МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

„КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**КУРСОВА РОБОТА**

з дисципліни «Об’єктно-орієнтоване програмування»

на тему

**Шаблони проектування в ООП. Програмне забезпечення мовної школи.**

Виконала студентка

ІІ курсу групи КП-82

Лахман Ксенія Вікторівна

залікова книжка КП-8214

Керівник роботи

доцент, к.т.н. Заболотня Т.М.

Оцінка

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата, підпис)

КИЇВ 2020

**ЗМІСТ**

ВСТУП 3

1. СТРУКТУРНО-АЛГОРИТМІЧНА ОРГАНІЗАЦІЯ ПРОГРАМНОЇ СИСТЕМИ ОБЛІКУ РОБОЧОГО ЧАСУ 5

1.1. Модульна організація програми 6

1.2. Функціональні характеристики 7

1.3. Опис реалізованих класів 7

2. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ЗА ДОПОМОГОЮ ШАБЛОНІВ ПРОЕКТУВАННЯ 19

2.1. Обґрунтування вибору та опис шаблонів проектування для   
 програмної реалізації мовної школи 19

2.2. Діаграма класів 23

2.3. Опис результатів роботи програми 25

ВИСНОВКИ 32

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 33

ВСТУП

Дана курсова робота присвячена розробці програмного забезпечення для управління роботи мовної школи за допомогою використання шаблонів проектування.

Нерідко процес запису клієнтом на мовний курс, що його цікавить, є доволі повільним. Клієнту спочатку необхідно подзвонити до представника школи, дізнатись список наявних курсів, деталі та вартість того курсу, що його цікавить. Подумати, чи дійсно він хоче придбати даний курс, у разі позитивного рішення – дізнатись деталі оплати потрібного курсу, часто прийти до фізичного об’єкту школи та розрахуватись. За допомогою даного програмного рішення вищеописаний процес можна суттєво прискорити. Звичайні відвідувачі можуть одразу переглянути усі наявні курси, додавати до своєї корзини та розрахуватись, не виходячи із програми.

Дана тематика обрана для виконання курсової роботи тому, що результати абстрагування об’єктів у цій предметній галузі дозволяють застосувати всі вивчені принципи та методи об’єктно-орієнтованого програмування для створення програмного забезпечення (ПЗ), зокрема шаблони проектування.

*Об’єктом* дослідження є процес купівлі та обліку курсів.

*Метою роботи* є розроблення програмного забезпечення управління мовною школою з використанням шаблонів проектування.

Для досягнення визначеної мети необхідно виконати такі *завдання*:

* абстрагувати об’єкти предметної галузі;
* розробити структурну організацію ПЗ за допомогою застосування основних принципів ООП та шаблонів проектування;
* визначити та описати функціональні характеристики програми;
* обґрунтувати вибір шаблонів проектування, використаних для побудови програми;
* розробити дизайн інтерфейсу користувача;
  + виконати реалізацію програмного забезпечення відповідно до вимог технічного завдання;
  + виконати тестування розробленої програми;
  + оформити документацію з курсової роботи.

Розроблене ПЗ мовної школи складається з наступних модулів: модуль для інтерфейсу користувача; модуль особистого кабінету користувача, модуль, що містить інформацію про усі доступні курси, модуль корзини користувача; модуль для керування базою даних курсів адміністратором.

Реалізовані шаблони проектування: **Command, Builder, Prototype, State, Visitor, Strategy, Proxy,** **Template Method.** Додатково було використано також **Singleton.**

До функціональних можливостей програми належать: перевірка коректності вхідних даних, додавання та видалення курсів, контроль доступу користувачів, автентифікація даних користувачів для входу у систему.

Для функціонування розробленої програми необхідно забезпечити наявність на комп’ютері 30 Мб вільного дискового простору.

Даний додаток може бути використаний мовною школою, власники якої хочуть забезпечити зручну взаємодію з клієнтами.

Пояснювальна записка складається зі вступу, двох розділів, загальних висновків, списку використаних джерел (5-ти найменувань). Робота містить 18 рисунків. Загальний обсяг роботи – 33 друкованих сторінок.

1. **ОПИС СТРУКТУРНО-ЛОГІЧНОЇ СХЕМИ ПРОГРАМИ**
   1. Модульна організація програми

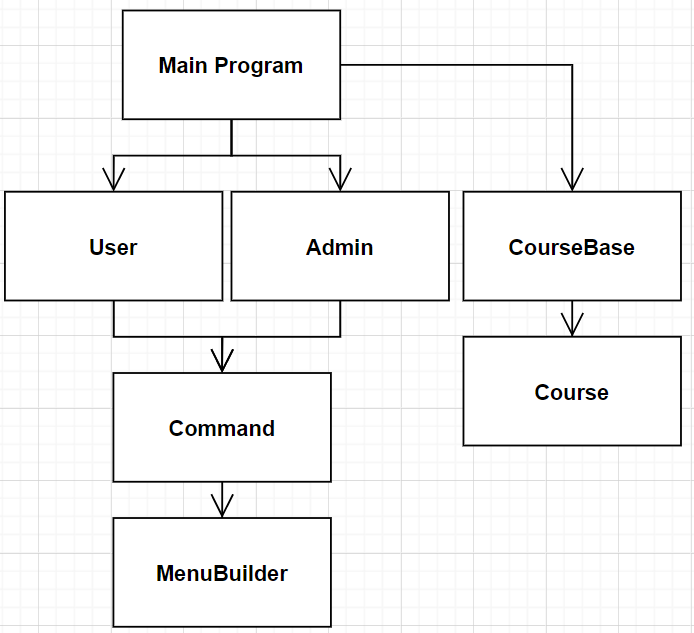


Рис. 1.1.1 «Модулі програми»

У програмі можна виділити декілька основних логічних модулів, що мають певну самостійність і обмінюються один з одним даними.

**Main Program** – головний модуль програми. Через нього відбувається взаємодія зі всіма частинами програми.

**User –** модуль для роботи користувача. Відповідає за взаємодію користувача із системою, реалізацію усіх можливостей користувача.

**Admin –** модуль для роботи адміністратора. Відповідає за взаємодію адміністратора із системою, реалізацію усіх прав адміністратора.

**Course** – модуль, який відповідає за головний об’єкт системи – курс. Може бути створений лише адміністратором. Є об’єктом наповнення бази даних.

**Course Base** – набір класів та методів, які відповідають за взаємодію із базою даних курсів, виконуючи основні функції, потрібні для відображення, створення та видалення сутностей програми.

**Command** – модуль, який надає можливість користувачу/адміністратору обрати, яке меню він хоче бачити. Команда надсилає інформацію модулю **Menu Builder.**

**Menu Builder** – модуль, який відповідає за побудову потрібного меню відповідно до обраного параметру.

* 1. **Функціональні характеристики**

Робота з програмою розпочинається із *Main Menu* (головного меню). Користувачу пропонується обрати відповідну роль і пройти автентифікацію своєї особистості. Після входу в систему як користувач, пропонується перейти до одного із наступних меню: *Personal Area, Cart, Courses.*

Меню *Personal Area* – особистий кабінет користувача. У ньому можна побачити свої персональні дані, які є у системі, та список придбаних курсів.

Меню *Cart –* корзина. Користувач може переглянути курси, додані до корзини, а також відредагувати їх зміст, скориставшись наступними командами:

*Buy Courses –* купити всі наявні у корзині одиниці. При цьому із фінансового рахунку користувача зніметься сумарна вартість даних курсів.

*Remove Course* дозволяє видалити конкретну одиницю із корзину.

*Clear Cart –* повністю очистити корзину.

Меню *Courses –* меню перегляду курсів у двох режимах: стандартному, що одразу виводиться на екран, та вивід курсів за рівнем, який обирає користувач. З цього меню відбувається додавання бажаного курсу до кошику. Таку функціональність забезпечують команди *level* та *add.*

Після входу як адміністратор, пропонується перейти у меню *Courses.* Наступні методи дозволяють взаємодію із базою даних:

*add –* створити новий курс

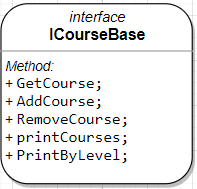
*remove* – видалити існуючий курс

*clone* – створити новий курс як копію уже існуючого

У будь-який момент роботи програми користувач має можливість повернутись у попереднє меню – *Back,* повернутись до меню вибору ролі обравши команду *Exit,* або вийти з програми – знову натиснувши *Exit.*

* 1. **Опис реалізованих класів**

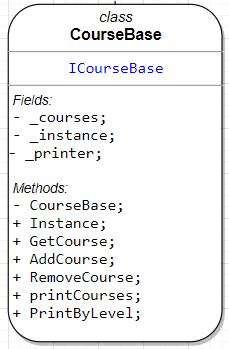
1. **Інтерфейс ICourseBase**

**Завдання:** Надати набір методів для роботи з базою даних.

**Методи:**

* *AddCourse* – додавання нової сутності
* *RemoveCourse –* видалення існуючої сутності
* *CloneCourse –* клонування існуючої сутності
* *GetCourse –* читання існуючої сутності
* *PrintByLevel –* вивід наявних курсів за потрібним рівнем.
* *PrintCourses –* вивід наявних курсів

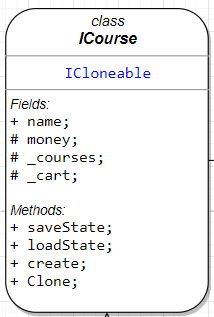
1. **CourseBase**

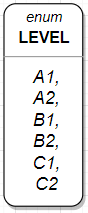
****

**Завдання:** реалізувати методи інтерфейсу ICourseBase.

**Методи:**

* *AddCourse*, *RemoveCourse, CloneCourse, GetCourse, PrintByLevel, PrintCourses -* реалізації методів, заданих інтерфейсом ICourseBase
* *Instance –* статичний метод, який за допомогою шаблона Singleton забезпечує створення лише одного екземпляра класу.
* *CourseBase –* конструктор.

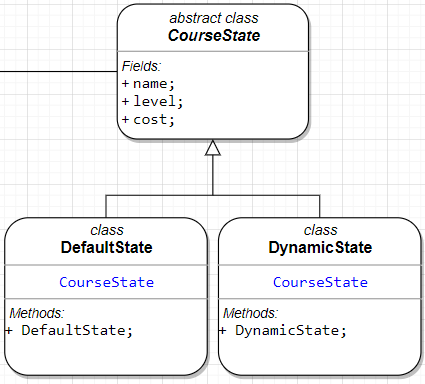
1. **Інтерфейс ICourse**

** Завдання:** Надати набір методів для роботи з головним об’єктом. Надає набір для роботи із шаблоном State. Наслідується від інтерфейсу ICloneable для реалізації шаблону Prototype.

**Методи:**

* *saveState* – створює та зберігає новий динамічний стан об’єкта.
* *loadState –* зберігання даних стану, отриманих із параметрів.
* *create –* створення сутності
* *Clone –* створення копії сутності.

1. **Система класів CourseState, DefaultState, DynamicState**

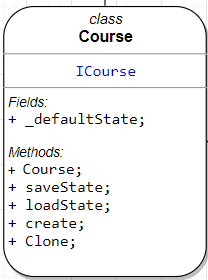
**Завдання:** надати можливості реалізації шаблону Prototype.

**CourseState :** реалізація шаблону State.

**DefaultState :** створення стану сутності за константними даними.

**DynamicState :** створення стану сутності за заданими даними.

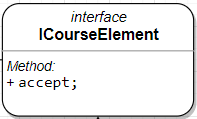
1. **Course**

**Завдання:** реалізація методів інтерфейсу ICourse. Виступає конкретним прототипом у шаблоні Prototype.

**Методи:**

* *Course –* конструктор, який створює стандартну сутність через \_defaultState.
* *saveState*, *loadState, create,* *Clone –* реалізація методів, заданих інтерфейсом ICourse.

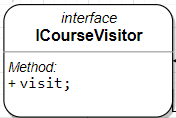
1. **Інтерфейс ICourseElement**

**Завдання:** Описати метод прийому відвідувача. Виступає Елементом у шаблоні Visitor

**Методи:**

* *accept –* приймає об’єкт відвідувача.

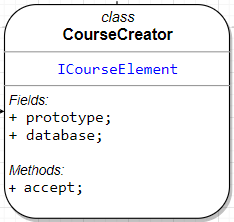
1. **Інтерфейс ICourseVisitor**

**Завдання:** надати методи реалізації для ролі Відідувач у шаблоні Visitor.

**Методи:**

* *visit - «*відвідує» обраний елемент.

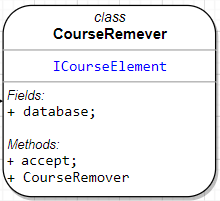
1. **CourseCreator**

**Завдання:** реалізація методів інтерфейсу ICourseElement. Після успішного «прийняття» відвідувача – створити новий елемент.

**Методи:**

* *accept –* реалізація методу інтерфейсу ICourseElement.

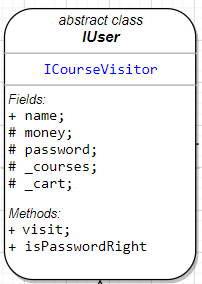
1. **CourseRemover**

**Завдання:** реалізація методів інтерфейсу ICourseElement. Після успішного «прийняття» відвідувача – видалити елемент.

**Методи:**

* *accept –* реалізація методу інтерфейсу ICourseElement.
* *CourseRemover –* конструктор, в який передається ключ елемента, який необхідно видалити

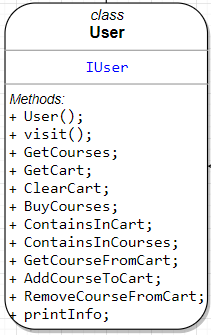
1. **Інтерфейс IUser**

**Завдання:** надати набір методів для конкретних відвідувачів у шаблоні Visitor

**Методи:**

* *visit –* реалізує інтерфейс ICourseVisitor.
* *IsPasswordRight –* перевіряє правильність паролю для шаблона Proxy

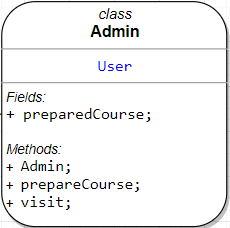
1. **User**

**Завдання:** описати поведінку користувача у системі. Клас є конкретним відвідувачем конкретним відвідувачем у шаблоні Visitor.

**Методи:**

* *visit –* при відвідуванні цього класу користувач отримує сповіщення, що йому не дозволено створювати курс, адже він не має необхідних прав доступу.
* *GetCourses, GetCar –* повертають зміст придбані користувачем курси та зміст його кошика.
* *ContainsInCourses, ContainsInCart –* відповідає на питання «Чи купив уже користувач цей курс? », «Чи вже додано цей курс до кошика? »
* *User –* конструктор, який створює користувача в системі. Має декілька перевантажень, реалізації яких залежать від кількості введеної інформації.
* *ClearCart, BuyCourses, RemoveCourseFromCart, AddCourseToCart –* забезпечують взаємодії з кошиком користувача. Реалізують можливості додавання, видалення та купівлю курсів із кошика, повного його очищення.

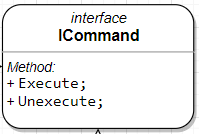
1. **Admin**

**Завдання:** описати поведінку адміністратора у системі. Клас є конкретним відвідувачем у шаблоні Visitor.

**Методи:**

* *Admin –* конструктор, який створює адміністратора у системі.
* *prepareCourse –* отримує як параметр конкретні дані про курс, який необхідно створити після відвідування методом visit
* *visit –* реалізація інтерфейсу ICourseVisitor. При відвідуванні цього класу в залежності від конкретних елементів, створюється підготовлений раніше курс, використовуючи CourseCreator, і додається у базу даних CourseBase, або видаляється існуючий курс завдяки CourseRemover.

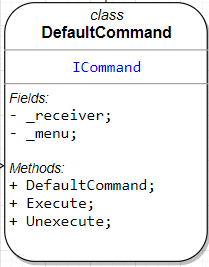
1. **Інтерфейс ICommand**

**Завдання:** описати інтерфейс, спільний для всіх конкретних команд.

**Методи:**

* *Execute –* запуск команди.
* *Unexecute –* відміна дії команди, повернення до попереднього стану.

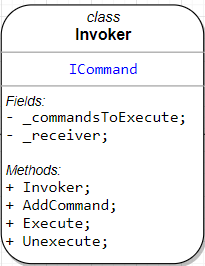
1. **DefaultCommand**

**Завдання:** реалізувати методи інтерфейсу ICommand. Виступає конкретною командою у шаблоні Command. Команда не робить всю роботу самостійно, а лише передає виклик до IMenu.

**Методи:**

* *DefaultCommand –* зберігання даних стану, отриманих із параметрів.
* *Execute та Unexecute –* реалізація інтерфейсу ICommand. Додає або видаляє меню отримувача, яким виступає IMenu.

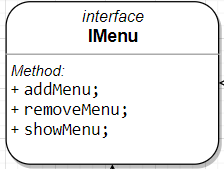
1. **Invoker**

****Виступає Відправником у шаблоні Command. Зберігає посилання на об’єкт команди та звертається до нього, коли потрібно виконати якусь дію. Відправник працює з командами тільки через їхній загальний інтерфейс. Він не знає, яку конкретно команду використовує, оскільки отримує готовий об’єкт команди від клієнта.

**Методи:**

* *Invoker –* конструктор, створення відправника.
* *AddCommand –* додавання команди до списку команд \_commandsToExecute.
* *Execute та* *Unxecute* *–* реалізація інтерфейсу ICommand.

1. **Інтерфейс IMenu**

** Завдання:** оголосити інтерфейс методів для побудови меню. Виступає Отримувачем у шаблоні Command.

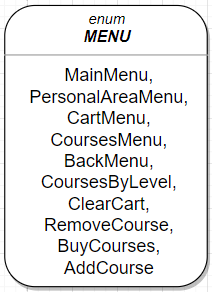
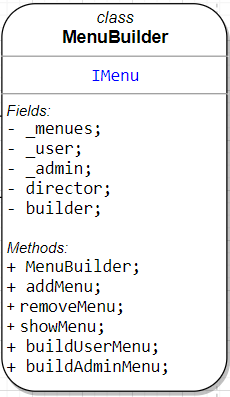
**Методи:**

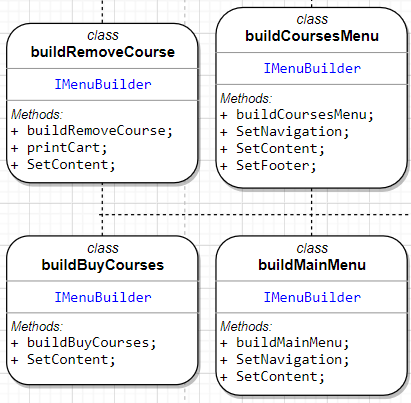
* *addMenu –* додає меню до стеку тих, які треба побудувати.
* *removeMenu –* видаляє вершину стеку.
* *showMenu –* показує меню.

1. **MenuBuilder**

**Завдання:** реалізувати методи інтерфейсу IMenu

**Методи:**

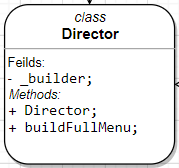
* *****AddMenu, removeMenu, showMenu –* реалізація методів інтерфейсу
* *MenuBuilder –*констуктор
* *buildUserMenu*, *buildAdminMenu–*приймає параметр MENU і на його основі будує відповідне меню за кроками, які задає director.

1. **Класи, які наслідуються від IMenuBuilder:**

**buildAddCourseMenu, buildCartMenu, buildPersonalAreaMenu, buildMainMenu, buildCoursesByLevel, buildCoursesMenu, buildClearCart, buildRemoveCourse, buildBuyCourses, buildAddCourseToDatabase, buildCloneCourseMenu, buildRemoveCourseFromDatabase**

Конкретні будівники у шаблоні Builder. Реалізують кроки будівництва, кожен по-своєму.

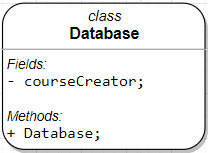
1. **Director**

** Завдання:** задати порядок кроків, які буде виконувати будівельник. Є директором у шаблоні Builder

**Методи:**

* *Director* – конструктор
* *BuildFullMenu –* метод, який задає послідовність кроків

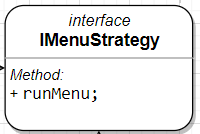
1. **Database**

**Завдання:** наповнити базу даними.

**Методи:**

* *Database –* адміністратор створює курси.

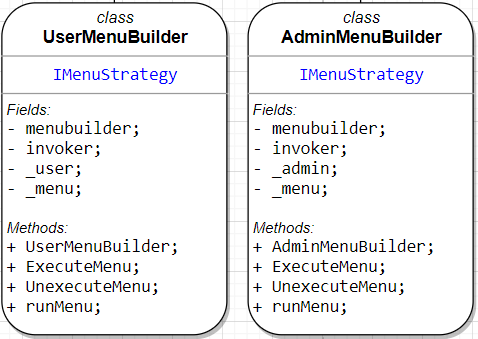
1. **IMenuStrategy**

**Завдання:** визначити інтерфейс, спільний для всіх варіацій алгоритму побудови меню. Є інтерфейсом сервісу для шаблона Proxy та інтерфейсом для шаблона Strategy.

**Методи:**

* *runMenu –* відтворити меню.

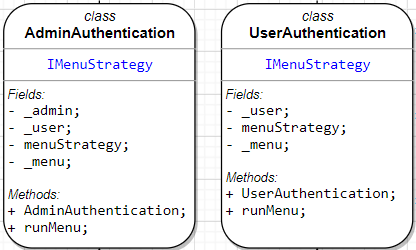
1. **UserMenuBuilder та AdminMenuBuilder**

**Завдання:** реалізувати методи інтерфейсу IMenuStrategy. Створити користувацький інтерфейс за двома стратегіями.

**Методи:**

* *UserMenuBuilder та AdminMenuBuilder –* конструктори, отримує дані про користувача, для якого будується меню.
* *ExecuteMenu –* описує шаблонні кроки для побудови будь-якого елементу меню: створити стандартну команду на основі обраного користувачем меню, додати її до відправника, і відтворити.
* *UnexecuteMenu –* кроки для реалізації дії, протилежної до *ExecuteMenu.*

1. **AdminAuthentication та UserAuthentication**

** Завдання:** реалізувати методи інтерфейсу IMenuStrategy. Забезпечити автентифікацію даних користувача/адміністратора для доступу до UserMenuBuilder та AdminMenuBuilder.

**Методи:**

* *AdminAuthentication та UserAuthentication –* конструктори, отримує дані про користувача.
* *runMenu* – реалізація інтерфейса *IMenuStrategy.*

1. **ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ЗА ДОПОМОГОЮ ШАБЛОНІВ ПРОЕКТУВАННЯ**
   1. **Обґрунтування вибору та опис шаблонів проектування для реалізації програмного забезпечення автомату**
2. **Command**

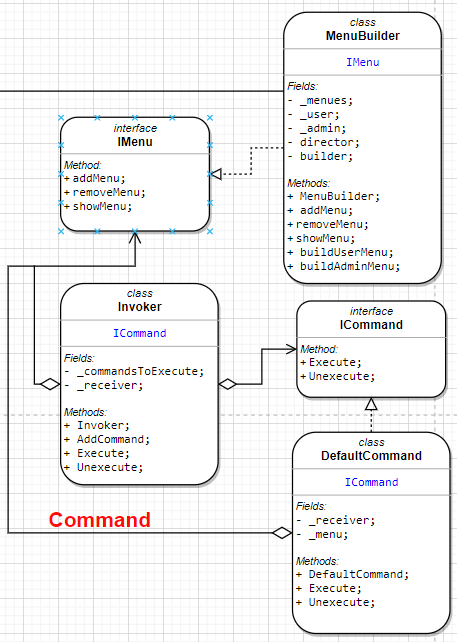
****

Рис. 2.1.1 «UML-діаграма шаблону «Command» »

Команда — це поведінковий патерн проектування, який дозволяє ставити запити в чергу, логувати їх, а також підтримувати скасування операцій. **Обґрунтування використання даного шаблону:**

Command:

* потрібна операція скасування, тобто повернення до минулого меню
* прагнення розділення інтерфейсу та логіки.

1. **Builder + Template Method**

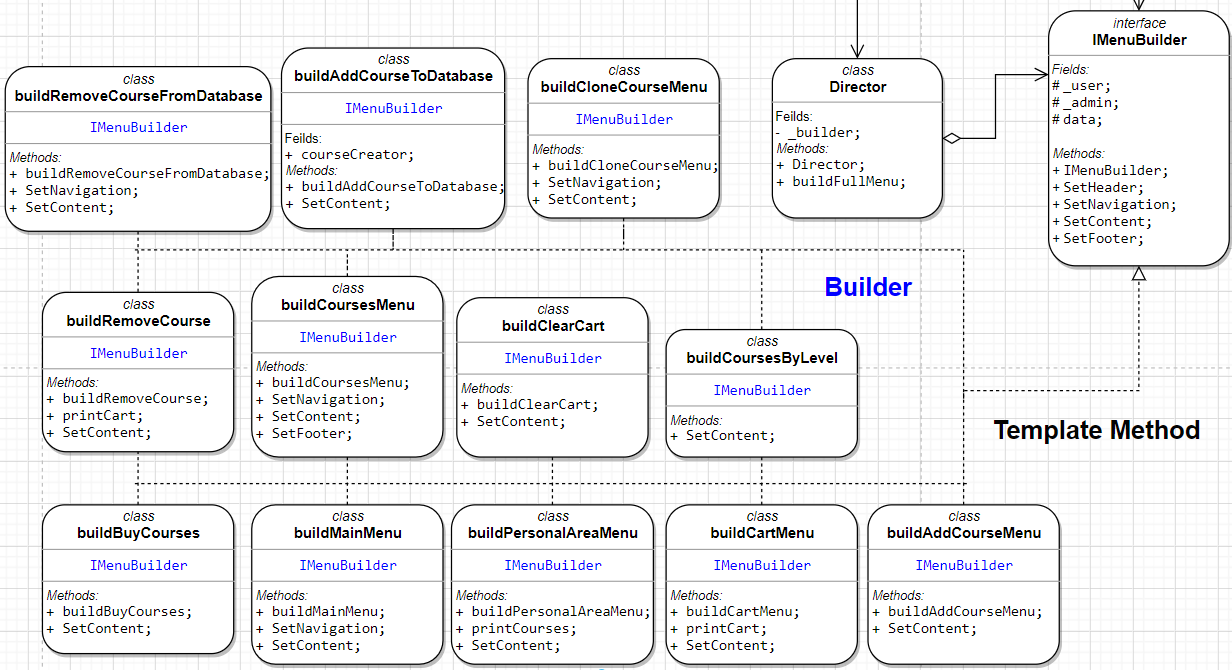
****

Рис. 2.1.2 «UML-діаграма шаблону «Command» та «Template Method»»

Будівник— це породжувальний шаблон проектування, що дає змогу створювати складні об’єкти крок за кроком.

Шаблонний метод визначає основу алгоритму та дозволяє підкласам перевизначити деякі кроки алгоритму, не змінюючи структуру в цілому.

**Структура:**

Побудова меню складається з наступних кроків: *SetHeader, SetContent, SetFooter. Director* визначає чітку послідовність виконання кроків. Кожен конкретний будівник поступово будує меню так, як йому потрібно. *SetFooter* у більшості будівників однаковий, тому його винесено як шаблон. Перевизначається лише у одному класі.

**Обґрунтування використання даного шаблону:**

Builder:

* код повинен створювати різні уявлення одного об’єкта
* алгоритм створення складного об’єкта не повинен залежати від того, з яких частин складається об’єкт і яким чином вони між собою з’єднані;
* процес конструювання повинен забезпечувати створення різних видів подання об’єкта, що конструюється.

Template Method:

* треба відокремити та локалізувати в одному класі поведінку, що є загальною для усіх підкласів, щоб запобігти дублювання коду.

1. **Visitor**

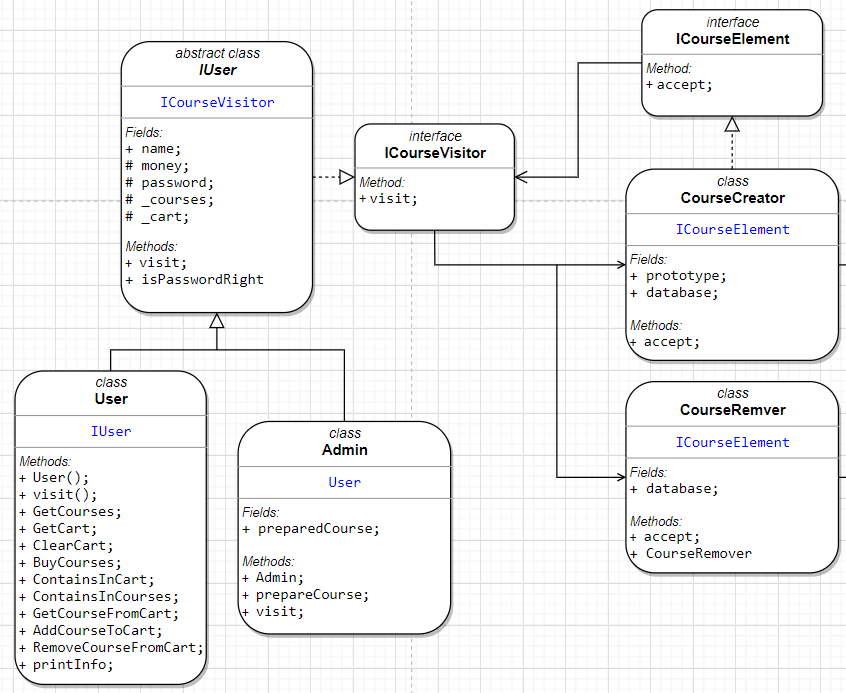
****

Рис. 2.1.3 «UML-діаграма шаблона «Visitor»»

**Структура:**

ICourseElement описує метод прийому відвідувача. ICourseVisitor описує спільний для всіх типів відвідувачів інтерфейс. CourseCreator і CourseRemover – конкретні елементи. CourseCreator дозволяє створювати новий елемент, CourseRemover – видаляти курс із бази даних. IUser описує інтерфейс для конкретних відвідувачів – класів User та Admin. Вони реалізують особливу поведінку для всіх типів елементів, які можна подати через методи інтерфейсу відвідувача. Відвідувач забороняє класу User створювати новий об’єкт курсу, а класу Admin – дозволяє і створює, додаючи новий об'єкт у базу даних.

**Обґрунтування використання даного шаблону:**

* + потрібно виконати операцію над усіма елементами складної структури і забезпечити доступ лише для одного класу
  + така поведінка має сенс тільки для деяких класів з існуючої ієрархії.

1. **Prototype + State**

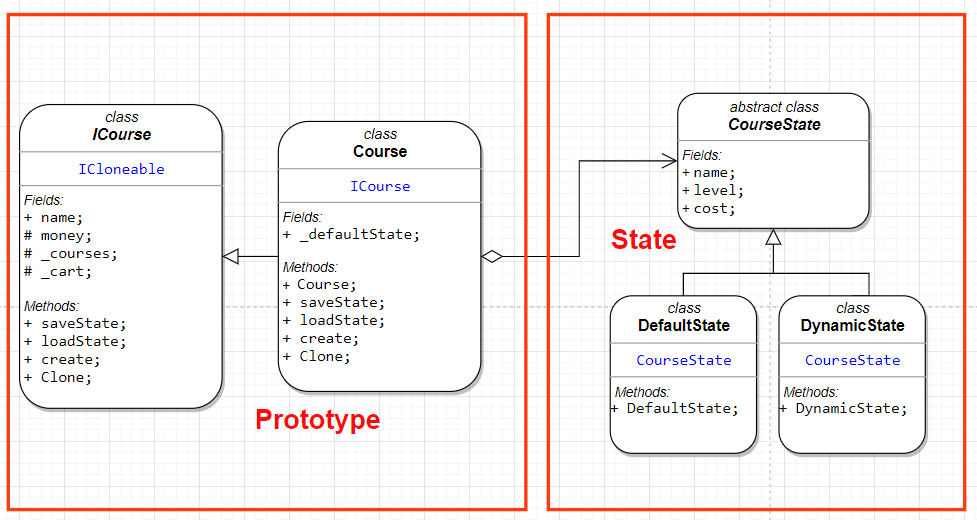
****

Рис. 2.1.4 «UML-діаграма шаблонів «Prototype» + «State» »

Прототип — це породжувальний шаблон проектування, що дає змогу копіювати об’єкти, не вдаючись у подробиці їхньої реалізації.

**Структура:**

Абстрактний клас ICourse оголошує методи для Course. CourseState має два стани: DefaultState – аби створювати стандарний об’єкт, DynamicState – аби клонувати їх, попередньо змінивши. Course має поле \_defaultState та методи для управління станом.

**Обґрунтування використання даного шаблону:**

Prototype та State використано разом, для того, щоб забезпечити клонування об’єктів, але мати змогу змінити деякі його властивості. Передбачається можливість, що у майбутньому об’єктів буде ставати більше, і простіше їх клонувати, змінюючи властивості, якщо потрібно. Також це зроблено для того, аби мати можливість «запускати» новий курс, аналогічний старому, якщо набір на нього вже закінчено.

1. **Proxy**

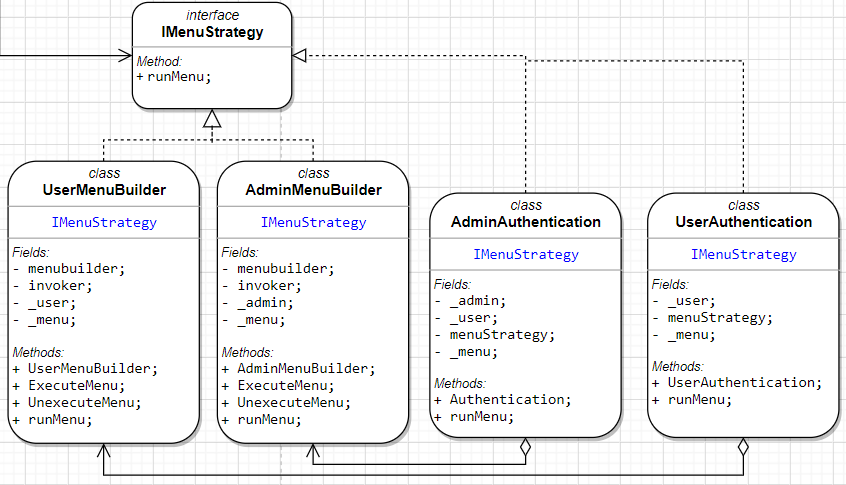
****

Рис. 2.1.5 «UML-діаграма шаблонів «Proxy» »

Заступник для автентифікації (заступник-захисник): перевіряє правильність умов доступу до об’єкту;

**Структура:**

Інтерфейс сервісу визначає загальний інтерфейс для сервісу й замісника. UserMenuBuilder та AdminMenuBuilder містять логіку побудови меню. Замісник зберігає посилання на об’єкт сервісу. Після того, як замісник закінчує свою роботу (автентифікацію), він передає виклики вкладеному UserMenuBuilder та AdminMenuBuilder.

**Обґрунтування використання даного шаблону:**

* + потрібно контролювати доступ до об’єкту: для різних об’єктів можна встановлювати різні рівні доступу;

1. **Strategy**

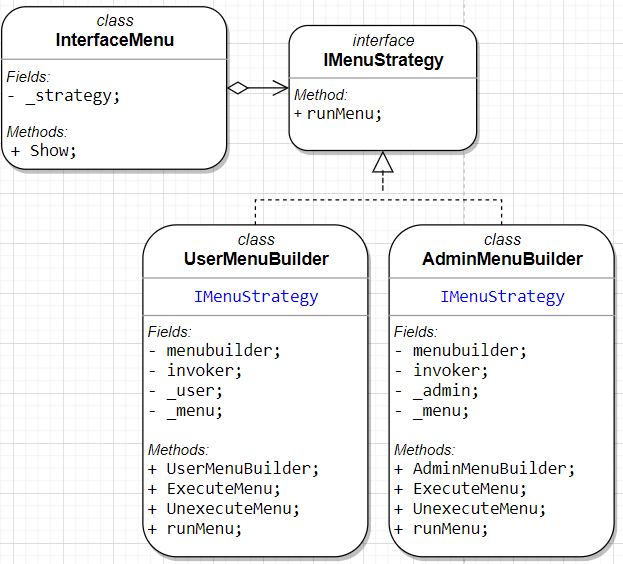
****

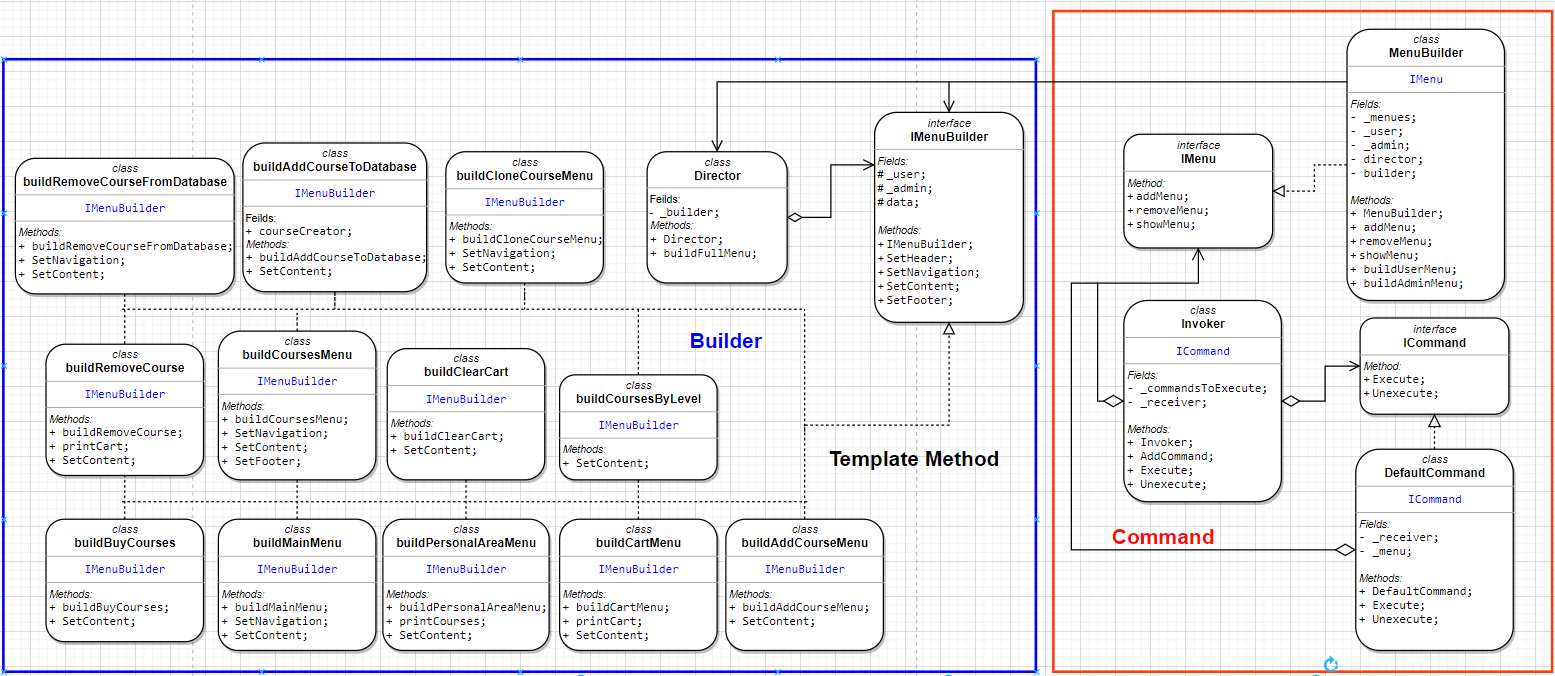
Рис. 2.1.6 «UML-діаграма шаблону « Strategy»»

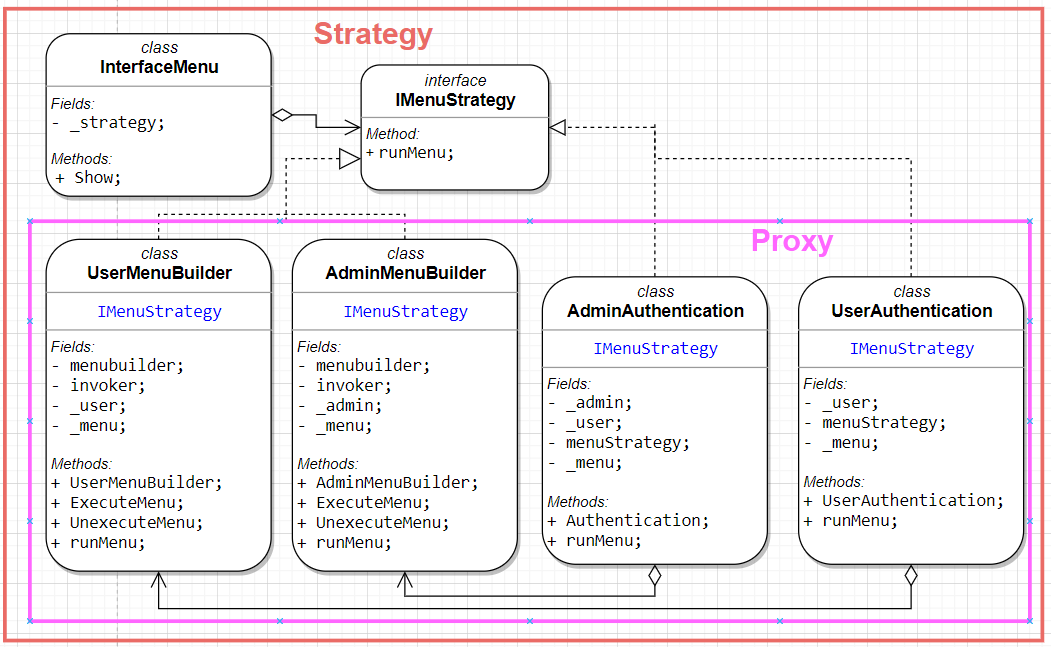
Стратегія — це поведінковий шаблон проектування, який визначає сімейство схожих алгоритмів і розміщує кожен з них у власному класі. Після цього алгоритми можна заміняти один на інший прямо під час виконання програми.

**Обґрунтування використання даного шаблону:**

Під час роботи програми треба мати можливість переходити із режиму користувача до режиму адміністратора. Для кожного з них передбачено різний інтерфейс і різний функціонал. Потрібно використовувати дві різні варіації одного і того ж алгоритму.

* 1. **Діаграма класів**





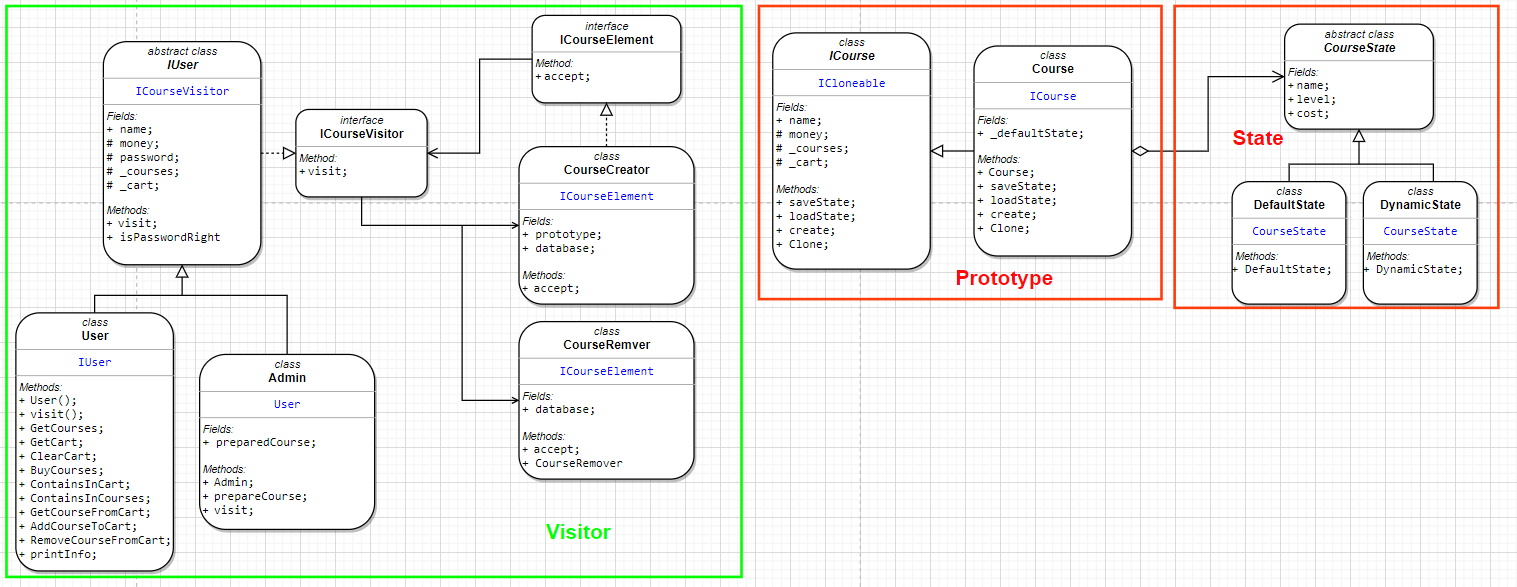


Рис. 2.2.1 «UML-діаграма класів »

**2.3 Результати роботи програми**

Робота з додатком починається з вибору ролі та сторінки автентифікації – обираємо роль адміністратора чи користувача, вводимо дані. Якщо введені дані невірні –програма про це повідомить, якщо вірні - надається доступ до програми.

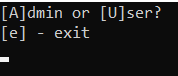
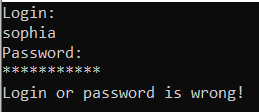
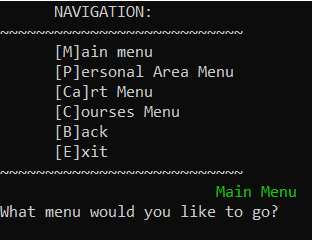


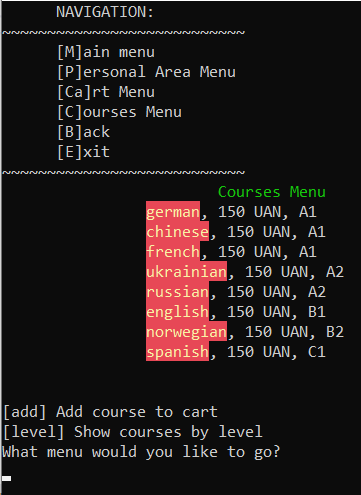
Рис. 2.3.1 «Admin/user»

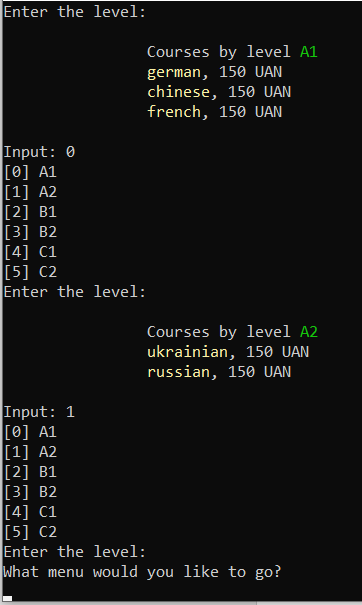
 Рис. 2.3.2 «Authentication»

Main Menu. До уваги користувача представлено 6 варіантів меню. Аби переходити з одного меню в інше, слід натискати букви, які зображено між дужками «[ ]».

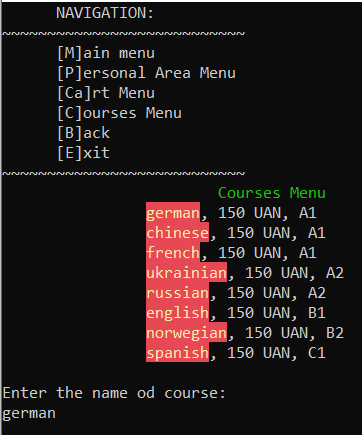
 Рис. 2.3.3 «User main menu»

Тиснемо «ca» + enter і переходимо у меню перегляду наявних курсів. Можемо додати потрібний курс до свого кошика, або обрати перегляд курсів відповідно до потрібного рівня.

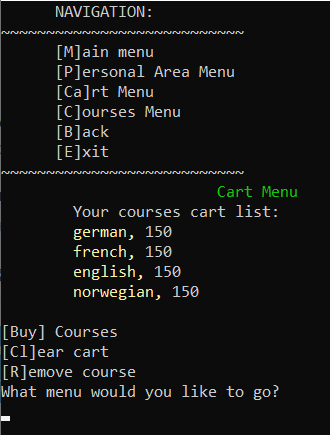
 Рис. 2.3.4 «Courses Menu»

 Рис. 2.3.5 «Levels Menu»

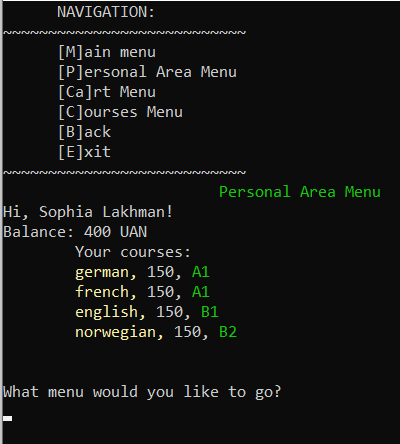
Ввівши команду «add», користувач потрапляє у меню додавання курса до свого кошика. Зауважимо, що обравши пункт [b]ack, користувач одразу повертається у попереднє меню.

 Рис. 2.3.6 «Courses Menu»

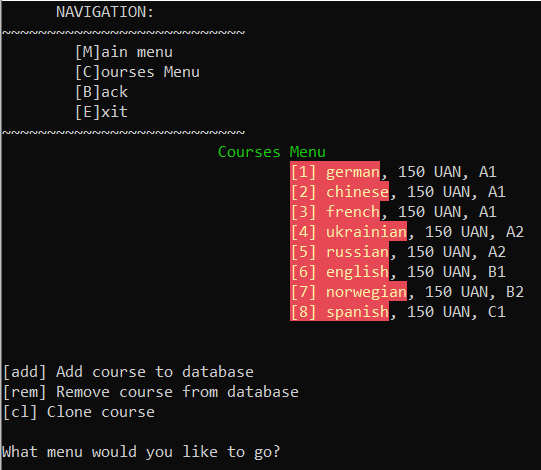
Переходимо у меню керування кошиком. Тут користувач може відредарувати вміст кошика: видалити об’єкти, очистити весь кошик, або купити наявні у ньому одиниці.

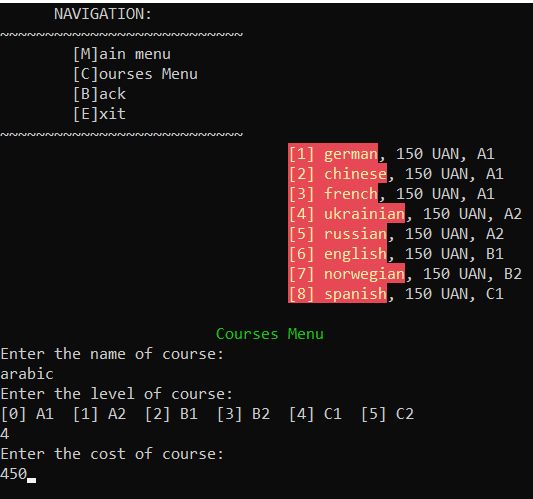
 Рис. 2.3.7 «Cart Menu»

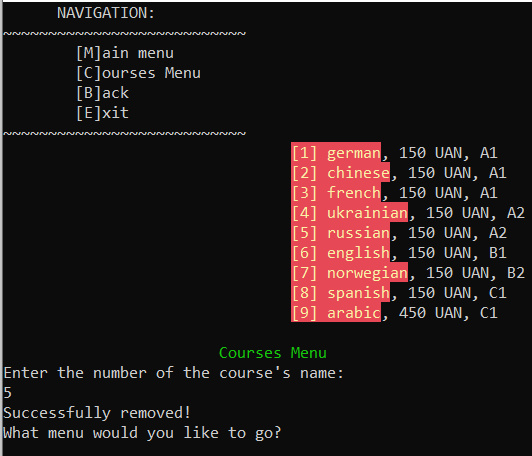
Купуємо все, що є у кошику. Після чого у меню особистого кабінету бачимо список придбаних курсів. Рахунок після цього також змінився.

 Рис. 2.3.8 «Levels Menu»

Якщо з меню вибору ролі обрати роль адміністратора, у меню курсів маємо:

 Рис. 2.3.9 «Admin Main Menu»

Можемо додати курс, видалити або клонувати вже існуючий:  Рис. 2.3.10 «Creating course»

 Рис. 2.3.11 «Removing course»

**ВИСНОВКИ**

Метою даної курсової роботи було розроблення програмного забезпечення мовної школи з використанням шаблонів проектування. Підставою для розроблення стало завдання на виконання курсової роботи з дисципліни «Об’єктно-орієнтоване програмування» студентами ІІ курсу кафедри ПЗКС НТУУ «КПІ ім.І.Сікорського».

Для досягнення поставленої мети у повному обсязі виконано завдання, визначені у аркуші завдання на курсову роботу; розроблено графічні матеріали; реалізовано всі вимоги до програмного продукту, програмного та апаратного забезпечення, наведені у технічному завданні; створено відповідну документацію.

Розроблене програмне забезпечення дозволяє користувачу у зручний спосіб переглядати, обирати та купувати курси, переглядати актуальну інформацію у особистому кабінеті. Більше того, адміністратор має можливість наповнювати та корегувати вміст бази даних курсів.

Програму створено на основі використання шаблонів проектування: зокрема, до структури програмного забезпечення входить реалізація семи шаблонів, які належать до різних груп шаблонів проектування (структурних, породжувальних та поведінкових).

Для розроблення програмного забезпечення була використана мова програмування C#.

Перспективним напрямком подальшого дослідження даної тематики є розроблення веб-сайту даного програмного забезпечення.

**СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ**

1. <http://design-pattern.ru>;
2. <https://refactoring.guru/uk/design-patterns>
3. Гамма, Э. Приемы объектно-ориентированого проектирования. Паттерны проектирования [Текст] / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. – СПб: Питер, 2009. – 366 с.
4. Э. Фримен, Э. Фримен, К. Сьерра, Б. Бейтс. Паттерны проектирования. — СПб.: Питер, 2011. — 656 с.: ил.