**COURSEWORK SUBMISSION FORM**

|  |  |  |  |
| --- | --- | --- | --- |
| **STUDENT USE** | | **STAFF USE** | |
| Module Name | Object Oriented Programming | First Marker’s  (acts as signature) |  |
| Module Code | 5COSC018C-n | Second Marker’s  (acts as signature) |  |
| Lecturer Name | Avazkhan Khalikov | Agreed Mark |  |
| UoW Student IDs |  | **For Registrar’s office use only (hard copy submission)** | |
| WIUT Student IDs | 00010023 |
| Deadline date | 12/17/2021 |
| Assignment Type | 🗌Group☑Individual |

**SUBMISSION INSTRUCTIONS**

**COURSEWORKS *must* be submitted in *both* HARD COPY (to the Registrar’s Office) *and* ELECTRONIC unless instructed otherwise.**

For hardcopy submission instructions refer to: [http://intranet.wiut.uz/Shared%20Documents/Forms/AllItems.aspx](http://charlotte.wiut.uz:36765/Shared%20Documents/Forms/AllItems.aspx) - Coursework hard copy submission instructions.doc

For online submission instructions refer to: [http://intranet.wiut.uz/Shared%20Documents/Forms/AllItems.aspx](http://charlotte.wiut.uz:36765/Shared%20Documents/Forms/AllItems.aspx) - Coursework online submission instructions.doc

|  |
| --- |
| **MARKERS FEEDBACK (Continued on the next page)** |
|  |

Table of Contents

[Introduction 3](#_Toc90655783)

[Online Library System for University 3](#_Toc90655784)

[Introduction 3](#_Toc90655785)

[Naïve Code 3](#_Toc90655786)

[Scenario 4](#_Toc90655787)

[UML Diagram 5](#_Toc90655788)

[Evaluation of academic performance system 5](#_Toc90655789)

[Introduction 5](#_Toc90655790)

[Naïve Code 6](#_Toc90655791)

[Chain of Responsibility 7](#_Toc90655792)

[UML Diagram 7](#_Toc90655793)

[Online Appointment Scheduling system for hospitals 8](#_Toc90655794)

[Introduction 8](#_Toc90655795)

[Naïve Code 8](#_Toc90655796)

[Abstract Factory 9](#_Toc90655797)

[UML Diagram 9](#_Toc90655798)

[Conclusion 10](#_Toc90655799)

# Introduction

As we are a company namely “Company” which serves for outsourcing products for company or organization needs, we have been hired for 3 products whereas we have to create products that should not be that complex, extensible and easy understandable which can be achieved by usage of various design patterns. In case of issues or problems, we are supposed to solve them and provide a further improvement for those issues.

# Online Library System for University

# Introduction

Library management system will be used by students, librarian and staff of the university. The system serves for finding required books and reserve it for loan as reservation whereas it will be taken back later. Only administrators are allowed to add, manage or delete books. Librarians has permissions to add or manage but they can’t delete books on their own for safety purposes.

# Naïve Code

It is possible to use the following approach:

*public class* Book  
{  
 *public string* Title { get; set; }  
 *public string* Author { get; set; }  
 *public string* Genre { get; set; }  
 *public string* Publisher { get; set; }  
}  
  
*public enum* Role  
{  
 Admin,  
 User  
}  
  
*public class* User  
{  
 *public string* Name { get; set; }  
 *public string* Surname { get; set; }  
 *public* Role Role { get; set; }  
}  
  
*public class* Library  
**{** List<Book> books = *new* List<Book>();  
 List<User> users = *new* List<User>();  
   
 *public void* AddBook(Book book)  
 {  
 books.Add(book);  
 }  
   
 *public void* AddUser(User user)  
 {  
 users.Add(user);  
 }  
   
 *public void* RemoveBook(Book book)  
 {  
 books.Remove(book);  
 }  
   
 *public void* RemoveUser(User user)  
 {  
 users.Remove(user);  
 }  
**}**

However, as this is a demo example of a code, the problem with this approach is that it is not following the rules whereas one class should do one task like one by one. Also, the following approach doesn’t seem to be extensible and optimized. Therefore, the design patterns come in help to resolve those issues mentioned above.

# Scenario

For this case, I created own design pattern where I constructed every single detail from zero to hero. At the beginning, I started with two main sections which are Account and Book. I created AccountRole enum for Account in order to validate Role for Account little bit later. As regards to the Book, it has all required state and methods that will be required little bit later.

The following of Role looks like this:

public enum AccountRole

{

Student,

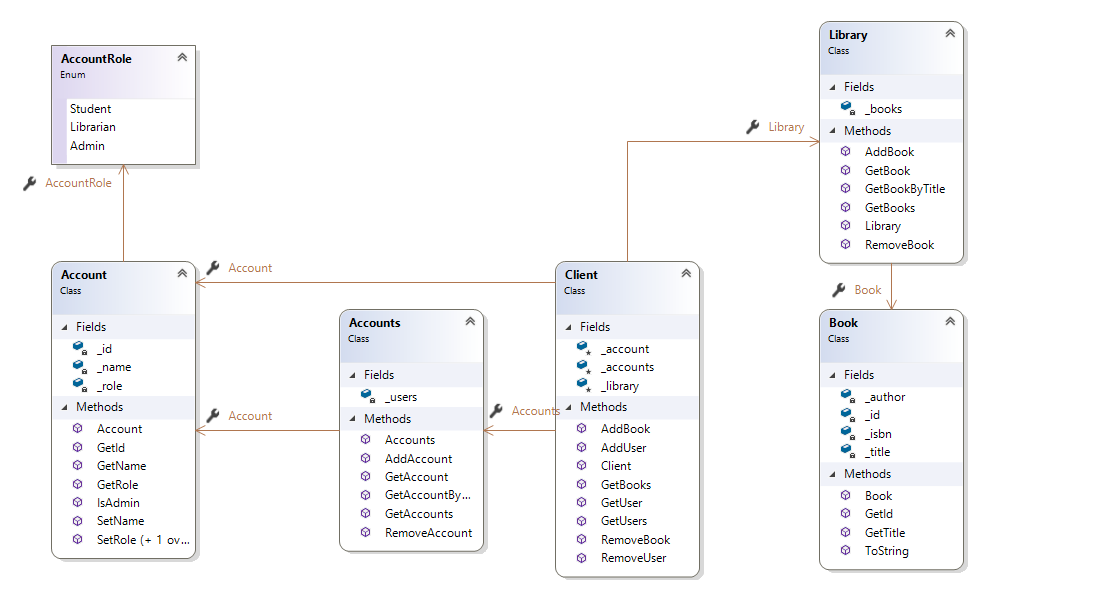
Librarian,

Admin

}

It will not be shown in the source codes of Book and Account itself, as you can visit my source code and check out how it was coded. I created yet another managers for those both classes namely Library which manages the Books and Accounts which is responsible for credential managing. Also, as we have mentioned above, we have client which connects those two Accounts and Library altogether whereas all validations for administration, librarian or student happens. The front-end part is supposed to use the Client class to interact with the backend part of the application. The main reason why I decided to go with my own pattern is that, due to lack of practice and experience, I couldn’t find the most suitable pattern for this case. However, all those design patterns were created as the GoF (Gang of Four) faced with some sort of problems and as a solution for these issues, those design patterns were created, Personally, myself, I decided to try go from scratch so I can understand those patterns by trying to find solution for those cases by myself. I might not be genius but practice matters.

# UML Diagram



# Evaluation of academic performance system

# Introduction

In this case, we have to create a platform where teachers will be able to keep track of their performance, update and add some goals or milestones that will be continuously reviewed by supervisors (administrators) or human resources. Teachers will have some sort of score that can be modified only by administrators or human resources. Also, human resources can add reports about teachers regarding to the teachers outperforms.

# Naïve Code

The solution might be something like this:

*public class* Teacher  
{  
 *public string* Name { get; set; }  
 *public int* Score { get; set; }  
 *public* List<*string*> Reports { get; set; }  
 *public* List<*string*> Goals { get; set; }  
   
 *public* Teacher(*string* name)  
 {  
 Name = name;  
 Score = 0;  
 Reports = *new* List<*string*>();  
 Goals = *new* List<*string*>();  
 }  
   
 *public void* AddReport(*string* report)  
 {  
 Reports.Add(report);  
 }  
   
 *public void* AddGoal(*string* goal)  
 {  
 Goals.Add(goal);  
 }  
   
 *public void* AddScore(*int* score)  
 {  
 Score += score;  
 }  
}  
  
*public enum* Personal  
{  
 HumanResource,  
 Administration  
}  
  
*public class* Administrator  
**{** *public int* Id { get; set; }  
 *public* Personal Roles { get; set; }  
   
 *public* Administrator(*int* id, Personal roles)  
 {  
 Id = id;  
 Roles = roles;  
 }  
   
**}**

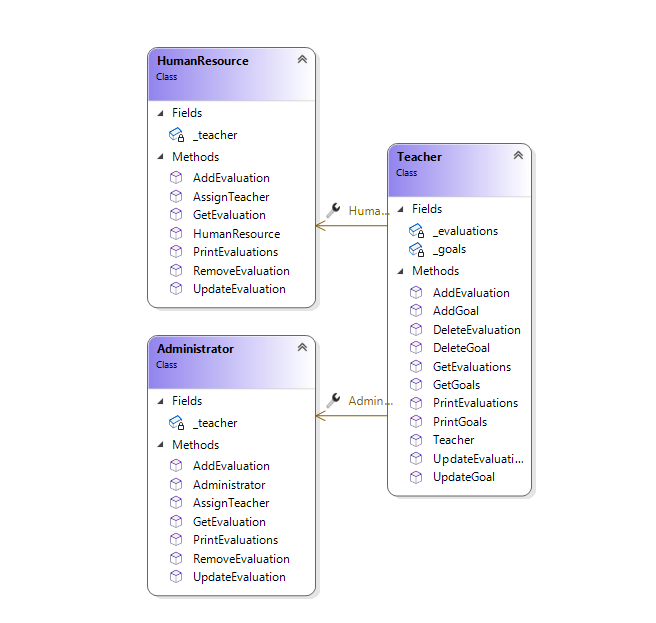
This is a demo code which doesn’t use any optimization, extensibility or refactoring. A pure code that works without design patterns. However, this code doesn’t deserve any improvements as the any kind of fix or improvement may require reproducing the whole project which consumes yet another big amount of time. However, by using design patterns, we might make our code something understandable and a code that can be contributed by anyone.

# Chain of Responsibility

The main intention of this design pattern is that avoiding coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.

In my case, I used it to pass the teacher things to Human Resource and Administration as it can be seen from the UML diagram below.

# UML Diagram



# Online Appointment Scheduling system for hospitals

# Introduction

The system is about appointing meetings between the doctors and patients. There should be 3 types of users that are client who can book free time of doctor, doctor who can manage their own data category and free time, administrator who is able to perform all basic crud operations.

# Naïve Code

Developer might use this naïve approach:

*public class* Patient  
{  
 *public string* Name { get; set; }  
 *public string* Address { get; set; }  
   
 *public* Patient(*string* name, *string* address)  
 {  
 Name = name;  
 Address = address;  
 }  
   
 *public override string* ToString()  
 {  
 *return $*"{Name} - {Address}";  
 }  
}  
  
*public class* Event  
{  
 *public string* Name { get; set; }  
 *public string* Date { get; set; }  
   
 *public* Patient Patient { get; set; }  
   
 *public* Event(*string* name, *string* date, Patient patient)  
 {  
 Name = name;  
 Date = date;  
 Patient = patient;  
 }  
  
 *public override string* ToString()  
 {  
 *return $*"{Name} - {Date} - {Patient.Name}";  
 }  
}  
  
*public class* Doctor  
**{** *private string* \_name;  
 *private string* \_specialty;  
 *private readonly* List<Event> \_events;  
   
 Doctor(*string* name, *string* specialty)  
 {  
 \_name = name;  
 \_specialty = specialty;  
 \_events = *new* List<Event>();  
 }  
 *public void* AddEvent(Event e)  
 {  
 \_events.Add(e);  
 }  
   
 *public void* PrintEvents()  
 {  
 *foreach* (Event e *in* \_events)  
 {  
 Console.*WriteLine*(e.ToString());  
 }  
 }  
**}**

However, the problem with the following solution is that after using this kind of approach, we can’t add something or we need to update the whole classes one more time if we want to make some changes to some methods or something else which means, our application is not extensible at all.

# Abstract Factory

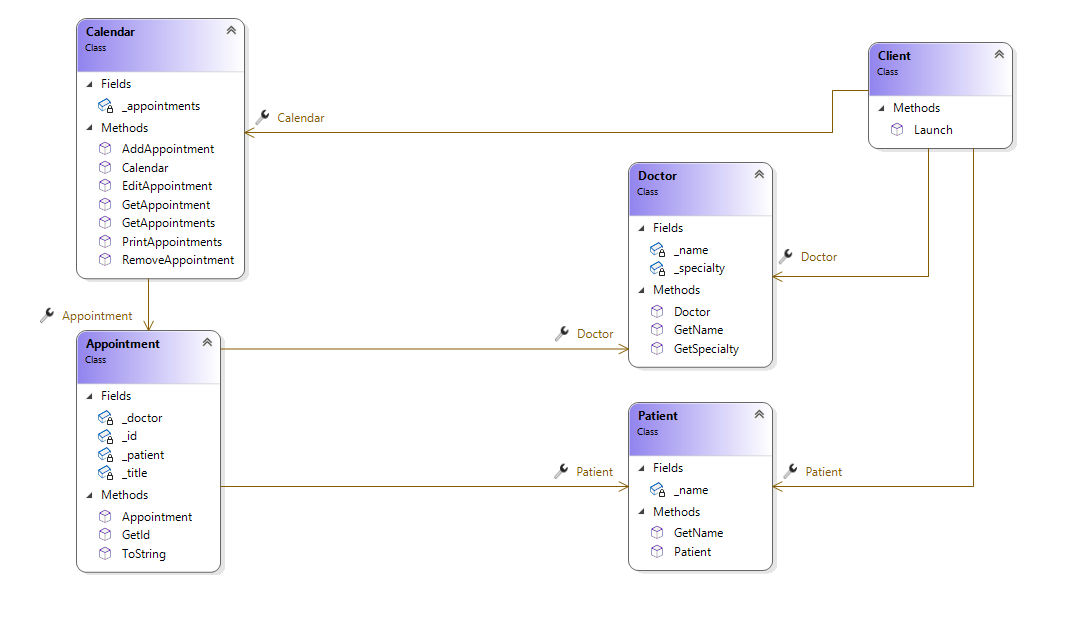
Provide an interface for creating families of related or dependent objects without specifying their concrete classes.

The reason why I used this is that abstract factory is applicable at four cases:

* a system should be independent of how its products are created, composed, and represented.
* a system should be configured with one of multiple families of products
* a family of related product objects is designed to be used together, and you need to enforce this constraint
* a family of related product objects is designed to be used together, and you need to enforce this constraint

and in our case, it really suits well. It helps us to isolate concrete classes.

# UML Diagram



# Conclusion

It might be hard to determine whether what design patterns to go with at the first side, but as more you will you design patterns while constructing schemes for software, more you get use to it. As it was stated during the lectures, design patterns that was created by the GoF are mostly used by experts to make out the whole potential of software. The lesson that I’ve learned is that when creating a software for open source or for some organization, design patterns will serve as a way of communication between groups of developers as it might not be understandable to update functions or individual classes due to lack of design of the whole software.