



Higher Nationals in Computing

# Unit 20: Advanced Programming ASSIGNMENT 1

Learner’s name:

ID:

Class:

Subject code: 1651

Assessor name: **PHAN MINH TAM**

Assignment due: Assignment submitted:

**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** |  | **Student ID** |  |
| **Class** |  | **Assessor name** | Phan Minh Tam |
| **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:** |
| **Signature & Date:** | | |

**ASSIGNMENT 1 BRIEF**

|  |  |  |  |
| --- | --- | --- | --- |
| **Qualification** | **BTEC Level 5 HND Diploma in Business** | | |
| **Unit number** | Unit 20: Advanced Programming | | |
| **Assignment title** | Examine and design solutions with OOP and Design Patterns | | |
| **Academic Year** | 2021 - 2022 | | |
| **Unit Tutor** | Phan Minh Tam | | |
| **Issue date** | 28 September 2021 | **Submission date** |  |
| **IV name and date** |  | | |

|  |
| --- |
| **Submission Format:** |
| *Format:* The submission is in the form of a **group written report**. This should be written in a concise, formal business style using single spacing and font size 12. You are required to make use of headings, paragraphs and subsections as appropriate, and all work must be supported with research and referenced using the Harvard referencing system. Please also provide a bibliography using the Harvard referencing system.  *Submission* Students are compulsory to submit the assignment in due date and in a way requested by the Tutors. The form of submission will be a soft copy in PDF posted on corresponding course of <http://cms.greenwich.edu.vn/>  *Note:* The Assignment *must* be your own work, and not copied by or from another student or from  books etc. If you use ideas, quotes or data (such as diagrams) from books, journals or other sources, you must reference your sources, using the Harvard style. Make sure that you know how to reference properly, and that understand the guidelines on plagiarism. *If you do not, you definitely get fail* |
| **Assignment Brief and Guidance:** |
| **Scenario**: You have recently joined a software development company to help improve their documentation of their in-houses software libraries which were developed with very poor documentation. As a result, it has been very difficult for the company to utilise their code in multiple projects due to poor documentation. Your role is to alleviate this situation by showing the efficient of UML diagrams in OOAD and Design Patterns in usages.  **Tasks**  You and your team need to explain characteristics of Object-oriented programming paradigm by applying Object-oriented analysis and design on a given (assumed) scenario. The scenario can be small but should be able to presents various characteristics of OOP (such as: encapsulation, inheritance, polymorphism, override, overload, etc.).  The second task is to introduce some design patterns (including 3 types: creational, structural and behavioral) to audience by giving real case scenarios, corresponding patterns illustrated by UML class diagrams.  To summarize, you should analyze the relationship between the object-orientated paradigm and design patterns.  The presentation should be about approximately 20-30 minutes and it should be summarized of the team report. |

|  |  |  |
| --- | --- | --- |
| **Learning Outcomes and Assessment Criteria** | | |
| **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. |

Table of Contents

[Unit 20: Advanced Programming ASSIGNMENT 1 1](#_Toc95989900)

[Table of Contents 1](#_Toc95989901)

[Introduction 2](#_Toc95989902)

[P1 Examine the characteristics of the object-orientated paradigm as well as the various class relationships. 2](#_Toc95989903)

[1. Introduction about OOP (Object-orientated paradigm) 2](#_Toc95989904)

[2. The characteristics of the OOP 3](#_Toc95989905)

[2.1. Class and Objects 3](#_Toc95989906)

[2.2. Inheritance 4](#_Toc95989907)

[2.3. Polyporphism 5](#_Toc95989908)

[2.4. Encapsulation 6](#_Toc95989909)

[2.5. Abstraction 7](#_Toc95989910)

[P2 Design and build class diagrams using a UML tool. 7](#_Toc95989911)

[1. Introduction about scenario 7](#_Toc95989912)

[2. Use-case diagrams 8](#_Toc95989913)

[3. Class diagrams 9](#_Toc95989914)

[4. Pseudo-code 9](#_Toc95989915)

[4.1. Select menu function 9](#_Toc95989916)

[4.2. Add new student 10](#_Toc95989917)

[4.3. Delete student 10](#_Toc95989918)

[4.4. Show all information of students 11](#_Toc95989919)

[4.5. Find student by ID 11](#_Toc95989920)

[5. Activity Diagrams 12](#_Toc95989921)

[5.1. Add new student 12](#_Toc95989922)

[5.2. Delete student 13](#_Toc95989923)

[5.3. Show students 14](#_Toc95989924)

[5.4. Find student 15](#_Toc95989925)

[**5.5.** Exit system 16](#_Toc95989926)

[CONCLUSION 17](#_Toc95989927)

[REFERENCES 18](#_Toc95989928)

**ASSIGNMENT 1 ANSWERS**

# Introduction

The next report will cover the fundamentals of OOP as well as a scenario for implementing an OOP product. The report also uses diagrams such as use-case and class diagrams to prepare for implementation in order to successfully design an OOP project. Learn about OOP design patterns as well as Design Pattern ideas including creation patterns, structure, behavior, and finally the integration of Design Patterns and OOP.

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

1. **Introduction about OOP (Object-orientated paradigm)**

Object-oriented programming (OOP) is a basic programming paradigm that almost every developer has utilized at some point in their career. The most prevalent programming paradigm is object-oriented programming (OOP), which is taught as the normal way to code for the majority of a programmer's educational career.

Object Oriented Programming (OOP) is a programming paradigm that uses classes and objects. It's utilized to break down a software program into reusable code blueprints (called classes) that may be used to build specific instances of things. Object-oriented programming languages include JavaScript, C++, Java, and Python, to name a few.

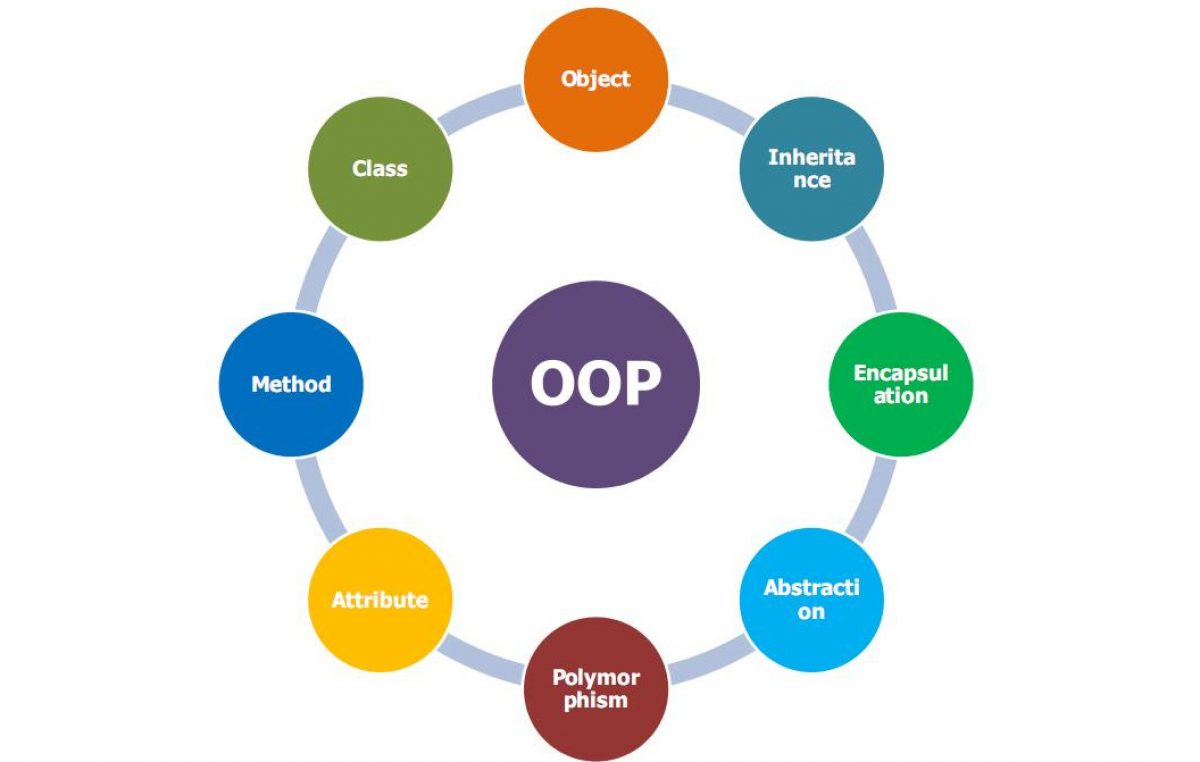


Figure – Ojbect-oriented paradigm

1. **The characteristics of the OOP**
2. Class and Objects

A programming language object is analogous to a real-world item. Object-Oriented Programming (OOP) is a programming paradigm in which programs are built using collections of classes and objects to make program development and maintenance easier.

A class is a logical grouping of objects that are similar in nature. In C#, it is one of the most basic types. It's essentially a data structure made up of Methods, Functions, and Fields. It specifies how dynamic instances, or objects that must be created for the class, should be defined. For instance, if the class is Apple, the object is the iPhone, which has several different attributes such as color, model, and price.



Figure 2 Class and Objects

1. Inheritance

It is a method that allows new classes to be generated by extending and enhancing the capabilities of existing classes. The base classes/parent classes/super-classes are the existing classes, while the derived classes/child classes/subclasses are the new classes.

If the super-class (es) allows it, the subclass can inherit or derive the super-properties class's and methods. A subclass may also add its own properties and methods, as well as alter any of the superclass's methods. A "is – a" connection is defined by inheritance.

For example, from the class Mammal, a number of classes such as Human, Cat, Dog, Cow, and so on can be deduced. Humans, cats, dogs, and cows all have distinct mammalian features. Furthermore, each has its own distinct qualities.

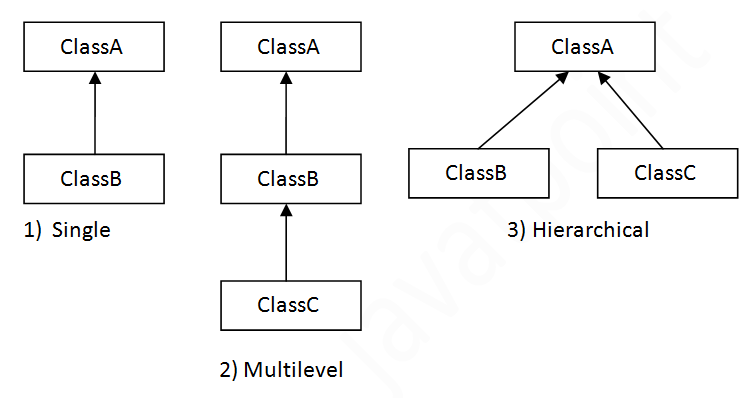


Figure – Inheritance in C#

1. Polyporphism

Polymorphism comes from the Greek word polymorphism, which meaning "capacity to take numerous forms." Polymorphism in the object-oriented paradigm refers to the use of operations in different ways depending on the instances they are applied to. Polymorphism enables things with various internal structures to share an external interface. When it comes to inheritance, polymorphism is especially useful.

Consider the following two classes: Circle and Square, both of which include a function findArea (). Although the names and purposes of the methods in the classes are the same, each class's internal implementation, i.e. the mechanism for computing an area, is different. When a Circle object calls its findArea() function, the process finds the circle's area without conflicting with the Square class's findArea() method.

In C#, there are two types of polymorphism:

* Static / Compile Time Polymorphism: In this case, the C# compiler evaluates the number of arguments supplied and the type of parameter before deciding which method to call, and if no matching method is found, it throws an error.
* Dynamic / Runtime Polymorphism: Late binding is another name for dynamic / runtime polymorphism. The method name and signature are shown here (number of parameters and parameter type must be the same and may have a different implementation). Dynamic polymorphism is exemplified through method overriding.

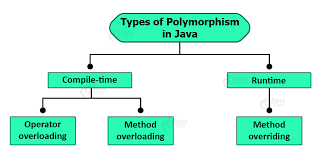


Figure – Polymorphism in C#

1. Encapsulation

The process of encapsulating both properties and methods within a class is known as encapsulation. The internal information of a class can be hidden from the outer world through encapsulation. It allows only the interface supplied by the class to be used to access the elements of the class from the outside.

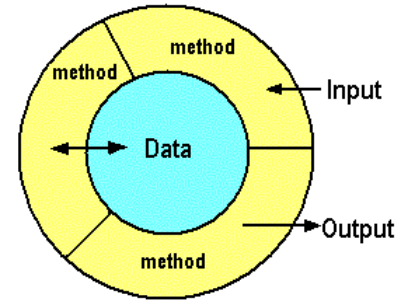


Figure 4 – Encapsulation in C#

1. Abstraction

Encapsulation allows a programmer to construct the necessary amount of abstraction while abstraction allows making relevant information visible.

Abstract classes in C# can be used to accomplish abstraction. Abstract classes are created in C# and used to offer a partial class implementation of an interface. When a derived class inherits from it, the implementation is complete. Abstract classes have abstract methods, which the derived class implements. The functionality of the derived classes is more specific.

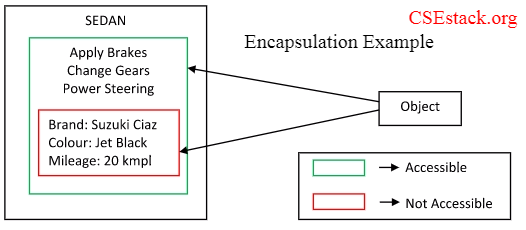


Figure - Abstraction in C#

# P2 Design and build class diagrams using a UML tool.

1. **Introduction about scenario**

Tripernex - This year, a new school opened, and it needed a way to manage the new kids. When it comes to managing student and instructor information, the technique will make the headmaster's job considerably easier. Users can gain access to their data by first submitting their names, ids, and other personal information. The goal of this paper is to offer a management system that will help the school better manage its members, such as teachers and pupils. As a result, this system should be able to do a wide range of basic data-driven tasks, such as adding, deleting, showing, and finding students by id. The following features will be available in the system:

* + The manager has the option of deleting pupils at the end of the course.
  + The management can enter information about the student, such as name, age, residence, ID, gender, and so on.
  + Each year, only the management has the authority to add new pupils.
  + The user can use the display functions to look up someone's information. After inputting the id, all non-confidential information, such as name, age, and class, will be displayed.
  + When the show function is used, the system displays all of the user's non-secret information.

1. **Use-case diagrams**

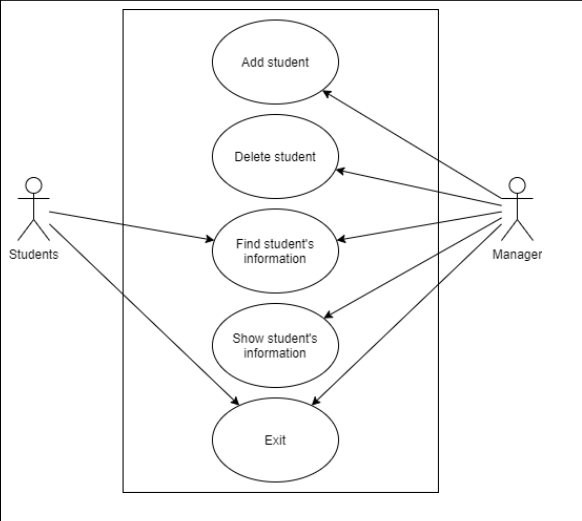


Figure - Use-case diagram

1. **Class diagrams**

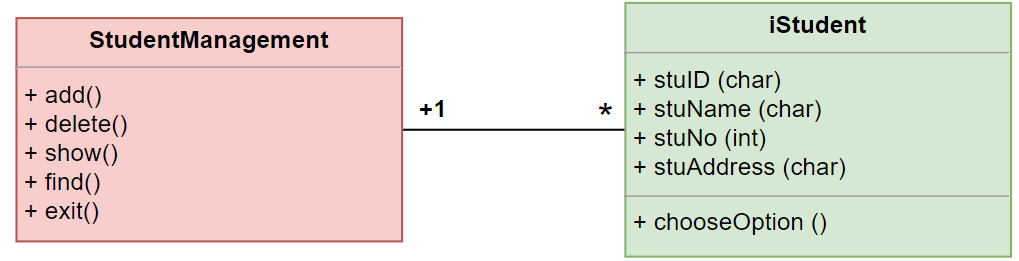


Figure - Class diagrams

1. **Pseudo-code**
2. Select menu function

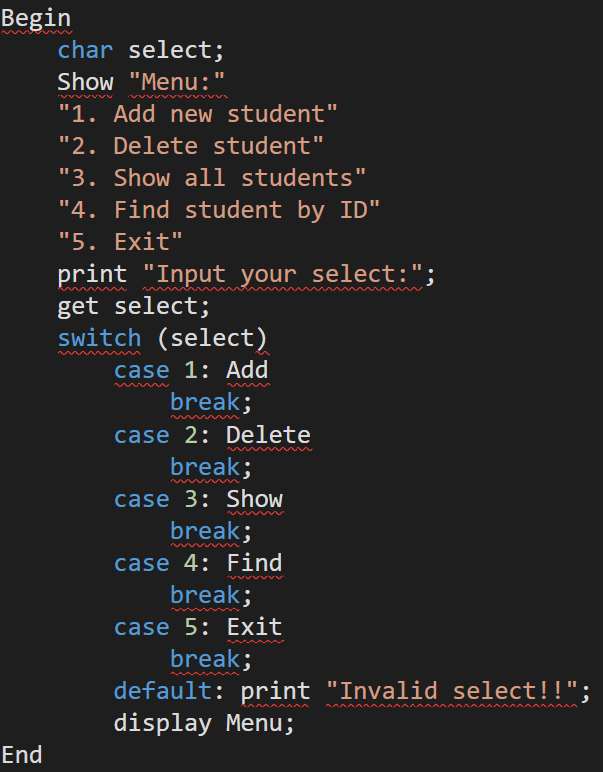


Figure - Select menu

1. Add new student

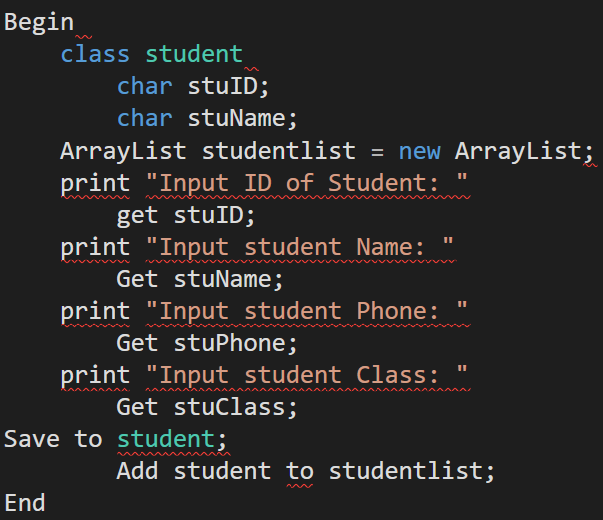


Figure - Add new student

1. Delete student

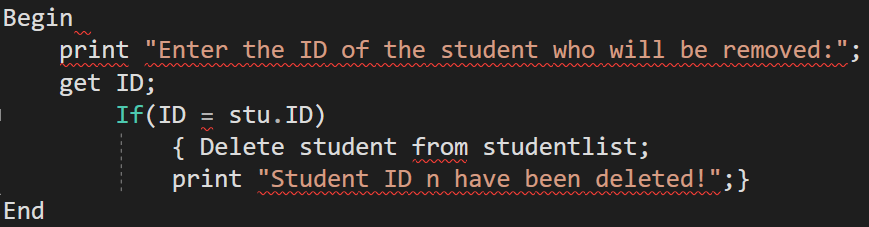


Figure - Delete Student

1. Show all information of students

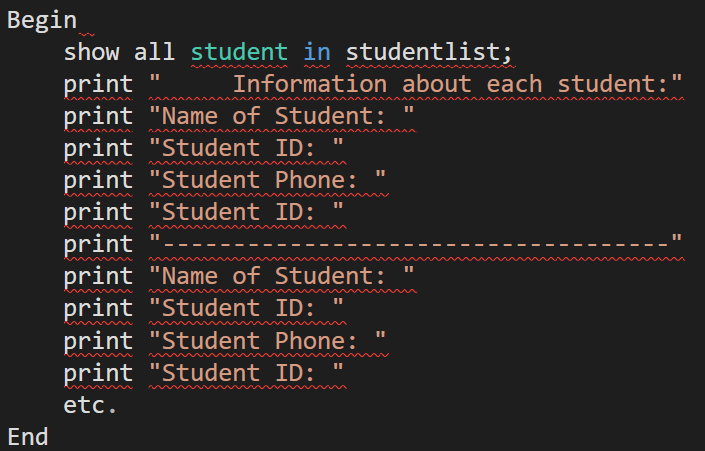


Figure - Show all students

1. Find student by ID

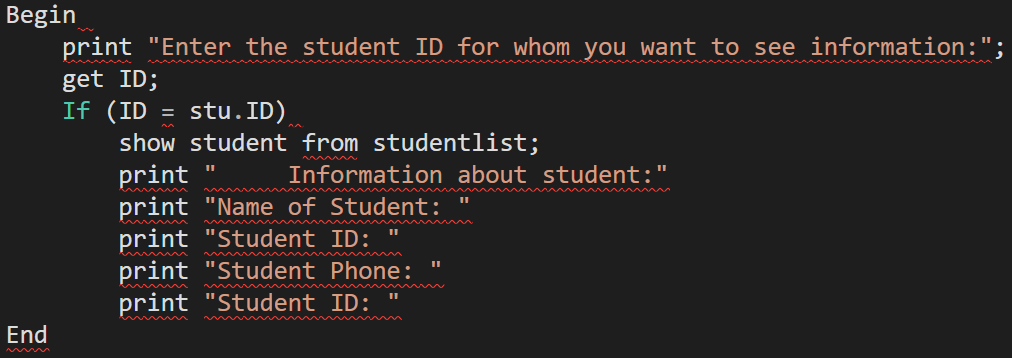


Figure - Find by ID

1. **Activity Diagrams**
2. Add new student

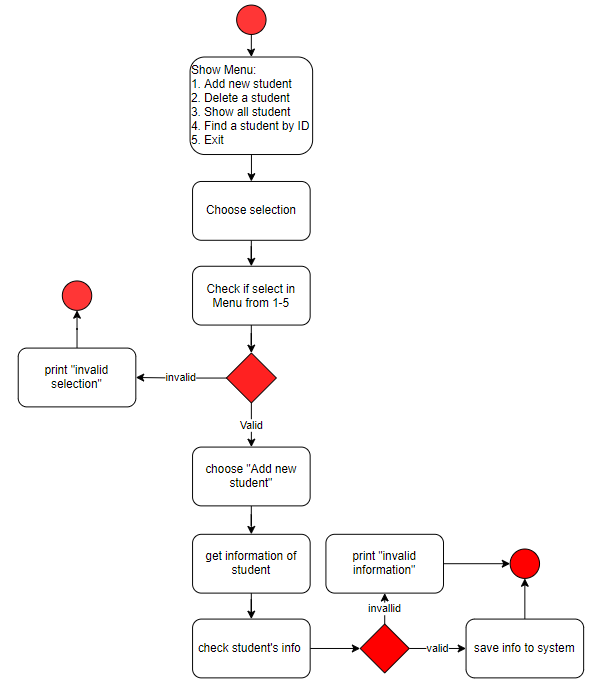


Figure - Add new diagram

1. Delete student

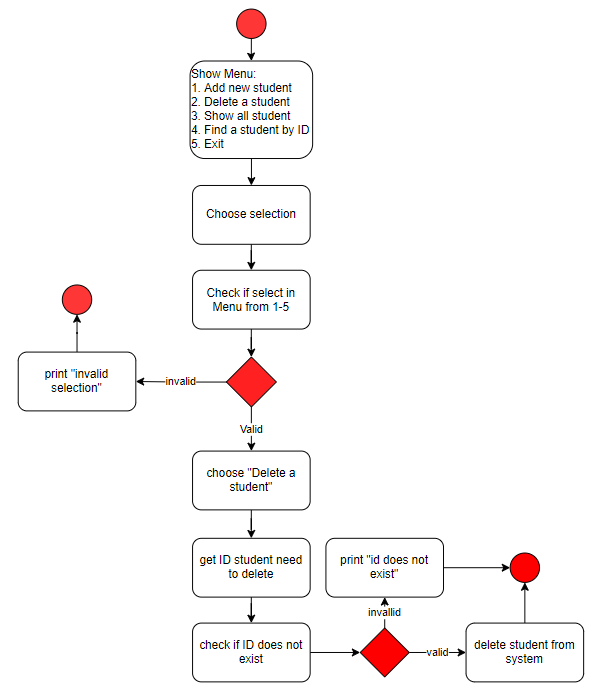


Figure - Delete one diagram

1. Show students

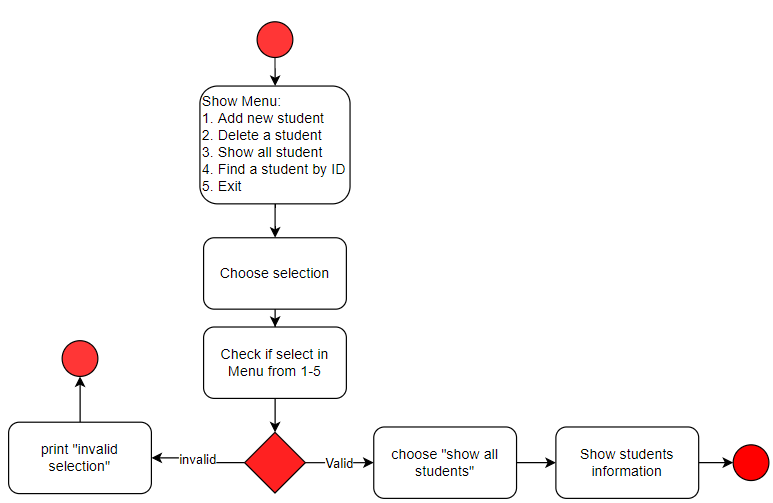


Figure - Show all diagram

1. Find student

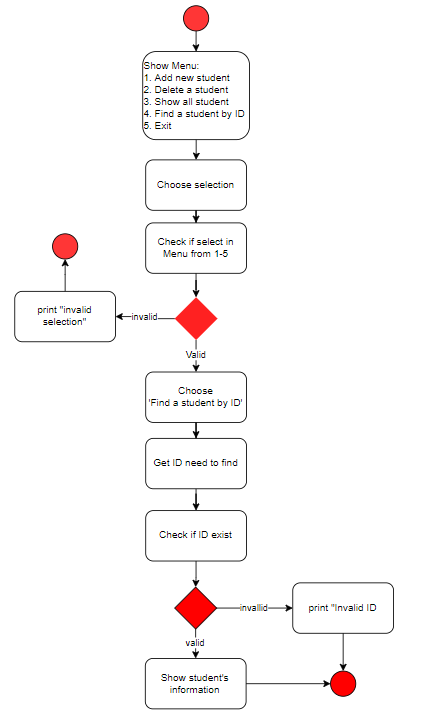


Figure - Find by ID diagram

1. Exit system

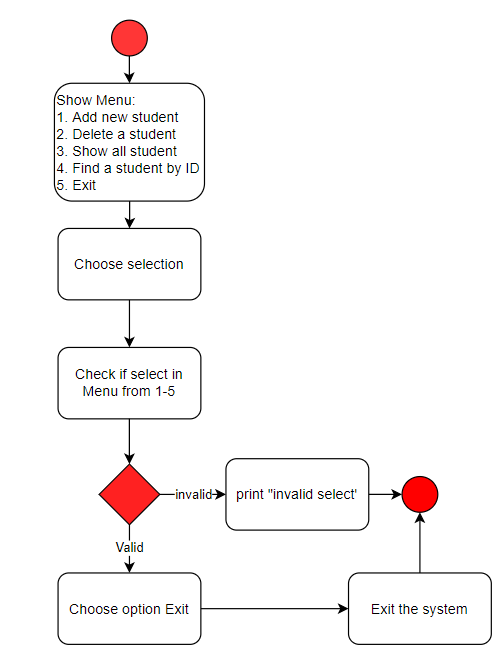


Figure - Exit system diagram

# CONCLUSION

After reading the above-mentioned report, it is clear that all of the implementation criteria have been met. All citation concepts are supported by reputable references and pictures so that readers may quickly grasp the content of the study. Users can easily examine the use case and class diagrams applied on applications like Draw.io.

# REFERENCES

Vidya, V, A (2021) Object Oriented Programming Using C# .NET [online] Available at: <https://www.c-sharpcorner.com/UploadFile/84c85b/object-oriented-programming-using-C-Sharp-net/> [Accessed at: 14 Feb.2022]

Tutorialspoint () What is abstraction in C#? [online] Available at: <https://www.tutorialspoint.com/What-is-abstraction-in-Chash> [Accessed at: 14 Feb.2022]

Tutorialspoint () OOAD - Object Oriented Paradigm [online] Available at: <https://www.tutorialspoint.com/object_oriented_analysis_design/ooad_object_oriented_paradigm.htm> [Accessed at: 14 Feb.2022]

Tutorialspoint () Object-Oriented Paradigm [online] Available at: <https://www.tutorialspoint.com/software_architecture_design/object_oriented_paradigm.htm> [Accessed at: 14 Feb.2022]

Samual, S (2018) Association, Composition and Aggregation in C# [online] Available at: <https://www.tutorialspoint.com/Association-Composition-and-Aggregation-in-Chash> [Accessed at: 14 Feb.2022]

Koderhq () C# Composition Tutorial [online] Available at: <https://www.koderhq.com/tutorial/csharp/oop-composition/> [Accessed at: 14 Feb.2022]

Jignest, T (2019) Understanding Polymorphism In C# [online] Available at: <https://www.c-sharpcorner.com/UploadFile/ff2f08/understanding-polymorphism-in-C-Sharp/> [Accessed at: 14 Feb.2022]

Jignest, T (2019) Association, Aggregation and Composition [online] Available at: <https://www.c-sharpcorner.com/UploadFile/ff2f08/association-aggregation-and-composition/> [Accessed at: 14 Feb.2022]

Jignest, T (2019) Association, Aggregation and Composition [online] Available at: <https://www.c-sharpcorner.com/UploadFile/ff2f08/association-aggregation-and-composition/> [Accessed at: 14 Feb.2022]

Geeksforgeeks (2021) C# | Abstract Classes [online] Available at: <https://www.geeksforgeeks.org/c-sharp-abstract-classes/> [Accessed at: 14 Feb.2022]

Geeksforgeeks (2021) C# | Abstract Classes [online] Available at: <https://www.geeksforgeeks.org/c-sharp-abstract-classes/> [Accessed at: 14 Feb.2022]

Geeksforgeeks (2020) C# | Class and Object [online] Available at: <https://www.geeksforgeeks.org/c-sharp-class-and-object/> [Accessed at: 14 Feb.2022]

Geeksforgeeks (2020) C# | Class and Object [online] Available at: <https://www.geeksforgeeks.org/c-sharp-class-and-object/> [Accessed at: 14 Feb.2022]

Erin, D (2020) What is Object Oriented Programming? OOP Explained in Depth [online] Available at: <https://www.educative.io/blog/object-oriented-programming> [Accessed at: 14 Feb.2022]

Erin, D (2020) What is Object Oriented Programming? OOP Explained in Depth [online] Available at: <https://www.educative.io/blog/object-oriented-programming> [Accessed at: 14 Feb.2022]

Dotnettutorials () Dependency Injection in C# [online] Available at: <https://dotnettutorials.net/lesson/dependency-injection-design-pattern-csharp/> [Accessed at: 14 Feb.2022]

Atul, R () Dependency Injection in C# [online] Available at: <https://www.c-sharpcorner.com/UploadFile/85ed7a/dependency-injection-in-C-Sharp/> [Accessed at: 14 Feb.2022]

Atul, R () Dependency Injection in C# [online] Available at: <https://www.c-sharpcorner.com/UploadFile/85ed7a/dependency-injection-in-C-Sharp/> [Accessed at: 14 Feb.2022]

Arjun, T (2018) Composition vs Aggregation in C# [online] Available at: <https://www.tutorialspoint.com/composition-vs-aggregation-in-chash> [Accessed at: 14 Feb.2022]

Arjun, T (2018) Composition vs Aggregation in C# [online] Available at: <https://www.tutorialspoint.com/composition-vs-aggregation-in-chash> [Accessed at: 14 Feb.2022]