Final assignment Spring- 2021-2022

ADVANCED PROGRAMMING

Name: Mahmoud Rumaneh

Id: 20120103

Date: June 20, 2022

Dr. Hamzah Asefan

CONTENTS

PART 1:

1.1 Object-Oriented Programming characteristics:

- **Encapsulation:** Encapsulation is one of the four concepts of OOP, which is the mechanism of wrapping variables and instructions (methods) that work together as a single unit. The idea of encapsulation is to hide data inside the class and when the variable is defined as private, it will disappear from the rest of the classes who cannot access this class through the methods in it. To achieve the encapsulation the variable must be in 4 situations: **public**, **protected**, **private**, **default** (Access Modifiers). **Public** keyword it's very common, because it allows any user to use it, even outside the place that he declared it in, it's ok if you don't use any other access modifier, but **protected** keyword we can use it in inheritance (parent-child relationship between objects) and the protected data structures can be used in subclass that are outside its package, **private** it works as the opposite of public keyword, you can use it outside the class that you declared in it, so you must provide it with setter and getter methods to show and display the value of the variable, the get is a call to the value of the variable and the set is the setting of the value of the variable, **default** doesn't have a keyword, so if you don't wite private or protected or public it will be default access modifier as default, if the data structure was default you won't can to use it in another package, but you can use it in the same package. [1]

- **Polymorphism:** Polymorphism is the ability to make an object take more than one form and in Java it is the ability to have different representations of the same method name and depends on the type of object that is defined in the method. Examples of Java Polymorphism are Method Overloading and Method Overriding. The advantages of OOP that it gives you the ability to do a single task in different ways makes you do many representations in one interface definition. **Overloading** is that you have in a class more than one method with the same name. **The Overriding** is that in the sub class it has a method that has the same name as another method in the super class. [2]

- **Constructor:** The Constructor is to create a method with the same class name and it is called the default constructor if there is no other constructor available in the same class then the Java compiler makes it default constructor by default. Constructor has two types, the **no-arg constructor** and the **parameterized constructor**, and every time I create an object with the new() keyword it will call at least one constructor. [3]

- **Abstraction:** The Abstract keyword is written in order to create the Abstract Class and it can also be created Abstract method which is a method without code and the Abstract class is not a requirement to contain only Abstract methods and also the Abstract method must always be redefined in the sub class and I cannot create an Abstract method with the same name of another Abstract method.[4]

- **Interface:** The Interface is a Full Abstract Method and interface in Java is a blueprint of a class, all the methods it contains are without code, all the methods will be public and abstract without writing the Abstract keyword, all variables in the interface are static and public by default, the naming convention is adjective and start with the uppercase letter and the keyword for calling the Interface is **implements**. [5]

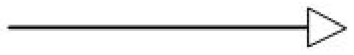
- **Collections:** The collections or containers is a framework that provides us with a structure to store a set of objects or variables in one place and has many interfaces such as (List, Queue, Dequeue) and has many classes such as (Array, ArrayList, Vector, LinkedList, PriorityQueue, HashMap, HashSet, LinkedHashSet, TreeSet).[6]

- **Array:** Array is one of the collections and stores similar elements in contiguous locations called indices, they are an object that contains elements of the same data type. The array in Java depends on the index, the first element is stored in the 0th index, the second element is stored in the 1st index, and so on. The advantage of the Array is that the code is optimized with it and we can retrieve or sort the data easily, also we can access the element in a random way which is specifying the index we want. As for its disadvantages, the size has a specific limit, the storage has a fixed size, it doesn't increase at runtime.[7]
- **ArrayList:** The ArrayList is a class that uses a dynamic array to store the elements, it is like the array but it does not have a specific size, we can add and delete on it at any time and we can build an object or a class from it. One of her features is can contain duplicated elements and save the arrangement of elements because it works by the index system. ArrayList is not synchronized meaning that many threads can work on it simultaneously.[8]
- **Vector:** The vector is dynamically compatible with an ArrayList, but differs in that it is synchronized, meaning that only one thread can work on the array, it has a lock on it, and another thread that wants to work has to wait until this lock is released, the ArrayList is faster because of this waiting process. Also, the way to increase the size differs between them, the ArrayList increases by 50%, and the Vector increases by 100%, which is a doubling of the current size of the Array.[9]
- **HashMap:** The HashMap is a class that represents a map interface that makes us store the key and its value and the key must be unique. If you try to repeat the key, the new element will be replaced by the old key element. It is easy to add and delete operations on the HashMap using the key index. It can contain one null key and multiple null values and the HashMap is not synchronized.[10]

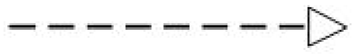
❖ **The Relationship Between Class and Objects:** The class is the main thing in the code and contains a set of methods and the object is the constructor or implementation of the schema built by the class, so the class is the schema and the object is the construct, the new operator is used to create an object of a class. When an object of a class is instantiated, the system specifies memory for every data in the class and the runtime is an instance of a class in memory. [11]

❖ **Static Keyword:** The static keyword in Java is used for memory management mainly, it's a class for attributes and methods, a common property of all objects that I can use by the name of a class without the need to create an object. We can apply the static keyword with variables, methods, blocks, and nested classes. The static keyword belongs to the class than an instance of the class. The static method only deals with static attributes and methods. its advantage is making your program memory efficient.[12]

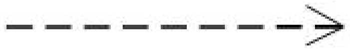
❖ **Generalization (Inheritance):** Generalization is the process of taking out common properties and functionalities from two or more classes and combining them together into another class which acts as the relationship is between the general thing, which is the superclass, and the specific thing, which is the subclass, it is expressed by "is a" and the relationship is expressed by a line and an arrow at the end of it.[13]



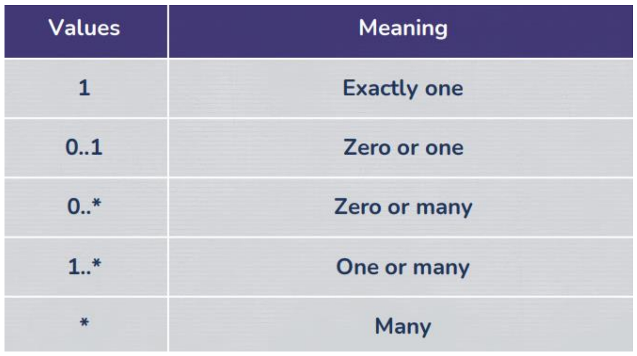
❖ **Realization:** Realization is a relationship between the blueprint class and the object containing its respective implementation level details. It represents the relationship between the interface and the class of implementation and the relationship is expressed by a broken line and an arrow at the end of it.[14]



- **Dependency:** A relationship represents the use of the class to another class, a change in structure or behavior of a class affects the other related class, then there is a dependency between those two classes. and the form of the relationship is that the class outputs a dashed arrow in the direction of the class it uses and is expressed by the word "using". [14]

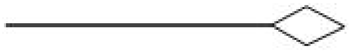


- **Association**: Association is a relation between two separate classes which establishes through their objects. A Binary Association is when the relationship is between two classes and Ternary Association is when the relationship is between more than two classes. It is expressed in a solid line only, and I can add to the line the Role and Multiplicity, the Role has clarified the relationship in it such that the student learns from the instructor and the instructor teaches the student, and Multiplicity that how many objects I can build in each class, one student can be taught by one or more instructors, and the instructor can teach one or more students. The relationships are as follows: [14]

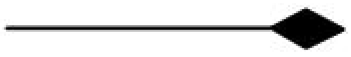


Values	Meaning
1	Exactly one
0..1	Zero or one
0..*	Zero or many
1..*	One or many
*	Many

- **Aggregation:** Aggregation is a kind of association. A directional association between objects. When an object 'has-a' another object, then you have got an aggregation between them. It is a relationship between the parent and the child in which the parent depends on the child but the child does not depend on the parent such that a car cannot run without tires but they can be the tires of another vehicle type such as a bicycle and the relationship is represented by the word "has a".[14]



- **Composition:** Composition is a kind of aggregation. In a more specific way, a restricted aggregation is called composition. It is a relationship in which the child cannot dispense with the parent and is represented by the word "part of" an example is that the heart, mind, and eyes are all parts of the body. [14]



1.2 Intuition, when to use, and disadvantages for design patterns:

Prototype: The prototype design pattern is one of the creational patterns, this pattern is used when the method of creating the object is complex and expensive. When the customer application must be insensible of making an object. When you must keep the number of classes to a minimum level, also the process of creating the object may need calculations, this pattern works by cloning an existing object instead of creating a new one, the newly copied object has the same properties as the source object and after cloning we can change the properties of the new object as required. [15]

The prototype has **advantages** such as adding or removing objects at runtime like adding a new concrete product class into the system easily by registering the client with a prototypical instance, which makes the code more flexible because the client will install and remove prototypes at run time. Cutting down the resource exhaustion to object creation. By defining new objects by different values, you will define new behavior through object structure by defining the values for the object's variables, not specifying by classes. It reduces the necessity for sub-classing. [16]

The prototype also has **disadvantages**, each subclass must implement the clone() method and this can be difficult when these classes are still being tested, and difficult to implement when they include objects that do not support copying or circular references, also the prototype has overkill in a project that uses count few of the objects or it already has a primary focus on the branches of the prototype threads, in addition, prototype hides the product classes from the user. [16]

Adapter: The adapter is a type of structural design pattern which is pattern that acts as a connector between two different interfaces that cannot be connected directly. The adapter wraps the existing class with a new interface until it becomes an interface that the client wants. The main reason behind using this pattern is to transform an existing interface and the client can't use it to an interface that the client wants, when the application isn't suitable with the interface that the client is expecting, and when you want to reuse old code in an application without making any change in the main code. [17]

The adapter has **advantages** like the reusability and flexibility, so the class can be accessed by many different systems using different adapters and interfaces, two classes have different interfaces can communicate with each other by the adapter, and the change in either the client interface or the adapter, it just means a change in the adapter. [18]

There are also **disadvantages** to the adapter, the frequent use of it may make the system as a whole very messy and difficult to understand by programmers, the adapter should not be used if it is not necessary to use it. But in java, it can only deal with at most one adapter because it inherits at most one class, so the target class must be abstract. When the adapter is used as an object it cannot override the method, it cannot reuse the adapter for another subclass, but when the adapter is used as a class it cannot be used as a subclass of the parent class. [18]

Iterator: The Iterator Pattern is common in Java and is used when the programmer wants a way to access certain collection elements sequentially according to the index of the element. The Iterator falls under the Behavioral Design Pattern and is implemented using Java Enumeration sometimes. The loop in general is such as for loop depends on the index, unlike the Iterator can use a setter and getter, and we can also delete certain elements during the Iterating process, but the foreach loop cannot do that, and the Iterator is less readable than the foreach. [19]

One of the **advantages** of the Iterator is that it is easy to implement because it uses the interface and the concrete class, so the iterator is easy to implement, and it uses the single responsibility principle, this design pattern helps with cleaning up for the user, and storage through the collections of traversal algorithms into separate classes. So, the Iterator gives you clean code because the client will use an interface that is easy to use and can be reused, and the Iterator has an open/close principle that helps in creating different types of collections and iterators without errors. [20]

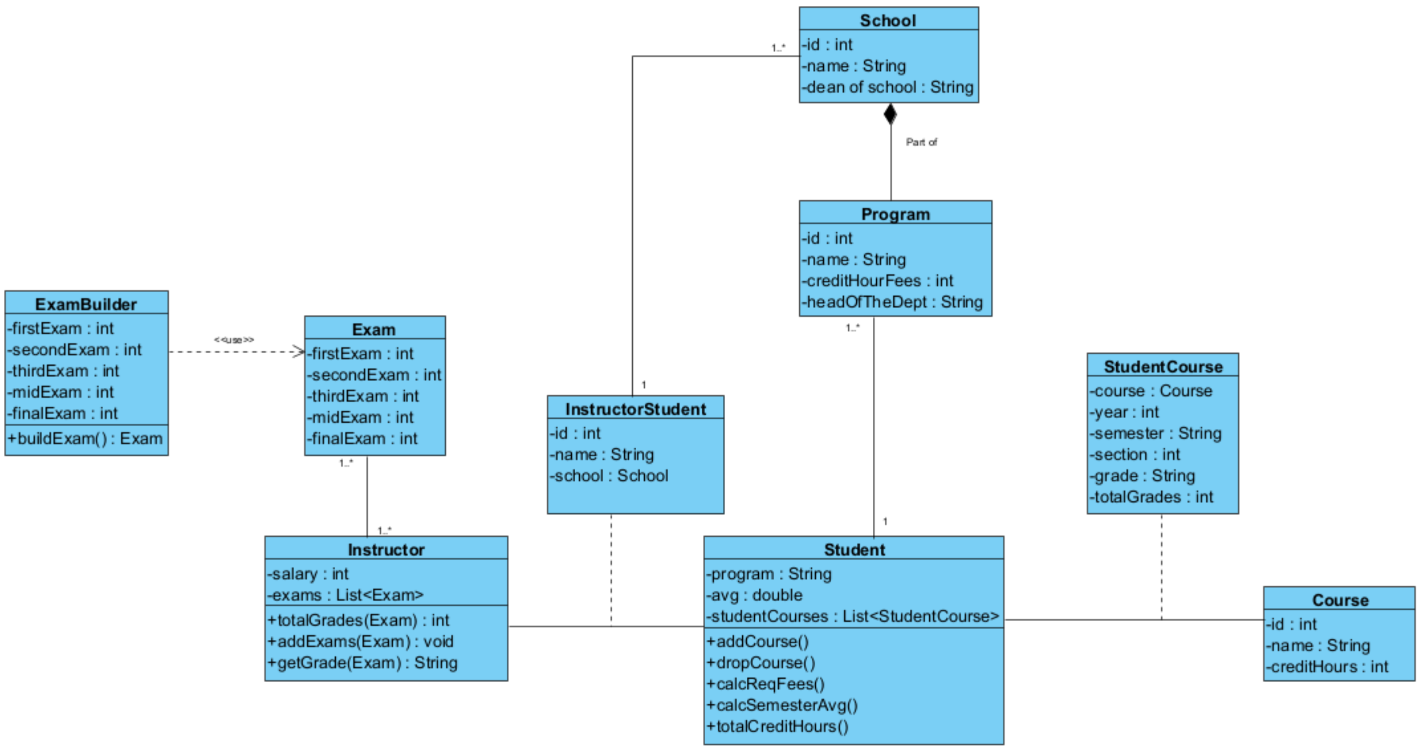
The **disadvantages** of the iterator are that it is less efficient in passing through elements and uses more memory than direct access to the element, the iterator is not preferred to use for small and simple groups and is considered overused, and the iterator is a read-only pattern so the group cannot be changed while passing it using the iterator. [20]

1.3 The relationship between OOP Paradigm and The Design Patterns:

OOP (Object Oriented Programming) is a way to program using classes and many features and characteristics. A design pattern is a pattern that is identified by looking at many solutions and comparing similarities in the details applied to solve similar problems. The design patterns are used when the OOP is used without regard to the programming language because it provides an idea, not a particular implementation. design patterns are the best practices for solving problems, and these problems are general, as it is a template that has a solution to a problem that has occurred with many programmers in OOP and is applied to any programming language. which makes the application of this code easy to implement and reduces the problems that programmers face. The design patterns are divided into three types, the Creational, which deals with the problems of the objects, trying to create objects in a way that suits the situation, the Structural, which specializes in the code structure of the classes and the objects, and the third type is the Behavioral that describes how to interact and deal with the classes and the objects with each other.

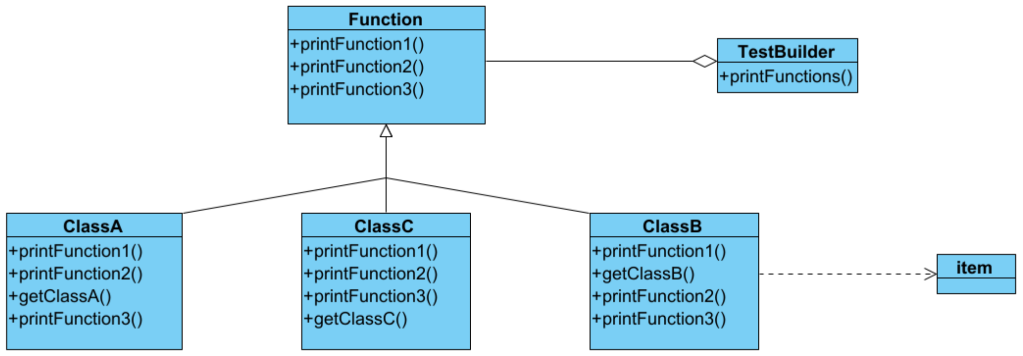
Part 2

2.1 Design and build class diagrams for the system using a UML tool

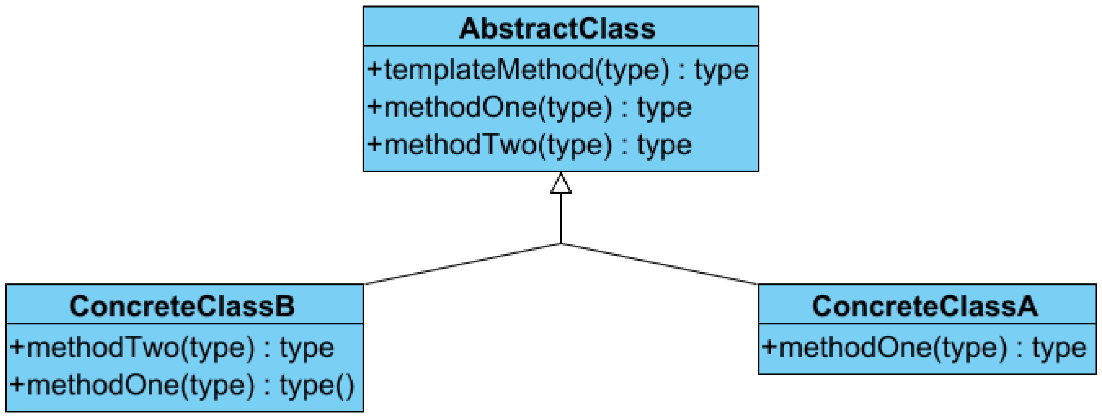


2.2 Define class diagrams for Builder and Template design patterns using a UML tool

1. Builder design pattern:



2. Template design pattern:



2.3 How Class Diagrams that drew in 2.2 can be derived from the code

The design pattern is the best application for solving problems as I mentioned earlier, and it helped me a lot in writing the code I wanted without errors. The design pattern helps in accelerating the development process by providing tried and proven development models by professionals. It looks at the deep and complex problems that may not be visible to the programmer, the design pattern helps prevent these problems. It improves the readability of the code for programmers and others, and the design pattern provides programmers with easy communication with each other because the naming convention that it follows it's known to everyone.

I realized that I will use **Builder Design Pattern** when I read the scenario and what requirements do I have to create this system, the builder is used when I have a complex object and when I want to create an object, I don't put all the values that I defined before as attributes, and this agrees with the thing required of me in the system. He wants the instructor to enter the weight of the marks in a certain way, which is that the mid-exam and the final exam are mandatory to put their mark and he can add other exams and divide the total marks by more than two exams, but in all cases, he must put the mark of the mid-exam and the final exam, and this is what conditions exist in the builder, so I used it in this part of the system.

I noticed that I have to use the **Template Design Pattern** when I read the scenario and what are the requirements that I have to do in order to build the system completely, the template is used when I want to make a system that has steps that are binding on all the subclasses, so the methods they take from the interface is abstract and its value is final, the subclasses cannot override to modify it, I just saw the need to do the same thing to create a web page and a text file to print the entire the schedule on them and no one can modify the schedule information, it will print this information for the student to see and he knows the registration information for the semester subjects he has chosen.

Part 3

3.1 Build an application derived from system UML class diagrams

In the code.

3.2 develop code that implements template, chain of responsibility, facade and singleton design patterns

In the code.

3.3 Evaluate the use of design patterns used in 3.2

As I mentioned earlier that the **Template Design Pattern** should have been used to achieve all the requirements of the system that I must apply, and it helps me a lot in applying these requirements because of the many useful uses that I mentioned earlier, but in applying the code I used the template to improve and develop it, so I must print the student registration schedule as text and as a web page containing a header that contains the title, student name, school name, program name, and current date, and the content contains a number, name, course, credit hours, total marks and final mark, and in the end, it is in the footer total of credit hours.

In the **Chain of responsibility Design Pattern** I have used to create the system, its goal is to solve the problem that the client enters or leave it to the other object that was built in the chain class if the object is built from a particular class wants to solve the problem, it is He will do the required process, I employ this idea on the scenario, it is required of me to find the approval of the grade set by the instructor, it must pass on the head of the department and the dean of the school, and all of them must agree to it until it is approved, and this is what I did. The mark will pass to the instructor, then to the head of the department, then to the dean of the school, and the approval will be printed if it is done, and that's improved my code.

I used the **Facade Design Patten** in the code to improve it because the idea of the facade is to provide a library of a group of classes in one place, it helps me reduce the effort and when I modify something inside one of the classes, the client will not be affected by this modification, the modification is dynamic, and in the code, I used to display a list to the student that when he uses the system, a list of the names of the available schools will appear for him and he can choose one of them and it will print a sentence saying that he chose the school successfully.

I used the **Singleton Design Pattern** in the code to improve it, its main idea is to force the client to create only one object to store it in one index or pointer in the database, and to do this I should make the constructor as private, and in order to create the method without creating the object I have to make the method of the static type, and this is what I used In the code, what is required of me is to make a way to make the user enter the database information only once and cannot repeat it, so I used the Singleton to make the student put the Oracle database information in one place in the memory.

Part 4

4.1 Implement design patterns using Java technology by developing a small application

The **Factory** Design Pattern is used when I want the creation of the object in the client class to be divided into more than one separate class, the client will not see those classes and will build a generic object containing all the classes, the client cares to see and handle with one class, so the example was that I have a bank that contains more than one type of account, which are saving, personal, and business account and each of them is a class that contains a certain interest that the bank has been setting it and contains a method that takes the value of the amount that the client wants to calculate the interest to it, by entering this value and choosing the symbol that represents his account type, the symbols represent the first letter of each account type and they are S, P, B and through this symbol will call the class that is concerned with calculating the interest for this type of account.

The **Bridge** Design Pattern is his idea that implementation is separated from abstraction because it is a type of structural design patterns, it is specific to the structure of the code, and it is used when I have two classes between them have things in common, they are separated according to the common things, the common things will be put them in the form of two classes and they are called one separate class which is the bridge pattern class, the example that I implemented is a representation of this idea and the most appropriate solution to implement this example was to use the bridge design pattern, the example was that I have certain geometric shapes such as a square and a triangle and these shapes have different colors but they have a common characteristics which is that they have colors, so I put the color characteristic in a separate interface, and also I put the shape characteristic in a separate abstract class, I created two classes for the green and blue and made them implement the interface and put inside them method prints me the color that the class represents, then I create two more square and triangle classes and they take the color from the interface as parameter inside the constructor, in the end I created a class named BridgePattern in order to combine the two attributes inside this class and create an object from them and print the color for the shape I want.

The **Proxy** Design Pattern is also a type of Structural Design Pattern, and its idea is that it allows me to control from a second class through the proxy class, this control is that it prevents access to certain places in the database, which is the mediator between the databases and client, it is as a layer before accessing the real object and sets a certain condition if it checks, the proxy will allow access to the real object, then the proxy and the real object implement the same interface that contains the request method, I applied this idea to an example of preventing and blocking downloading specific files about the user, I created an interface that contains a method parameter for the client to enter the file he wants to download, and I created two classes, one for the real object and another for the proxy and the two do implementation from the same interface but the proxy class contains the files that can't be downloaded by the user, and in the test class, I created an object for the user to put in it his file name, that he want to download it, and I used try and catch to print a block message in case the client wanted to download a file that prevents it from being downloaded.

The **Strategy** Design Pattern is one type of Behavioral Design Pattern, and its idea is that it collects a group of algorithms that are assembled with it, and calling any algorithm I want to activate through the test class, it creates an object of another class that links the algorithms with the value that the user will enter in the form parameter method, each algorithm has a separate class, and I applied this idea to the example of a payment card, there are two types of payment card which are credit card and PayPal, I created the interface and called it Payment and made the two classes to implement the interface, the product details like name and UPC is placed in a class named Item and also according to the information that the user must enter in each class of the two types of purchase, and through the class called shopping cart the user can add and drop any product and calculate the total of price, the number of products that were made is collected by the attribute in the method of the interface, and in the end, the total number of price and the payment method are printed which was carried out by test class.

4.2 Reconciling the most appropriate design pattern that can be used in some scenarios

- Strategy
- Singleton
- Builder
- Prototype
- Proxy
- Strategy

4.3 Critically evaluate the use of Design Patterns in 4.2

- As I mentioned previously about the importance of implementing the **Strategy Design Pattern**, the benefits of using it, and when to use it, and I mentioned this in the previous question, here I saw that what was required of me corresponds to the uses of the strategy, so I applied it, the requirement is to give the student the option to register and then pay or pay Then it is registered, I created an interface named Strategy which contains a method of type string and takes a parameter of type String, I created two classes PayThenRegister and RegisterThenPay and made them implement the interface, each class in them contains a print statement according to the function of the class, then I created a class named context that creates a variable from the interface and stores the value that the user will enter as a string, then I activate it for the student in the UserInterface.

- Here I chose the **Singleton** Design Pattern because its idea is to force the user to create only one object, as I mentioned in Question 3.2, and I applied it in the same way that I mentioned.

- Here I chose the **Builder** Design Pattern because it used when I have complex object or when I need to or also when I don't want to put all values, so I see I can apply this design pattern in this situation, I mentioned the details in put the values not in order to force the user to create only one object, as I mentioned in Question 3.2, and I applied it in the same way that I mentioned in 2.3.

- I chose the answer to be the **prototype** because its idea is to create an initial object and the next object I will build using the initial object by cloning it with a method called clone(), the prototype is there to make it easier to create the object it also has two types And two methods, shallow copy and deep clone, all of which make me create an object more quickly and save more time on the database.

- As I said earlier, the **Proxy** Design Pattern is his idea that it allows me to control another class, meaning that it has the authority to prevent the object from accessing the real object, and in this example, it will ask the user if he is a bachelor or master student and on this basis I will the proxy will block certain websites, so I chose it.

- As I mentioned earlier, the **strategy** compiles a set of algorithms for the user to deal with in one place and can easily call any of these algorithms, and it changes the behavioral during the runtime, so in this example, all the shapes will be called by the user and all of them are linked to the same interface, so I chose this design pattern.

References:

[1] Tutorialspoint.com. 2022. *Java - Encapsulation*. [online] Available at: <https://www.tutorialspoint.com/java/java_encapsulation.htm> [Accessed 14 May 2022].

[2] 2022. [online] Available at: <https://www.mygreatlearning.com/blog/polymorphism-in-java/> [Accessed 14 May 2022].

[3] www.javatpoint.com. 2022. *Java Constructor - Javatpoint*. [online] Available at: <https://www.javatpoint.com/java-constructor> [Accessed 14 May 2022].

[4] W3schools.com. 2022. *Java Abstraction*. [online] Available at: <https://www.w3schools.com/java/java_abstract.asp> [Accessed 14 May 2022].

[5] GeeksforGeeks. 2022. *Interfaces in Java - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/interfaces-in-java/> [Accessed 14 May 2022].

[6] www.javatpoint.com. 2022. *Collections in Java - javatpoint*. [online] Available at: <https://www.javatpoint.com/collections-in-java> [Accessed 15 May 2022].

[7] www.javatpoint.com. 2022. *Java Array - Javatpoint*. [online] Available at: <https://www.javatpoint.com/array-in-java> [Accessed 15 May 2022].

[8] www.javatpoint.com. 2022. *ArrayList in Java - javatpoint*. [online] Available at: <https://www.javatpoint.com/java-arraylist> [Accessed 16 May 2022].

[9] www.javatpoint.com. 2022. *Java Vector - Javatpoint*. [online] Available at: <https://www.javatpoint.com/java-vector> [Accessed 17 May 2022].

[10] www.javatpoint.com. 2022. *HashMap in Java - javatpoint*. [online] Available at: <https://www.javatpoint.com/java-hashmap> [Accessed 17 May 2022].

[11] Careerride.com. 2022. *Relation between class and object*. [online] Available at: <https://www.careerride.com/java-relation-between-class-and-object.aspx> [Accessed 20 May 2022].

[12] www.javatpoint.com. 2022. *Static keyword in Java - Javatpoint*. [online] Available at: <https://www.javatpoint.com/static-keyword-in-java> [Accessed 20 May 2022].

[13] GeeksforGeeks. 2022. *OOPS | Generalization as extension and restriction using Java - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/oops-generalization-as-extension-and-restriction-using-java/> [Accessed 20 May 2022].

[14] Javapapers. 2022. *Association, Aggregation, Composition, Abstraction, Generalization, Realization, Dependency - Javapapers*. [online] Available at: <https://javapapers.com/oops/association-aggregation-composition-abstraction-generalization-realization-dependency/> [Accessed 20 May 2022].

[15] Medium. 2022. *Overview Of Prototype Desing Pattern*. [online] Available at: <https://medium.com/geekculture/overview-of-prototype-desing-pattern-3eafaf006fde> [Accessed 1 June 2022].

[16] GeeksforGeeks. 2022. *Prototype Design Pattern - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/prototype-design-pattern/> [Accessed 1 June 2022].

[17] 2022. [online] Available at: <https://www.baeldung.com/java-adapter-pattern> [Accessed 1 June 2022].

[18] Course, T., 2022. *Adapter Design Pattern*. [online] Techcrashcourse.com. Available at: <https://www.techcrashcourse.com/2015/10/adapter-design-pattern.html> [Accessed 1 June 2022].

[19] GeeksforGeeks. 2022. Iterator Pattern - GeeksforGeeks. [online] Available at: <https://www.geeksforgeeks.org/iterator-pattern/> [Accessed 15 June 2022].

[20] Kane, A. and Kendre, N., 2022. CodeChef4U | Pros and cons using Iterator Design Pattern?. [online] Codechef4u.com. Available at: <https://www.codechef4u.com/post/2018/10/25/Pros-and-cons-using-Iterator-Design-Pattern> [Accessed 15 June 2022].

STUDENT ASSESSMENT SUBMISSION AND DECLARATION

When submitting evidence for assessment, each student must sign a declaration confirming that the work is their own.

Student name: Mahmoud Rumaneh	**Assessor name: Hamzeh Asefan**	
Issue date: 16/4/2022	**Submission date:** **20/6/2022**	**Submitted on:** **20/6/2022**
Programme: Computer Science		
Course Name: Advanced Programming **HTU Course Code : 30201200**	**BTEC UNIT:** 6	
Assignment number and title: **No. 1 Advanced Programming Assignment**		

Plagiarism

Plagiarism is a particular form of cheating. Plagiarism must be avoided at all costs and students who break the rules, however innocently, may be penalized. It is your responsibility to ensure that you understand correct referencing practices. As a university level student, you are expected to use appropriate references throughout and keep carefully detailed notes of all your sources of materials for material you have used in your work, including any material downloaded from the Internet. Please consult the relevant unit lecturer or your course tutor if you need any further advice.

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 signature: [illegible] **Date: 20/6/2022**