ASSIGNMENT 1 FRONT SHEET

Qualification	**BTEC Level 5 HND Diploma in Computing**		
Unit number and title	Unit 20: Advanced Programming		
Submission date	Sunday, April 25, 2021	**Date Received 1st submission**	
Re-submission Date		**Date Received 2nd submission**	
Student Name	Nguyen Quoc Viet	**Student ID**	GCC18157
Class	GCC0701-1651	**Assessor name**	Nguyen Trung Viet

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	Quoc Viet

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		
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.	**D1** Analyse the relationship between the object-orientated paradigm and design patterns.
LO2 Design a series of UML class diagrams		
P2 Design and build class diagrams using a UML tool.	**M2** Define class diagrams for specific design patterns using a UML tool.	**D2** Define/refine class diagrams derived from a given code scenario using a UML tool.

Specific requirements *(see Appendix for assessment criteria*	**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.

and grade descriptors)	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 ====================== Manage Students Manage Lecturers Exit ====================== Please choose: When user selects 3, the program will exit.

	When user selects 1, the program will display submenu for managing students: ====================== Add new student View all students Search students Delete students Update student Back to main menu ====================== Please choose: When user selects 2, the program will display submenu for managing lecturers: ====================== Add new lecturer View all lecturers Search lecturers Delete lecturers Update lecturer 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.
Student guidelines	For the assignment assessments, you are required to: 1. Produce a design in UML that fully utilizes OOP principles and features. (Use case, class diagrams, collaboration diagrams etc.).
Submission requirements	Students are expected to submit hard copy of assignment

Grade Descriptor

PASS criteria

LO	Learning outcome (LO)	AC	In this assessment you will have the opportunity to present evidence that shows you are able to:	Task no.
LO1	Examine the key components related to the object-orientated programming paradigm, analysing design pattern types	1	Examine the characteristics of the object-orientated paradigm as well as the various class relationships.	1
LO2	Design a series of UML class diagrams		Design and build class diagrams using a UML tool.	

In addition to the above PASS criteria, this assignment gives you the opportunity to submit evidence in order to achieve the following MERIT and DISTINCTION grades		
Grade Descriptor	Indicative characteristic/s	Contextualization
M1	Determine a design pattern from each of the creational, structural and behavioural pattern types.	
M2	Define class diagrams for specific design patterns using a UML tool.	
D1	Analyse the relationship between the object-orientated paradigm and design patterns.	

D2	Define/refine class diagrams derived from a given code scenario using a UML tool.	

This brief has been verified as being fit for purpose					
Internal Verifier 1		Signature		Date	
Internal Verifier 2		Signature		Date	

Contents

P1 Examine the characteristics of the object-orientated paradigm as well as the various class relationships.

There are three widely used programming paradigms: procedural programming, functional programming and object-oriented programming. C# supports both procedural and object-oriented programming.

I. What is POP?

Procedural Oriented Programming is one of the programming methods where the main focus is on functions or procedures required for computation, instead of data.

The program is divided into functions, and the task is done sequentially. These functions share the global data or variables, and there is an exchange of data among those functions.

II. What is OOP?

Object-Oriented Programming is one of the high-level programming languages in which a program is divided into objects. Using objects, the programmer can model real-world scenarios. An object is an instance of a class and has state and behavior. The state is the attributes, or data, whereas Behavior is called a method.

The difference between POP and OOP:

	OOP	**POP**
Basic Definition	OOP is object-oriented.	POP is structure or procedure-oriented.
Approach	Bottom-Up approach	Top-down approach
Technology background	The main focus is on "how to complete the task" means about the procedure or structure of a program.	The main focus is "data security". Therefore, only objects are allowed access to instances of a class.
Division	Large programs are divided into units called functions.	The entire program is divided into objects.
Entity access	No access specifiers	Access is determined "public", "private", "protected".
Mode		"protected".
Share data	System-wide data is shared among program functions.	Data is shared among objects through functions.
Commonly used language	TC, VB, INTRODUCTION, Pascal	C ++, JAVA, VB.NET, C # .NET, Ruby

1. **Objects/Classes:**
   Objects: An object is an instance of a class that collects data and procedures for manipulating data.
   Classes: A class defines the properties of objects linked to it.

Relationship:

NO.	**Object**	**Class**
1.	The object is an instance of a class	Classes are a template or design for creating objects
2.	The object is a real-world entity like a pencil or a bicycle	A class is a group of similar objects
3.	The object is a physical entity	Class is a logical entity
4.	The object is created mostly from the **new** keyword. Ex: Student s1=new Student();	Class is declared using the class keyword Ex: class Student()
5.	Objects can be created many times	Class is declared only once
6.	The object is allocated memory when it is created	The class is not allocated memory when it is created

2. **Abstraction:**
   Abstract classes and interfaces are used to hide the internal details and show the functionality.
3. **Encapsulation:**
   Data is secured with encapsulation, and binds the attributes and methods together.

Difference between Abstraction and Encapsulation:

Abstraction	**Encapsulation**
Abstraction is the process or method of gaining the information.	While encapsulation is the process or method to contain the information.
In abstraction, problems are solved at the design or interface level.	While in encapsulation, problems are solved at the implementation level.
Used to hide the unwanted data and only give relevant data.	Hide the code & data to protect the data from outer world.
Abstraction is outer layout in terms of design.	Encapsulation is inner layout in terms of implementation.
Focus on the object instead of how does it work.	Encapsulation means hiding the internal details or mechanics of how an object does something.

4. **Inheritance**:
   An object using the methods and properties of an existing object, is called inheritance. It enhances code reusability.
5. **Polymorphism:**
   With polymorphism, an object can function in multiple ways. Examples: Method overloading and method overriding.

Difference between Inheritance and Polymorphism:

Inheritance	**Polymorphism:**
Inheritance is one in which a new class is created (derived class) that inherits the features from the already existing class(Base class).	Whereas polymorphism is that which can be defined in multiple forms.

It is basically applied to classes.	Whereas it is basically applied to functions or methods.
Inheritance supports the concept of reusability and reduces code length in object-oriented programming.	Polymorphism allows the object to decide which form of the function to implement at compile-time (overloading) as well as run-time (overriding).
Inheritance can be single, hybrid, multiple, hierarchical and multilevel inheritance.	Whereas it can be compiled-time polymorphism (overload) as well as run-time polymorphism (overriding).
It is used in pattern designing.	While it is also used in pattern designing.

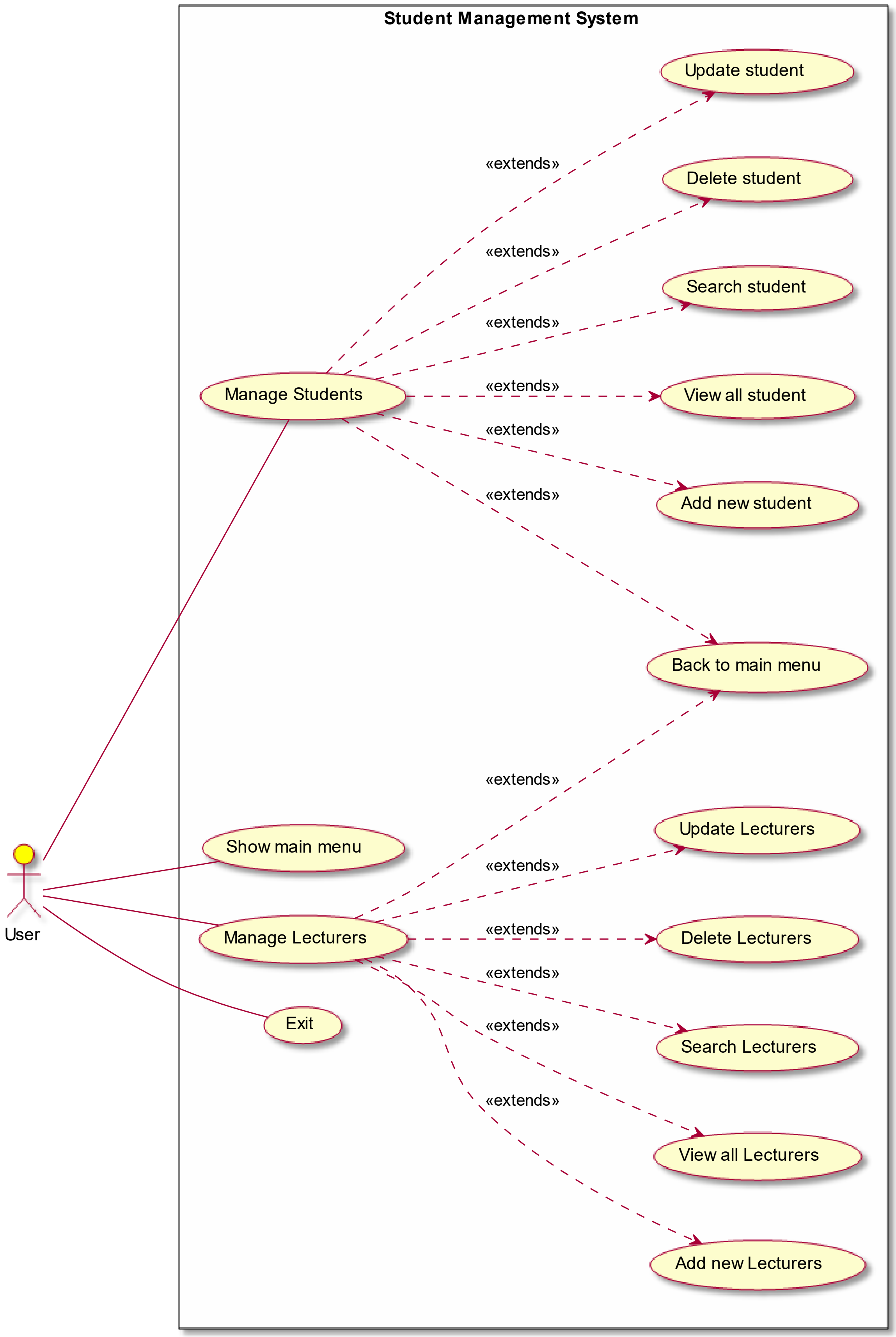
6. **Abstract Class:**
   Abstract classes "can never be instantiated and is marked by the keyword abstract. An abstract class should have a minimum of one abstract method. Abstract class acts as a base class and is designed to be inherited by subclasses that either implement or either override its method."
7. **Interfaces:**
   "An interface is a contract that guarantees to a client how a class or struct will behave. When a class implements an interface, it tells any potential client "I guarantee I'll support all the methods, properties, events, and indexers of the named interface." - The syntax for defining an interface is as follows:
   [attributes] [access-modifier] interface interface-name[:base-list] {interface-body}
   Alteration of access, including internal public, private, secure, internal and safe.
   The keyword interface is accompanied by the interface name. Starting the interface name with a capital I (thus, IStorable, ICloneable, IClaudius etc.) is popular (but not required).
   The base list lists the interfaces that this interface extends (as defined in the "Implementing More than One Interface" section below). The interface-body specifies the methods, the properties, and so on that the implementing class must enforce.

P2 Design and build class diagrams using a UML tool.

I. Use case Diagram

Student administration: Administrators may conduct student management roles to handle students in an overall manner. Student management functions include basic functions such as: adding , deleting, upgrading students, administrators can scan for and display a list of all the students entered while active, and a spin feature. Again Main Menu.

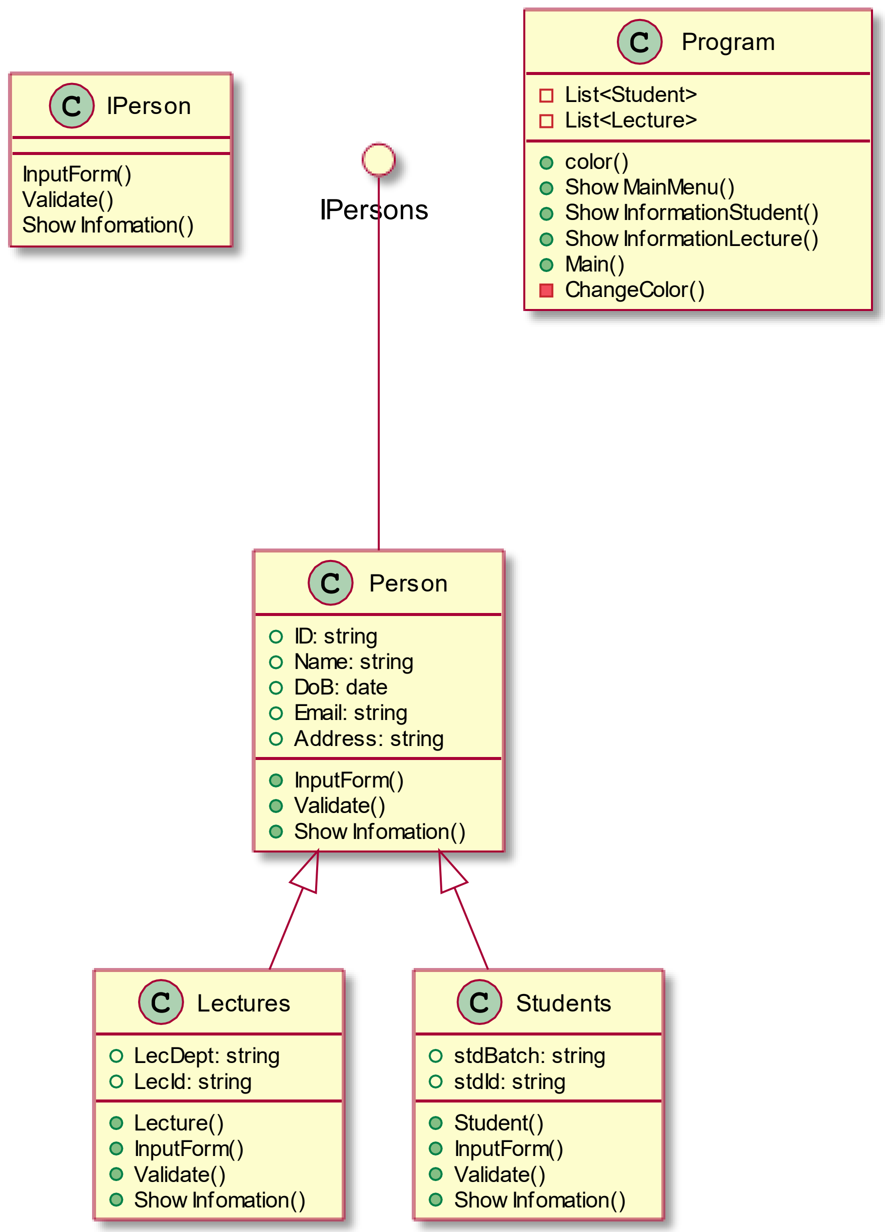
Lectures in Management: Administrators may perform the same roles as the Student Management section.


Student Management System
Update student
«extends»
Delete student
«extends»
Search student
«extends»
Manage Students
«extends»
View all student
«extends»
Add new student
«extends»
Back to main menu
«extends»
Update Lecturers
«extends»
Show main menu
Delete Lecturers
«extends»
Manage Lecturers
«extends»
Search Lecturers
«extends»
«extends»
View all Lecturers
Add new Lecturers
User
Exit

Use case code:

```
@startuml
" UseCase
left to right direction
skinparam packageStyle rectangle
actor User #yellow
rectangle "Student Management System" {
  User --- (Manage Students)
  (Manage Students) ...> (Add new student) : <<extends>>
  (Manage Students) ...> (View all student) : <<extends>>
  (Manage Students) ...> (Search student) : <<extends>>
  (Manage Students) ...> (Delete student) : <<extends>>
  (Manage Students) ...> (Update student) : <<extends>>
  (Manage Students) ...> (Back to main menu) : <<extends>>
    User --- (Show main menu)
  User --- (Manage Lecturers)
  (Manage Lecturers) ...> (Add new Lecturers) : <<extends>>
  (Manage Lecturers) ...> (View all Lecturers) : <<extends>>
  (Manage Lecturers) ...> (Search Lecturers) : <<extends>>
  (Manage Lecturers) ...> (Delete Lecturers) : <<extends>>
  (Manage Lecturers) ...> (Update Lecturers) : <<extends>>
  (Manage Lecturers) ...> (Back to main menu) : <<extends>>
   User --- (Exit)
@enduml
```

II. Class Diagram:



Class Diagram code:

```
@startuml
" Class
Class IPerson
Class Person
Class Lectures
Class Students
Class Program
circle IPersons
```

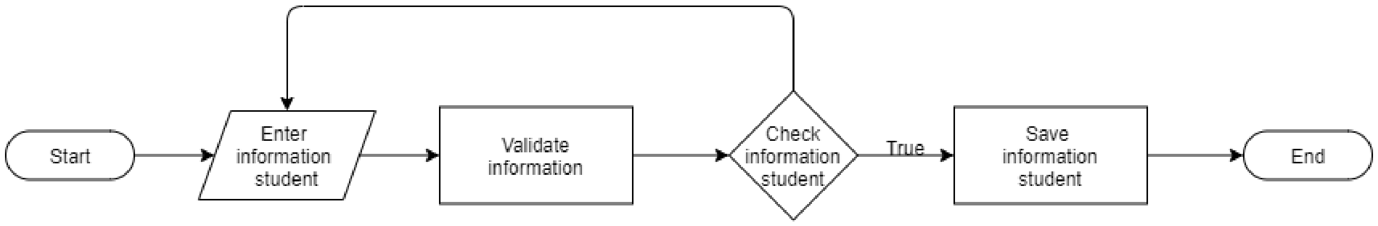
```
IPersons --- Person
Person <|-- Students
Person <|-- Lectures
IPerson : InputForm()
IPerson : Validate()
IPerson : ShowInfomation()
Class Person {
            +ID: string
            +Name: string
            +DoB: date
            +Email: string
            +Address: string

            +InputForm()
            +Validate()
            +ShowInfomation()
}
Class Students {
            +stdBatch: string
            +stdId: string
            +Student()
            +InputForm()
            +Validate()
            +ShowInfomation()
}

Class Program {
 -List<Student>
 -List<Lecture>
 +color()
 +ShowMainMenu()
 +ShowInformationStudent()
 +ShowInformationLecture()
 +Main()
 -ChangeColor()
}

Class Lectures{
            +LecDept: string
            +LecId: string
            +Lecture()
            +InputForm()
            +Validate()
            +ShowInfomation()
}
```

@enduml

III. Activity Diagram:

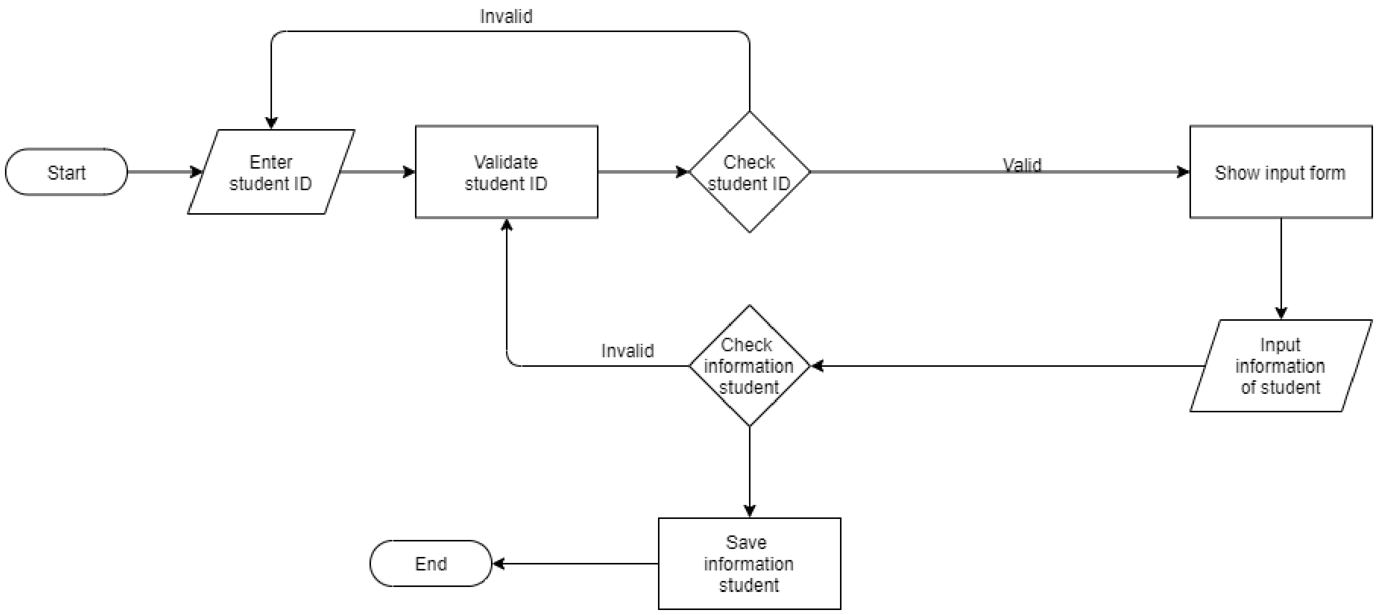
I. Information Student:

1. Add Information Student



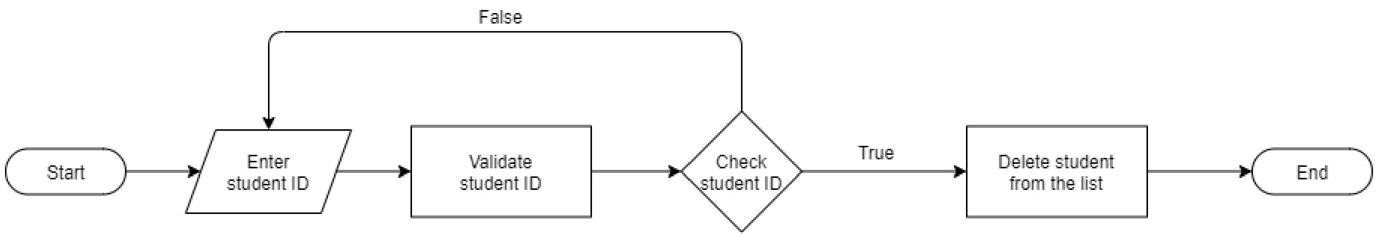
If the user needs to assign students to a specific class, use this case. To add students to a particular list, the system allows users to select classes and personal details of students.

2. Update Student:



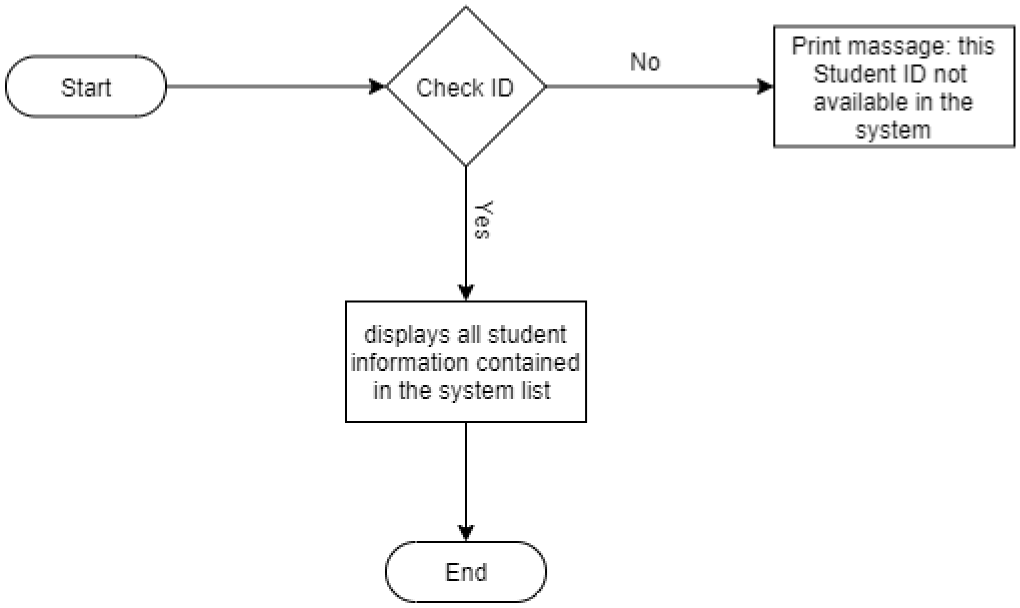
Update student information: When a student wishes to make changes to their information, they can go to the update student page and choose to save.

3. Delete student:



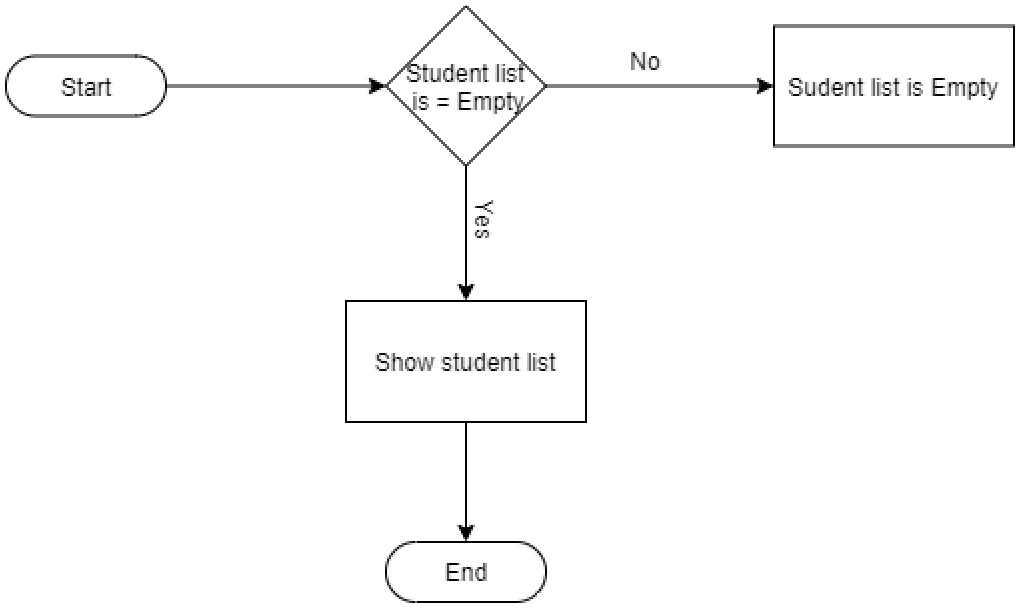
To delete students, the user selects Delete.

4. **Search information student:**


Start
Check ID
No
Print massage: this Student ID not available in the system
Yes
displays all student information contained in the system list
End

To search for students, the user enters the student number, the system will display the student name that needs to be searched.

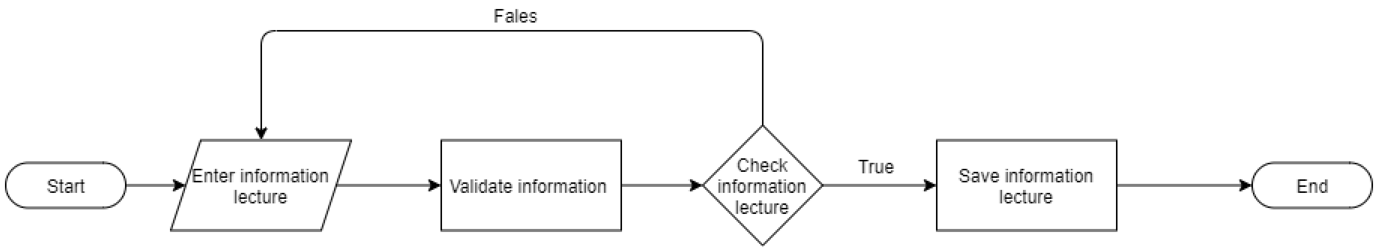
5. **Show student:**



To display students, the user enters the student number, the system will display student information including student number, Email ... etc ...

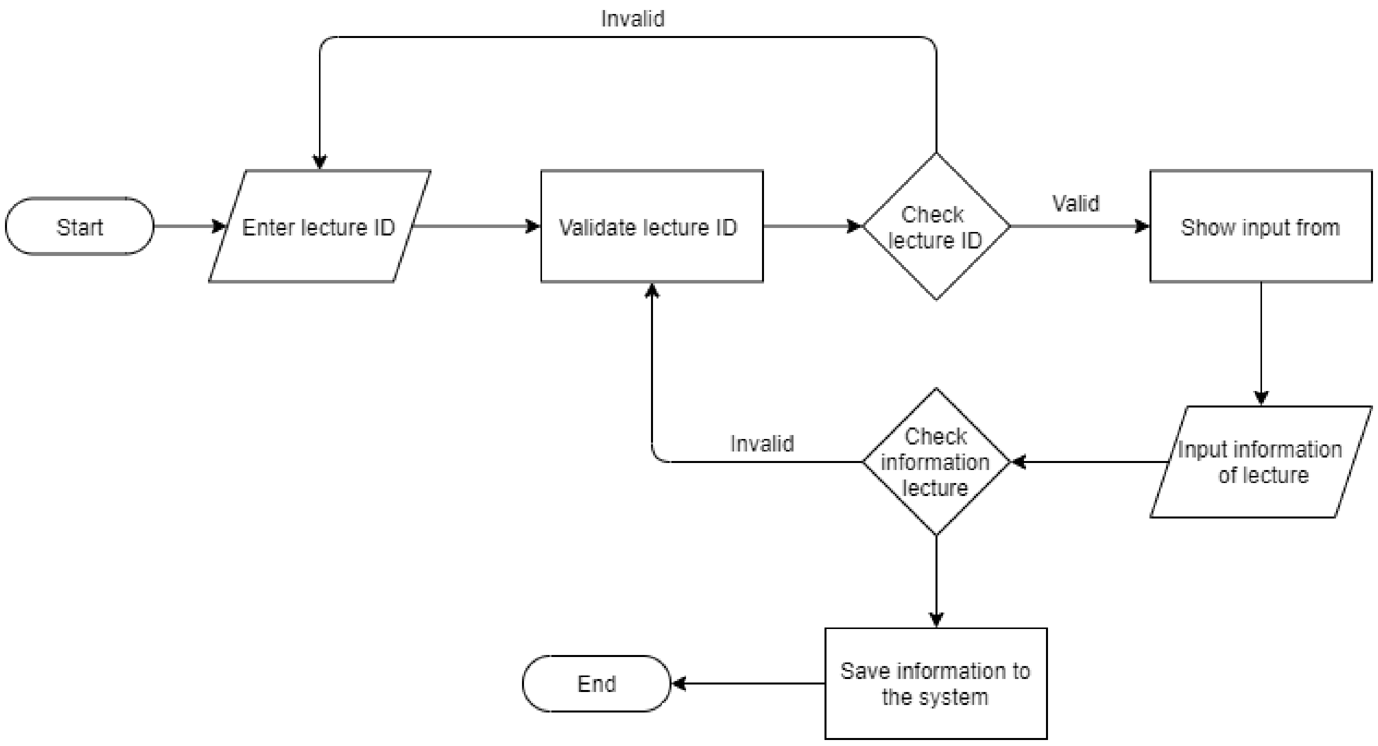
II. Information Lecture:

1. Add lecture:



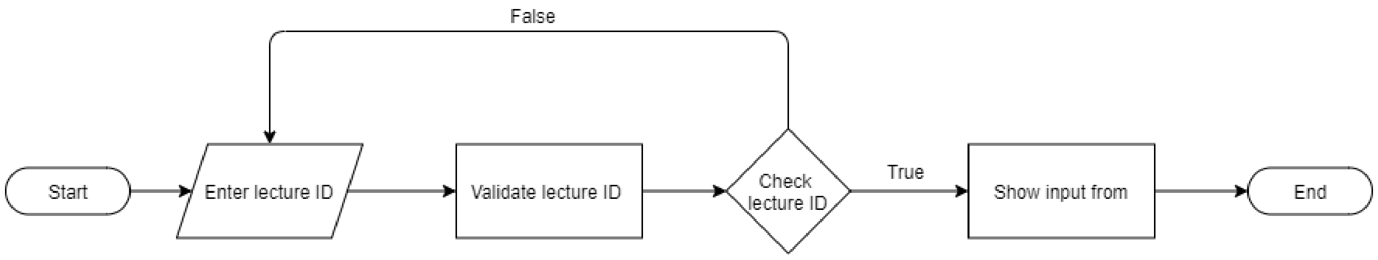
Use this case starting when a user wants to add a lecture of a certain class. The system requires the user to select the class and personal information of the lecture for the user to add the lecture to a specific list.

2. Update lecture:



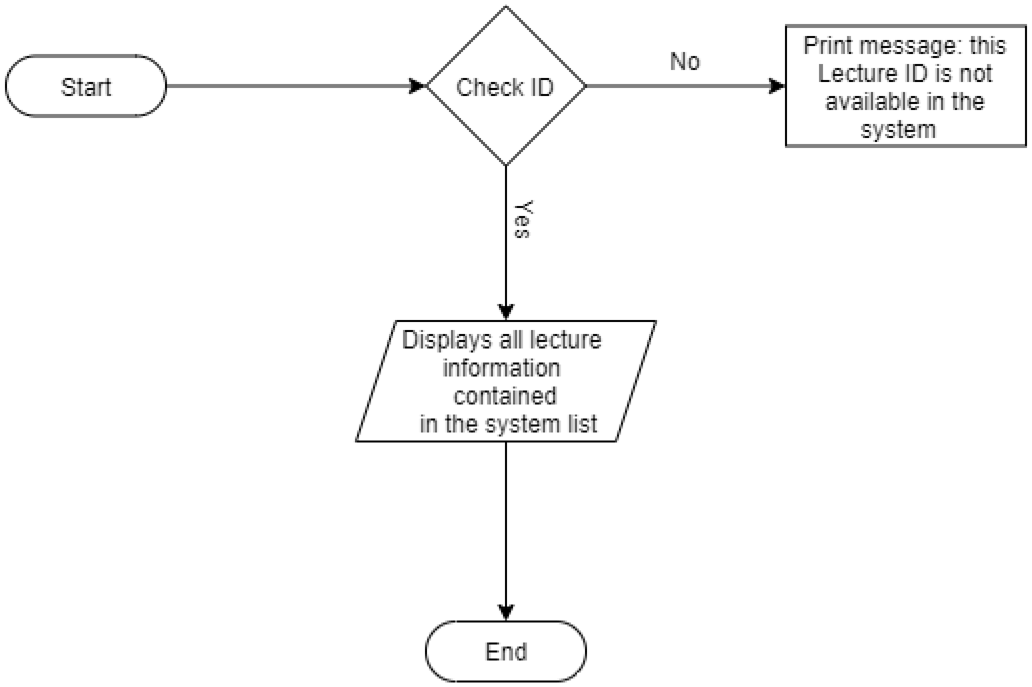
Update lecture information: When a lecture wishes to make changes to their information, they can go to the update lecture page and choose to save.

3. Delete lecture:



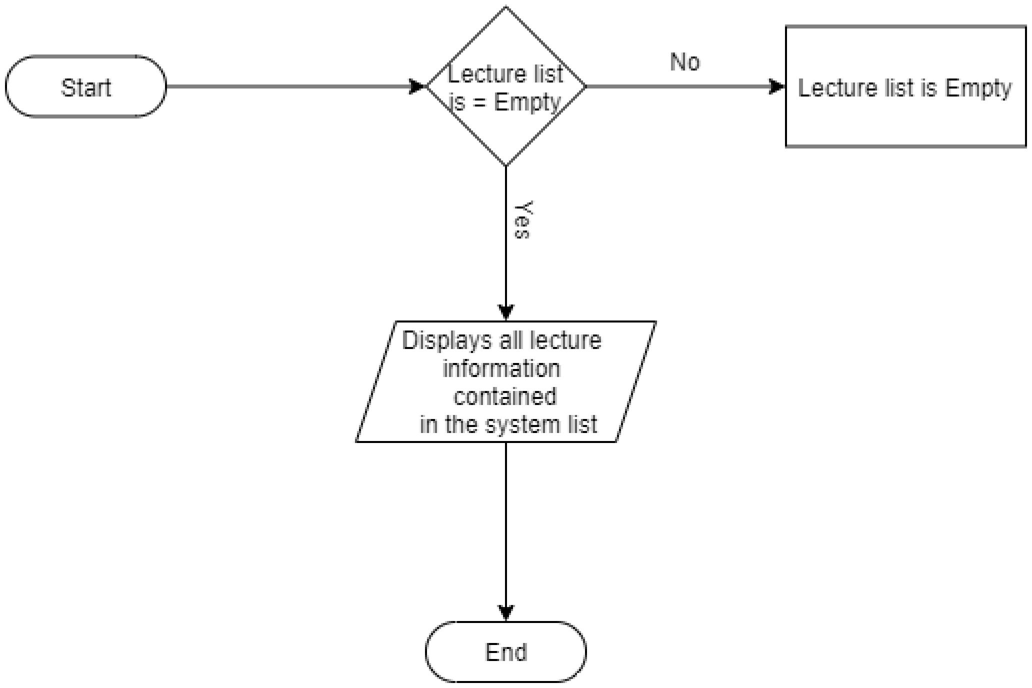
To delete lecture, the user selects Delete.

4. Search lecture:


Start
Check ID
No
Print message: this Lecture ID is not available in the system
Yes
Displays all lecture information contained in the system list
End

As the user types in the lecture number, the device displays the name of the lecture that needs to be found.

5. **Show information lecture:**


Start
Lecture list is = Empty
No
Lecture list is Empty
Yes
Displays all lecture information contained in the system list
End

The user enters the lecture number to view the lecture, and the device displays all details related to that lecture.

Reference:

Clark, D., 2013. *Beginning C# Object-Oriented Programming.* 2nd ed. New York: Springer Science+Business Media New York.

Guru99, n.d. *C# Abstract Classes Tutorial with Example.* [Online] Available at: https://www.guru99.com/c-sharp-abstract-class.html

Shildt, H., 2010. *C# 4.0: the complete reference.* 2nd ed. Tata McGraw-Hill: Education.

Xie, J. L. a. D., 2005. *Programming C#: building. NET applications with C.* 5th ed. Inc: O'Reilly Media.

https://www.programmingassignmentshelp.net/object-oriented-programming-assignment-help/