**ASSIGNMENT 2 FRONT SHEET**

|  |  |  |  |
| --- | --- | --- | --- |
| **Qualification** | **BTEC Level 5 HND Diploma in Computing** | | |
| **Unit number and title** | Unit 20: Advanced Programming | | |
| **Submission date** | 24/8/2023 | **Date Received 1st submission** |  |
| **Re-submission Date** |  | **Date Received 2nd submission** |  |
| **Student Name** | Nguyen Anh Trinh | **Student ID** | GCC210105 |
| **Class** | GCC1001 | **Assessor name** | Tran Thi Kim Khanh |
| **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**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P3 | P4 | M3 | M4 | D3 | D4 |
|  |  |  |  |  |  |

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

**ASSIGNMENT 2 BRIEF**

|  |  |  |  |
| --- | --- | --- | --- |
| **Qualification** | **BTEC Level 5 HND Diploma in Computing** | | |
| **Unit number and title** | Unit 2: Advanced Programming | | |
| **Assignment title** | Application development with class diagram and design patterns | | |
| **Academic Year** | 2018-2019 | | |
| **Unit Tutor** | Doan Trung Tung | | |
| **Issue date** | 30 April 2019 | **Submission date** | 11 May 2019 |

|  |
| --- |
| **Submission Format:** |
| *Format:* The submission is in the form of an individual 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/> together with zipped project files.  *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**: (continued from Assignment 1) Your team has shown the efficient of UML diagrams in OOAD and introduction of some Design Patterns in usages. The next tasks are giving a demonstration of using OOAD and DP in a small problem, as well as advanced discussion of range of design patterns.  **Tasks:**  Your team is now separated and perform similar tasks in parallel. You will choose one of the real scenarios that your team introduced about DP in previous phase, then implement that scenario based on the corresponding class diagram your team created. You may need to amend the diagram if it is needed for your implementation. In additional, you should discuss a range of DPs related / similar to your DP, evaluate them against your scenario and justify your choice.  In the end, you need to write a report with the following content:   * A final version of the class diagram based on chosen scenario which has potential of using DP. * Result of a small program implemented based on the class diagram, explain how you translate from design diagram to code. * Discussion of a range of DPs related / similar to your DP, evaluate them against your scenario and justify your choice (why your DP is the most appropriate in that case). |
|  |

|  |  |  |
| --- | --- | --- |
| Learning Outcomes and Assessment Criteria | | |
| Pass | Merit | Distinction |
| **LO3** Implement code applying design patterns | | |
| **P3** Build an application derived from UML class diagrams. | **M3** Develop code that implements a design pattern for a given purpose. | **D3** Evaluate the use of design patterns for the given purpose specified in M3. |
| **LO4** Investigate scenarios with respect to design patterns | | |
| **P4** Discuss a range of design patterns with relevant examples of creational, structural and behavioral pattern types. | **M4** Reconcile the most appropriate design pattern from a range with a series of given scenarios. | **D4** Critically evaluate a range of design patterns against the range of given scenarios with justification of your choices. |

Table of Contents

[I. Introduction 6](#_Toc143537391)

[II. Scenario analysis 6](#_Toc143537392)

[1. Scenario 6](#_Toc143537393)

[2. Diagram 7](#_Toc143537394)

[III. Implementation 10](#_Toc143537395)

[1. Code 10](#_Toc143537396)

[2. Program screenshots 33](#_Toc143537397)

[3. Test Plan 36](#_Toc143537398)

[IV. Discussion 41](#_Toc143537399)

[References 58](#_Toc143537400)

# Introduction

In this report, I will present the use cases, class diagrams that have been changed. give the code to build this system. Provide test plans of the system. In addition, the article also shows Design Patterns used in the system

I have the task of designing and building the zoo management system and application. The system has full functions of adding, updating, deleting, and searching to ensure the functions used for the management of a zoo.

# Scenario analysis

## Scenario

A company named ABC, they are a management and nurturing company for animals. They are currently facing difficulties in managing the animals in their zoo. They are struggling with the management of the animals that are present in their park. They are having trouble in managing the incoming animals and the ones whose information is altered or removed from the zoo. They have asked me to design a system to manage the animals in the zoo. The animals will be divided into 2 classes, which are wild animals and rare species, for easy management. Within these classes, each species will have its own specific characteristics and attributes

## Diagram

* ***Use case diagram***

A diagram of a diagram

Description automatically generated

In this use case there is but change as follows:

- The BackToMain is used in the menus. The purpose is for users to easily return to the main menu for management.

- The Exits are used in menus. The purpose of the user is to easily exit the system when the user wants to end the program

- One more function is the function to show all animals of that type of animal

- In use case I have include lines but because I didn't make it in time, in my post I don't do it to that extent

* ***Class diagram***

A diagram of a computer

Description automatically generated with medium confidence

In my class diagram there are many changes to complete my system.

- Added 2 classes: IMenuFactory class and GetMenuFactory class. Purpose to build a menu system using Factory pattern.

- Change parameter to function SelectMenu(): the added parameter is a zoo class because the zooo class contains a list to store information about animals when passing parameters so it will be easy to interact with the function add edit delete and list

- Changed the Search function parameter in the CRUDZoo interface: instead of the parameter being of type int , I changed the parameter type to string type and add the parameter type to string type to make it easier to find animals in the list.

- Add a new function ShowAllforCatetory() to search for animals of that species

# Implementation

## Code

* ***These are classes to contain the attributes and behaviour of a zoo animal***

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

* Explain: This is the Animal class, which is an abstract class. This class contains the attributes of the animals in the zoo

A screen shot of a computer

Description automatically generated

* Explain : This is a wild animal class, this class is also an abstract class, this class inherits the properties of the Animal class, and this class has a private property that is the origin of the animal

A screen shot of a computer program

Description automatically generated

* Explain : This is a Rare animal class, this class is also an abstract class, this class inherits the properties of the Animal class, and this class has a private property that is the rarity level of the animal
* ***This is the lion class: this class contains information about the lion as well as an instantiation of a lion object***

A screen shot of a computer program

Description automatically generated

* ***This is the Kangaroo class: this class contains information about the Kangaroo as well as an instantiation of a Kangaroo object***

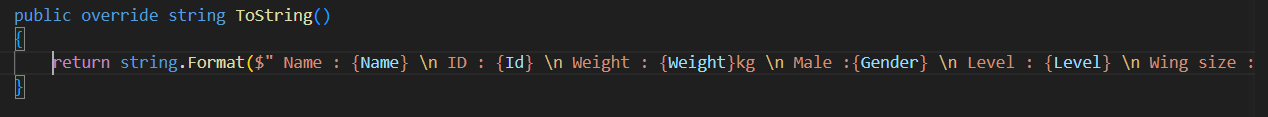
A screen shot of a computer program

Description automatically generated

* ***This is the Lophura Diardi class: this class contains information about the Lophura Diardi as well as an instantiation of a Lophura Diardi object***

A screen shot of a computer program

Description automatically generated



* ***This is the Trimeresurus Monticola class: this class contains information about the Trimeresurus Monticola as well as an instantiation of a Trimeresurus Monticola object***

A screen shot of a computer

Description automatically generated

* ***These classes are responsible for creating a management menu for the system***

A black rectangle with white text

Description automatically generated

* Explain : This interface creates a function to call the menu classes

A screen shot of a computer program

Description automatically generated

* Explain : This class inherits the IMenuFactory interface, and redefines the abstract function contained in the interface

A black rectangle with white text

Description automatically generated

* Explain : IMenuu contains abstract functions GetType(), ShowMenu(), SelectMenu(Zoo z).
* GetType(): this function's purpose is to return a string, when the user completes an action, it will return the menu that the user wants.
* ShowMenu(): this function is used to print the options available in the menu
* SelectMenu(Zoo z): This function is used to record the user's request and the Zoo parameter is used to call functions on the Zoo class.
* ***This is the class used to initialize the main menu of the application***

A screen shot of a computer program

Description automatically generated

A black rectangle with white text

Description automatically generated

* ***This is the class used to initialize the Wilds menu of the application***

A computer screen shot of a program code

Description automatically generated

A black rectangle with white text

Description automatically generated

* ***This is the class used to initialize the Rare menu of the application***

A screen shot of a computer program

Description automatically generated

A computer screen shot of text

Description automatically generated

* ***This is the class used to initialize the LionMenu menu of the application***

A screen shot of a computer code

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer screen

Description automatically generated

* ***This is the class used to initialize the KangarooMenu menu of the application***

A screen shot of a computer code

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

* ***This is the class used to initialize the TrimeresurusmonticolaMenu menu of the application***

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

* ***This is the class used to initialize the LophuradiardiMenu menu of the application***

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

* ***These are the classes I use to build system functions***

A computer screen shot of code

Description automatically generated

A screen shot of a computer

Description automatically generated

* ***This function has the function of adding a new animal to the zoo's list***

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

* ***This function is used to delete an animal in the zoo by the animal's ID***

A black screen with white text

Description automatically generated

* ***This function is used to search for an animal by its ID***

A screen shot of a computer program

Description automatically generated

* ***This function is used to update the new information of the animal according to the animal's ID***

A computer screen shot of a program code

Description automatically generated

* ***This function is used to print the animals of that species***

A computer screen with text on it

Description automatically generated

* ***Program to initialize the system and use the system***

A screen shot of a computer program

Description automatically generated

* I used Singleton Pattern in my Zoo, because it is a single class that is not inherited by other classes and is only instantiated once.
* Based on this pattern, I applied it with private constructor Zoo() and static \_instance field for my zooo class
* I used the Factory Pattern in my IMenuu, because the Factory pattern is suitable to use for building a menu.
* Based on this pattern, I applied it to create GetType(),SelectMenu() and ShowMenu() functions for my system.

## Program screenshots

* Step 1 : Main Menu

A black screen with blue text

Description automatically generated

* Step 2: when choosing option 1 , the program will switch to Wilds menu

A black screen with blue lines

Description automatically generated

* Step 3 : when choosing option 1 , the program will switch to Lion menu. Enter lion information

A computer screen with text on it

Description automatically generated

* Step 4 : when choosing option 1 , the program will switch to Lion menu. Enter lion information

A computer screen with white text

Description automatically generated

* Step 5 : when choosing 5, the program prints out the information of all the lions in the zoo

A black rectangle with white dots

Description automatically generated

* Step 6: when choosing 4, the program prints out all the lion's information according to that lion's ID

A black rectangle with white dots

Description automatically generated

* Step 7: when selecting 2, and enter the lion ID that you want to update. Enter the required code and new information

A black rectangle with white dots

Description automatically generated

* Step 8: when selecting 3, and enter the lion ID that you want to remove from the zoo

A black rectangle with white dots

Description automatically generated

* Step 9: when 5 is selected, the program prints out all the animals to check for updates and delete the animal

A black rectangle with white dots

Description automatically generated

## Test Plan

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| No | TEST CASE | FUNCTION | INPUT DATA | EXPECTED OUTPUT | ACTUAL OUTPUT | RESULT |
| 1 | Verify that when the user adds a new lion | AddNew() | Bob L00 100 true usa 500 | Successful | Successful | Pass |
| 2 | Verify that when user adds a new lion and check if that lion id is already in the list | AddNew() | Nigella L00 110 true usa 600 | ID is already | Successful | Fail |
| 3 | Verify that  when the user searches for a lion in the list by ID | Search() | L00 | Name : Bob  ID : L00  Weight : 100kg  Male : true  Origin : Usa  Maul :500cm | Name : Bob  ID : L00  Weight : 100kg  Male : true  Origin : Usa  Maul :500cm | Pass |
| 4 | Verify that when the user updates the Lion information already in the list | Update() | Enter your ID: L00  Name poppy | Update Successful | Update Successful | Pass |
| 5 | Verify that when the user wants to delete an existing lion in the list | Remove() | Enter your ID: L00 | Poppy removed from the zoo. | Poppy removed from the zoo. | Pass |
| 6 | Verify that when the user wants to update a lion that is not already in the list | Update() | Enter your ID: L00 | Not Found | The system asks the user to enter the information they want to update | Fail |
| 7 | Verify that when the user wants to delete a lion that is not already in the list | Remove() | Enter your ID: L00 | Not Found | Not Found | Pass |
| 8 | Verify that when the user adds a new Kangaroo | AddNew() | Nick K00 50 true Australia 500 | Successful | Successful | Pass |
| 9 | Verify that  when the user searches for a Kangaroo in the list by ID | Search() | K00 | Name : Nick  ID : K00  Weight : 50kg  Male : true  Origin : Australia  Maul :500cm | Name : Nick  ID : K00  Weight : 50kg  Male : true  Origin : Australia  Maul :500cm | Pass |
| 10 | Verify that when the user updates the Kangaroo information already in the list | Update() | Enter your ID: K00  Weight 55 | Update Successful | Update Successful | Pass |
| 11 | Verify that when the user wants to delete an existing Kangaroo in the list | Remove() | Enter your ID: K00 | Nick removed from the zoo. | Nick removed from the zoo. | Pass |
| 12 | Verify that when the user wants to delete a Kangaroo that is not already in the list | Remove() | Enter your ID: K00 | Not Found | Not Found | Pass |
| 13 | Verify that when the user adds a new Lophura Diardi | AddNew() | Harry lop00 2 true g3 100 | Successful | Successful | Pass |
| 14 | Verify that  when the user searches for a Lophura Diardi in the list by ID | Search() | lop00 | Name : Harry  ID : lop00  Weight : 2kg  Male : true  Level : g3  Wing size : 100cm | Name : Harry  ID : lop00  Weight : 2kg  Male : true  Level : g3  Wing size : 100cm | Pass |
| 15 | Verify that when the user updates the Lophura Diardi information already in the list | Update() | Enter your ID: lop00  Name Garen | Update Successful | Update Successful | Pass |
| 16 | Verify that when the user wants to delete an existing Lophura Diardi in the list | Remove() | Enter your ID: lop00 | Garen removed from the zoo. | Garen removed from the zoo. | Pass |
| 17 | Verify that when the user wants to delete a Lophura Diardi that is not already in the list | Remove() | Enter your ID: lop00 | Not Found | Not Found | Pass |
| 18 | Verify that when the user adds a new Trimeresurus Monticola | AddNew() | Cassiopeia t00 1 false g4 true | Successful | Successful | Pass |
| 19 | Verify that  when the user searches for a Trimeresurus Monticola in the list by ID | Search() | lop00 | Name : Cassiopeia  ID : t00  Weight : 1kg  Male : true  Level : g4  Danger : High | Name : Cassiopeia  ID : t00  Weight : 1kg  Male : true  Level : g4  Danger : High | Pass |
| 20 | Verify that when the user updates the Trimeresurus Monticola information already in the list | Update() | Enter your ID: t00  Name Sol | Update Successful | Update Successful | Pass |
| 21 | Verify that when the user wants to delete an existing Trimeresurus Monticola in the list | Remove() | Enter your ID: t00 | Sol removed from the zoo. | Sol removed from the zoo. | Pass |
| 22 | Verify that when the user wants to delete a Trimeresurus Monticola that is not already in the list | Remove() | Enter your ID: t00 | Not Found | Not Found | Pass |

With 2 failed test cases. I checked and found a solution to this problem. With add, I check if the newly entered ID is already in the list. if there is, then the id is already there. With the update, I check if the newly entered ID is already in the list, if not, print out the message not found

# Discussion

* + ***Range of similar patterns***

I use many design patterns in building my system:

* + - Singleton and Factory pattern: these are 2 design patterns I use in my system. this is a design pattern of type Creational Design pattern. In Creational Design pattern, there are many different patterns, but there are 3 that I think are most suitable for my system: Singleton, Factory and Builder.

+ Singleton : A singleton is a design pattern in software engineering that restricts the instantiation of a class to a single instance and provides a global point of access to that instance. In other words, it ensures that a particular class has only one instance in the entire application and provides a way to access that instance from anywhere within the codebase. (tutorialspoint, tutorialspoint, 2022)

* ***Class Diagram***

A screenshot of a computer

Description automatically generated

* ***Code :***

A screenshot of a computer

Description automatically generated

+ Factory : A Factory pattern is a design pattern in software engineering that provides an interface for creating objects in a super class, but allows subclasses to alter the type of objects that will be created. The main idea behind this pattern is to centralize the creation of objects, promoting loose coupling and easier maintenance. (tutorialspoint, tutorialspoint, 2022)

* ***Class Diagram***

A screenshot of a diagram

Description automatically generated

* ***Code***

A black rectangle with white text

Description automatically generated

A screenshot of a computer program

Description automatically generated



A screenshot of a computer program

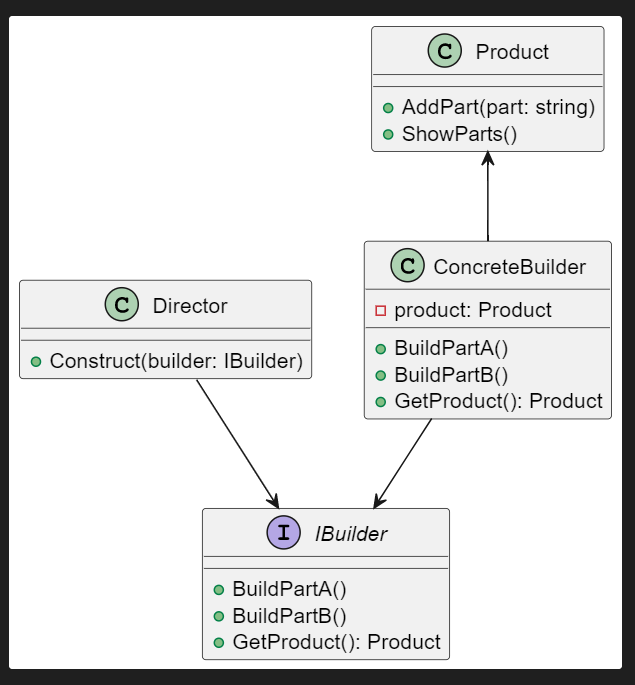
Description automatically generated

A screen shot of a computer

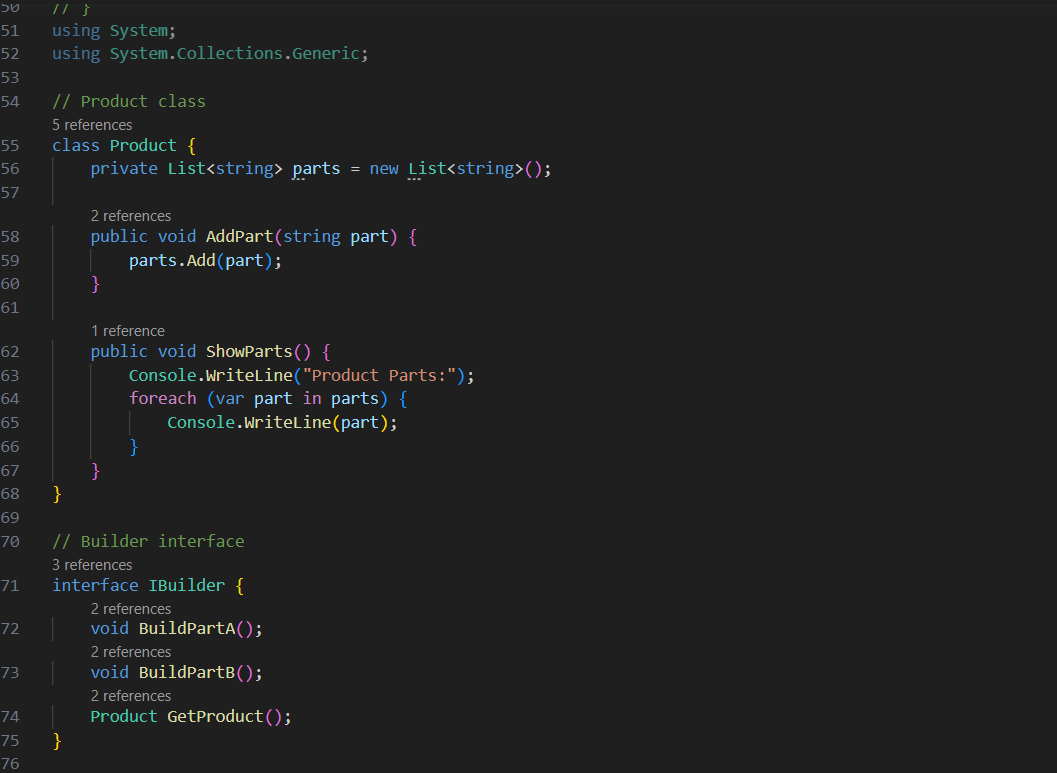
Description automatically generated

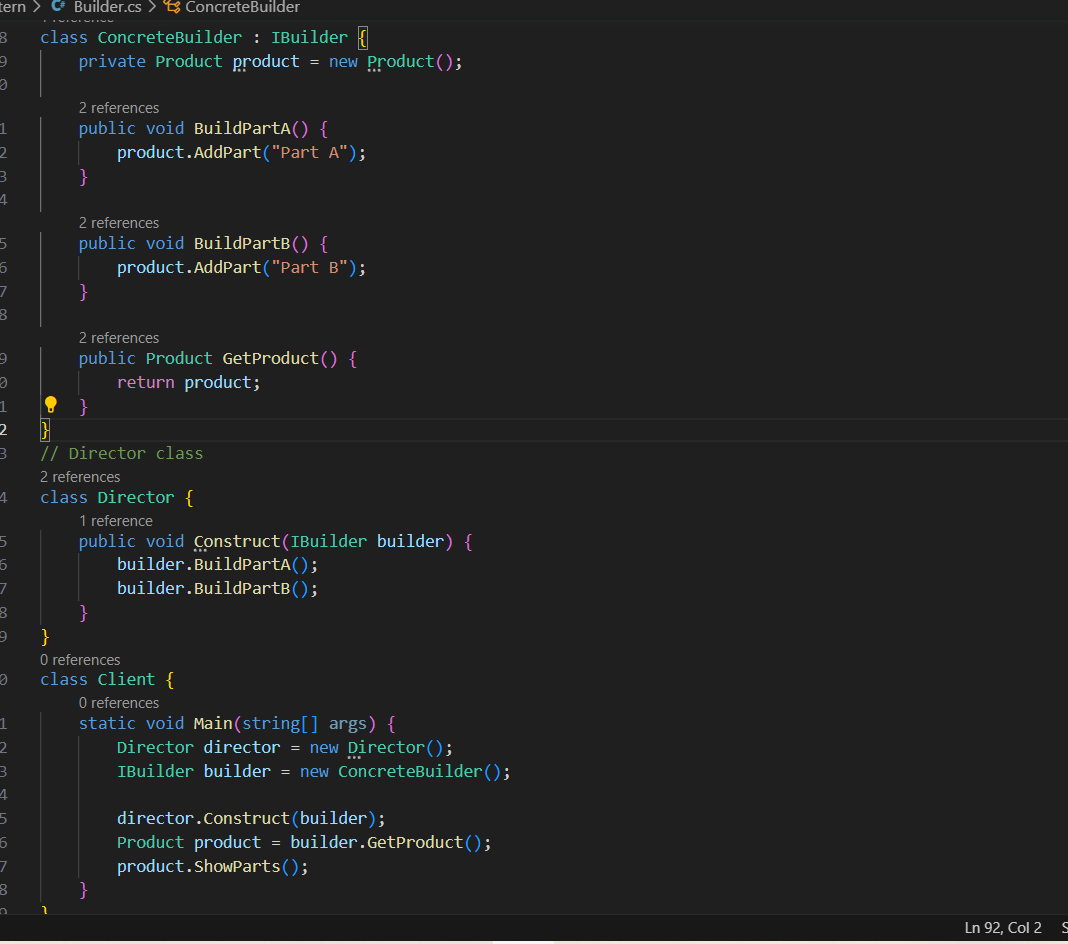
+ Builder pattern : The Builder pattern is a creational design pattern used in software engineering. It is used to construct complex objects step by step, allowing the separation of the construction process from the actual representation of the object. This pattern is particularly useful when an object has multiple parts or configurations, and you want to create different variations of the same object.

* ***Class Diagram***



* ***Code***





- Command pattern: this is one of many patterns in the structural design pattern. In the structural design pattern, there are commonly used types such as: Facade, composite, and command. and I think the above types will be suitable for my project

+ Composite : The Composite pattern is a design pattern in software engineering that allows you to compose objects into tree-like structures to represent part-whole hierarchies. It enables clients to treat individual objects and compositions of objects uniformly, meaning that a single object can stand alone or be composed of multiple objects in a nested fashion. (geeksforgeeks, 2022)

* ***Class Diagram***

A screenshot of a computer

Description automatically generated

* ***Code***

A computer screen shot of a program code

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

+ Command : The Command pattern is a design pattern in software engineering that encapsulates a request as an object, thereby allowing for parameterization of clients with different requests, queuing of requests, and logging of requests. It separates the sender (client) of a command from the receiver (the object that performs the action) and provides a way to decouple the two.

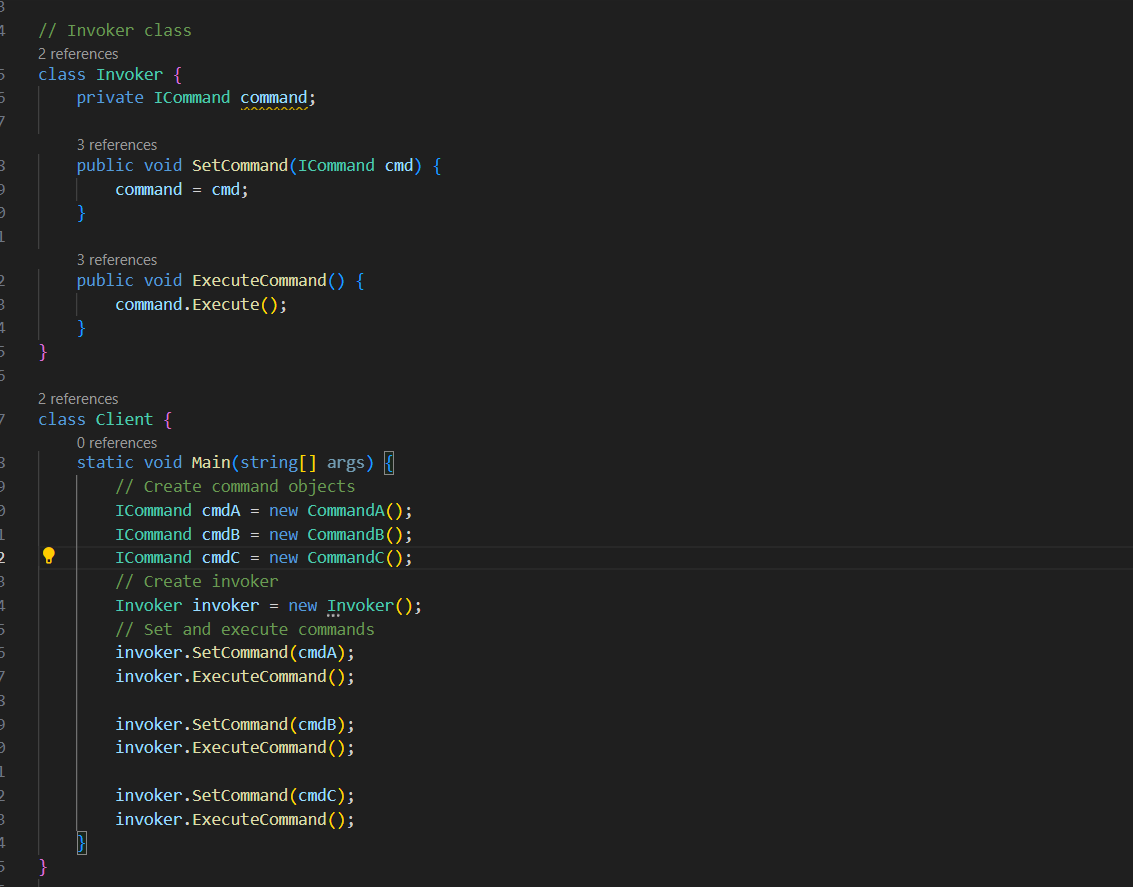
* ***Class Diagram***

A diagram of a computer

Description automatically generated

* ***Code***





+ Facade : The Facade pattern is a design pattern in software engineering that provides a simplified interface to a more complex subsystem or set of interfaces. It aims to hide the complexities of the underlying system and present a unified and easier-to-use interface to clients. (tutorialspoint, tutorialspoint, 2022)

* ***Class Diagram***

A screenshot of a computer

Description automatically generated

* ***Code***

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

* + ***Reconcile the most appropriate design pattern from a range with a series of given scenarios.***

|  |  |  |
| --- | --- | --- |
| Type | Design Pattern | When to user it |
| Creational Design Pattern | Factory Pattern | It's used to abstract the process of object creation and promote loose coupling between the client code and the object creation process |
| Creational Design Pattern | Singleton Pattern | It is used when you want to control the instantiation of a class to ensure that there's only one instance of the class throughout the application's lifecycle |
| Creational Design Pattern | Builder pattern | The Builder pattern finds application in software engineering when there's a requirement to construct intricate objects with numerous attributes or settings in an orderly and structured way. This pattern offers an approach to handle the creation of objects that involve a mix of mandatory and optional elements, all while permitting the creation of diverse versions of the eventual object. |
| Structural Design Pattern | Composite Pattern | It's used when you have objects that can be composed of other objects in a recursive manner and you want to treat individual objects and compositions of objects uniformly. |
| Structural Design Pattern | Command Pattern | This pattern is especially useful when dealing with hierarchical structures and you want to work with the individual components and the whole structure in a consistent way. |
| Structural Design Pattern | Facade Pattern | It's used to simplify the interactions between complex systems by providing a higher-level interface that encapsulates and hides the underlying complexity. |

(FLM, 2022)

# References

FLM. (2022). *FLM*. Retrieved from https://flm.greenwich.edu.vn/gui/role/student/SyllabusDetails?sylID=2543

geeksforgeeks. (2022). *geeksforgeeks*. Retrieved from https://www.geeksforgeeks.org/composite-design-pattern/

tutorialspoint. (2022). *tutorialspoint*. Retrieved from https://www.tutorialspoint.com/design\_pattern/singleton\_pattern.htm

tutorialspoint. (2022). *tutorialspoint*. Retrieved from https://www.tutorialspoint.com/design\_pattern/factory\_pattern.htm

tutorialspoint. (2022). *tutorialspoint*. Retrieved from https://www.tutorialspoint.com/design\_pattern/facade\_pattern.htm