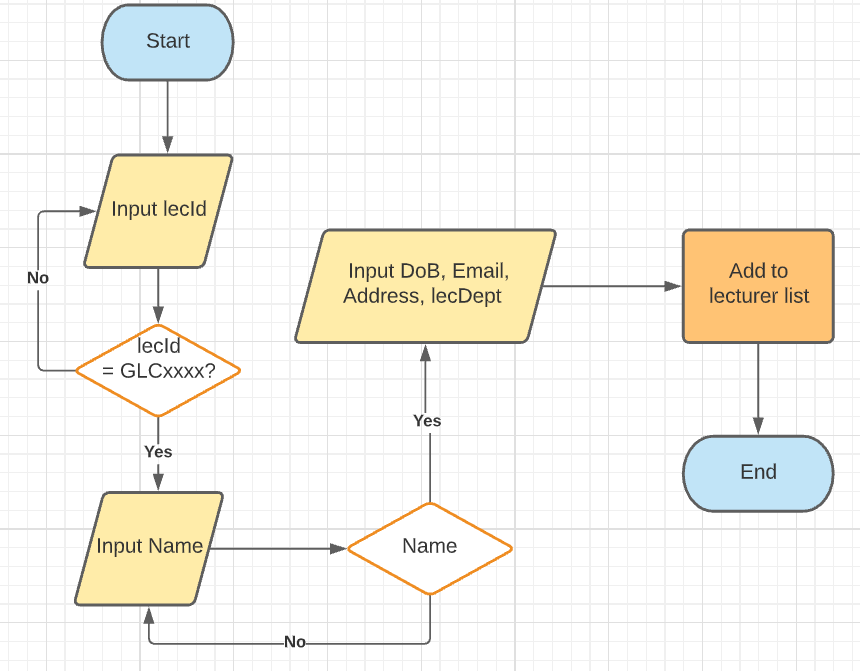


Figure : Add to lecturer

# References

geeksforgeeks, n.d. [Online]   
Available at: https://www.geeksforgeeks.org/c-sharp-class-and-object/  
[Accessed 24 4 2021].

techdifferences.com, n.d. [Online]   
Available at: https://techdifferences.com/difference-between-oop-and-pop.html  
[Accessed 25 4 2021].

tutlane.com, n.d. [Online]   
Available at: https://www.tutlane.com/tutorial/csharp/csharp-abstraction  
[Accessed 25 4 2021].

tutlane.com, n.d. [Online]   
Available at: https://www.tutlane.com/tutorial/csharp/csharp-abstraction  
[Accessed 25 4 2021].

tutorialsteacher.com, n.d. [Online]   
Available at: https://www.tutorialsteacher.com/csharp/csharp-interface  
[Accessed 25 4 2021].