**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ТАРАСА ШЕВЧЕНКА**

**ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

**Кафедра інтелектуальних технологій**

**Лабораторна робота №10**

З дисципліни «Об'єктно-орієнтований аналіз та проєктування»

Тема роботи: «Поведінкові патерни GoF»

**Варіант №7**

Виконала студентка групи КН-32

Гнатюк Анна Михайлівна

Перевірила:

Красовська Ганна Валеріївна

**Київ – 2024**

**Мета:**

Навчитися застосовувати патерни проектування при розробці об’єктно- орієнтованого програмного забезпечення.

**Завдання:**

1. Розібрати та порівняти патерни за двома групами, що наведені у таблиці 2. Дати відповідь на запитання про доцільність застосування кожного з цих патернів та можливість їх взаємозв’язку (сумісного використання) чи взаємозамінності.

2. Самостійно для свого варіанту завдання придумати та програмно реалізувати по одному прикладу сумісного використання патернів з двох наведених груп.

**Виконання:**

**Інформаційна система міжгалактичного парку атракціонів**

1. **Результати об’єктно-орієнтованого проектування програмної системи.**
   1. **Діаграма класів системи з врахуванням доданих класів, що беруть участь в реалізації поведінкових патернів.**

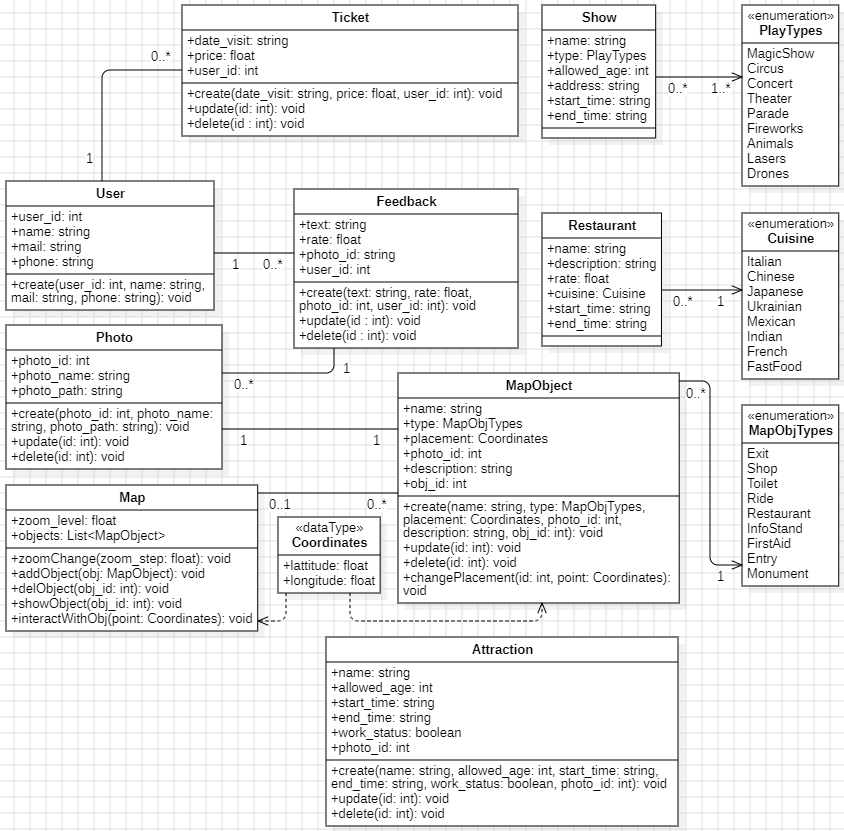


Рис. 1.1.1 – Діаграма класів системи

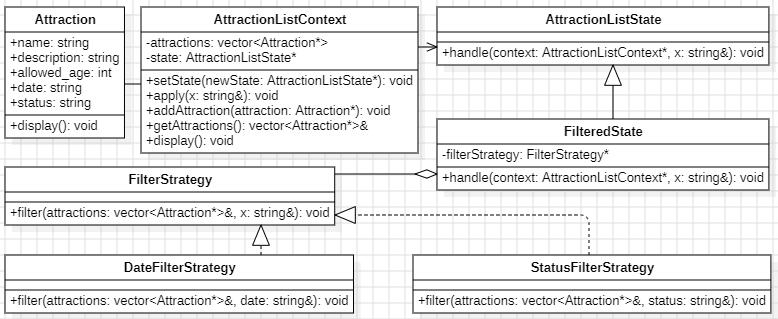


Рис. 1.1.2 – Діаграма класів що беруть участь в реаліз. ПП State + Strategy

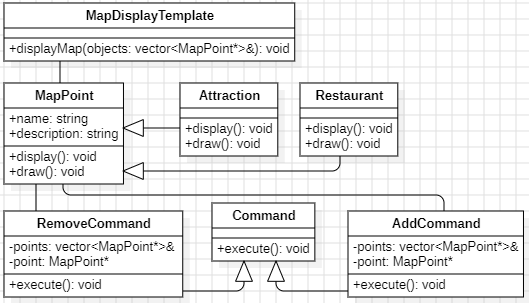


Рис. 1.1.3 – Діаграма класів що беруть участь в реаліз. ПП Template Method + Command

* 1. **Діаграма послідовностей для класів, що беруть участь в реалізації структурних паттернів.**

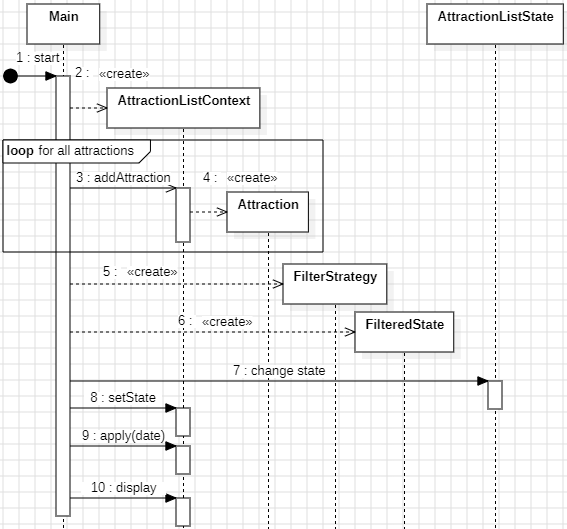


Рис. 1.2.1 – Діаграма послідовностей для класів що беруть участь в реаліз. ПП State + Strategy

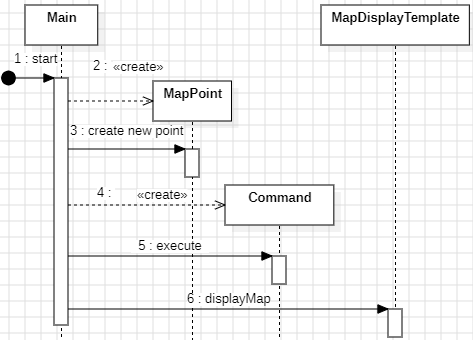


Рис. 1.2.2 – Діаграма послідовностей для класів що беруть участь в реаліз. ПП Template Method + Command

* 1. **Діаграма розгортання для розроблюваної системи з її описом.**

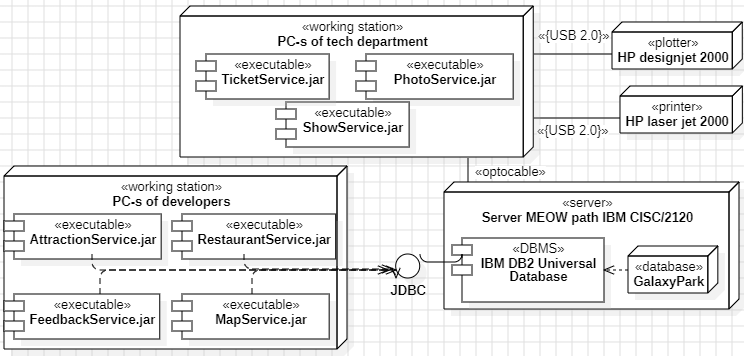


Рис. 1.3 – Діаграма розгортання для розроблюваної системи

**Вузли та компоненти:**

1. Робоча станція технічного відділу (PC-s of tech department)
   * Компоненти:
     + TicketService.jar
     + ShowService.jar
     + PhotoService.jar
2. Робоча станція розробників (PC-s of developers)
   * Компоненти:
     + AttractionService.jar
     + RestaurantService.jar
     + FeedbackService.jar
     + MapService.jar
3. Загальний сервер (Server MEOW path IBM CISC/2120)
   * Компоненти:
     + IBM DB2 Universal Database
   * База Даних:
     + GalaxyPark
4. Периферійні пристрої:
   * Плоттер (HP DesignJet 2000)
   * Принтер (HP LaserJet 2000)

**Зв'язки між компонентами:**

1. USB з'єднання:
   * Робоча станція технічного відділу підключена до плоттера (HP DesignJet 2000) через USB 2.0.
   * Робоча станція технічного відділу підключена до принтера (HP LaserJet 2000) через USB 2.0.
2. Оптоволоконне з'єднання (optocable):
   * Робочі станції (технічного відділу та розробників) підключені до загального сервера через оптоволоконне з'єднання.
3. JDBC з'єднання:
   * Загальний сервер використовує JDBC для взаємодії з базою даних IBM DB2 Universal Database.

**Функціональність та взаємодія:**

* Робоча станція технічного відділу:
  + Виконує сервіси для управління квитками, шоу та фото.
  + Підключена до принтера та плоттера для друку відповідних документів та зображень.
* Робоча станція розробників:
  + Виконує сервіси для управління атракціонами, ресторанами, відгуками та мапами.
  + Взаємодіє із загальним сервером для зберігання та отримання даних через базу даних.
* Загальний сервер:
  + Центральний вузол, що містить базу даних IBM DB2 Universal Database.
  + Виконує функцію зберігання та обробки даних для всієї системи.
  + Підключений до робочих станцій через оптоволоконне з'єднання та використовує JDBC для доступу до бази даних.
* База даних (GalaxyPark):
  + Містить всю необхідну інформацію для управління парком атракціонів.
  + Підключена до загального сервера для забезпечення зберігання та доступу до даних.

1. **Результати тестування програмної реалізації структурних патернів.** 
   1. **Розібрати та порівняти патерни за двома групами, що наведені у таблиці 1. Дати відповідь на запитання про доцільність застосування кожного з цих патернів та можливість їх взаємозв’язку (сумісного використання) чи взаємозамінності.**

|  |  |  |  |
| --- | --- | --- | --- |
| Перша група | | Друга група | |
| Mediator  State | Chain of responsibility | Template Method  Visitor | Strategy |
| Command | Memento |
| Observer | Visitor |
| Strategy | Command |

Таблиця 2.1.1 – завдання

|  |  |  |
| --- | --- | --- |
| **Паттерн** | **Опис** | **Доцільність застосування** |
| Mediator | Визначає об'єкт, що інкапсулює взаємодію множини об'єктів. Mediator робить систему слабо зв'язаною, позбавляючи об'єкти від необхідності посилатися один на одного, що дозволяє змінювати взаємодію між ними незалежно. | Використовується для зменшення взаємозалежностей між об'єктами та спрощення їх взаємодії​​. |
| State | Дозволяє об'єкту змінювати свою поведінку в залежності від стану. Створюється враження, що об'єкт змінив свій клас. Є об'єктно-орієнтованою реалізацією кінцевого автомата. | Використовується для реалізації кінцевого автомата, де об'єкт змінює поведінку при зміні свого стану​​. |
| Chain of responsibility | Дозволяє уникнути жорсткої залежності відправника запиту від його одержувача, при цьому запит може бути оброблений кількома об'єктами. Визначає конвеєрну обробку для запиту з множиною можливих оброблювачів. | Використовується, коли потрібно передавати запит по ланцюгу і обробити його один з об'єктів у ланцюгу​​. |
| Command | Інкапсулює запити як об'єкти, дозволяючи параметризувати клієнтів з чергами, запитами. Перетворює запит на виконання дії (команду) в об’єкт, який передається в інші методи для обробки/виконання. | Використовується для параметризації об'єктів з різними запитами, впровадження черги запитів, логів або скасування​​. |
| Observer | Визначає залежність один-до-багатьох між об'єктами так, що при зміні стану одного об'єкта всі залежні від нього об'єкти повідомляються і оновлюються автоматично. | Використовується для автоматичної нотифікації та оновлення залежних об'єктів при зміні стану одного об'єкта​​. |
| Strategy | Інкапсулює родини алгоритмів, що взаємозамінні. Всі алгоритми цього сімейства є спорідненими: призначені для вирішення спільних завдань, мають однаковий інтерфейс для використання і відрізняються тільки реалізацією (поведінкою). | Використовується, коли потрібно мати можливість вибору між різними алгоритмами чи стратегіями роботи​​. |
| Template Method | Визначає основу (кістяк) алгоритму і дозволяє підкласам змінити окремі кроки цього алгоритму без зміни його загальної структури. | Використовується, коли є загальний алгоритм з варіативними частинами, які можна реалізувати у підкласах​ |
| Visitor | Додає нові операції до класів без зміни самих класів. Визначає операцію, що виконується над кожним елементом з деякої структури (контейнера). | Використовується для виконання операцій над елементами структури об'єктів без зміни їх класів​​. |
| Memento | Захоплює і відновлює внутрішній стан об'єкта без порушення інкапсуляції. Є засобом для інкапсуляції "контрольних точок" програми. | Використовується для збереження та відновлення попереднього стану об'єкта, надання операціям "Скасування" (undo) або "Відкат" (rollback) статусу "повноцінного об'єкта"​ |

Таблиця 2.1.2 – Розбір паттернів

|  |  |  |  |
| --- | --- | --- | --- |
| **Паттерн 1** | **Паттерн 2** | **Можливість взаємозв’язку** | **Можливість взаємозамінності** |
| Mediator | Chain of responsibility | Mediator може використовувати Chain of Responsibility для делегування обробки запитів між об'єктами. | Не взаємозамінні, оскільки вирішують різні завдання: посередник координує, а ланцюг обробляє запити. |
| Mediator | Command | Mediator може координувати виконання команд, передаючи їх відповідним об'єктам для виконання. | Не взаємозамінні, оскільки Mediator координує, а Command інкапсулює запити як об'єкти. |
| Mediator | Observer | Mediator може використовувати Observer для динамічної реєстрації об'єктів і координації їх взаємодії. | Не взаємозамінні, оскільки Mediator координує взаємодію, а Observer нотифікує про зміни. |
| Mediator | Strategy | Mediator може вибирати та застосовувати різні стратегії для обробки запитів або координації дій об'єктів. | Не взаємозамінні, оскільки Mediator координує взаємодію, а Strategy визначає алгоритми. |
| State | Chain of responsibility | State може використовуватися для зміни оброблювачів у ланцюгу відповідно до поточного стану об'єкта. | Не взаємозамінні, оскільки вирішують різні завдання: State управляє станом, а Chain of Responsibility обробляє запити. |
| State | Command | State може змінювати виконання команд в залежності від поточного стану об'єкта. | Не взаємозамінні, оскільки State управляє станом, а Command інкапсулює запити. |
| State | Observer | State може використовувати Observer для сповіщення про зміну стану. | Не взаємозамінні, оскільки State управляє станом, а Observer нотифікує про зміни. |
| State | Strategy | State може вибирати стратегії в залежності від поточного стану об'єкта. | Не взаємозамінні, оскільки State управляє станом, а Strategy визначає алгоритми. |
| Template Method | Strategy | Template Method може використовувати стратегії для реалізації окремих кроків алгоритму. | Не взаємозамінні, оскільки Template Method визначає кістяк алгоритму, а Strategy реалізує алгоритми. |
| Template Method | Memento | Template Method може використовувати Memento для збереження проміжних станів під час виконання алгоритму. | Не взаємозамінні, оскільки Template Method визначає кістяк алгоритму, а Memento зберігає стан. |
| Template Method | Visitor | Template Method може використовувати Visitor для реалізації окремих кроків алгоритму. | Не взаємозамінні, оскільки Template Method визначає кістяк алгоритму, а Visitor додає нові операції. |
| Template Method | Command | Template Method може викликати команди як частину алгоритму. | Не взаємозамінні, оскільки Template Method визначає кістяк алгоритму, а Command інкапсулює запити. |
| Visitor | Strategy | Visitor може використовувати стратегії для реалізації відвідуваних операцій. | Не взаємозамінні, оскільки Visitor додає операції до класів, а Strategy визначає алгоритми. |
| Visitor | Memento | Visitor може зберігати стан об'єктів за допомогою Memento під час відвідування. | Не взаємозамінні, оскільки Visitor додає операції до класів, а Memento зберігає стан. |
| Visitor | Visitor | Visitor може використовувати інший Visitor для додавання різних операцій до класів. | Не взаємозамінні, оскільки кожен Visitor додає свої унікальні операції. |
| Visitor | Command | Visitor може виконувати команди під час відвідування об'єктів | Не взаємозамінні, оскільки Visitor додає операції до класів, а Command інкапсулює запити. |

Таблиця 2.1.3 – Аналіз взаємозв’язку паттернів

* 1. **Самостійно для свого варіанту завдання придумати та програмно реалізувати по одному прикладу сумісного використання паттернів з двох наведених груп.**

Для виконання завдання було обрано так дві пари паттернів:

State + Strategy

Template Method + Command

* + 1. **Реалізація сумісного використання паттернів State та Strategy**

State та Strategy можна реалізувати для фільтрації атракціонів, для варіанту використання та юзер сторі “Перегляд графіку роботи атракціонів”.

Так як суть Стейта, в зміні поведінки об’єкта в залежності від його стану, застосуємо його для фільтрації атракціонів (станами будуть типи фільтрації). Стейт буде дозволяти змінювати поведінку атракціонів в залежності від їхнього поточного стану (працює, не працює).

Так як суть Стратегії в інкапсуляції алгоритмів та можливості вибору між ними, застосуємо її для визначення поведінки атракціонів у різних станах. Стратегія буде дозволяти вибирати різні алгоритми для виконання дій атракціонів в залежності від їхнього стану (стратегія: відфільтрований за датою, відфільтрований за статусом).

Для реалізації Стейта, створимо стани фільтру (за датою, за статусом):

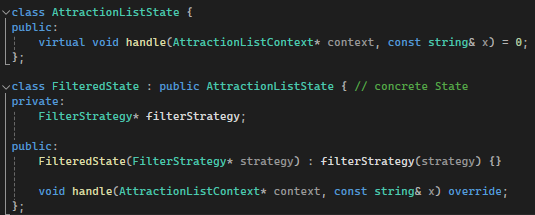


Рис.2.2.1.1 – використання Стейта

Для реалізації Стратегії, створимо різні стратегії для фільтрації атракціонів в залежності від їх станів:

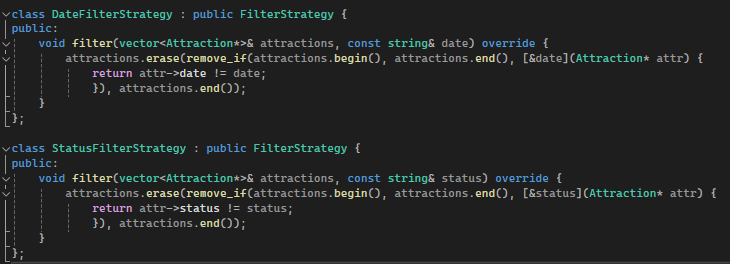


Рис.2.2.1.2 – використання Стратегії

Перевіримо роботу, ми повинні побачити повний список гірок, відфільтрований за статусом роботи (працює) та відфільтрований за датою (21.06.2004)

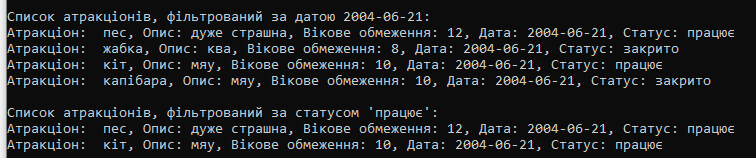


Рис.2.2.1.3 – результат роботи паттернів State + Strategy

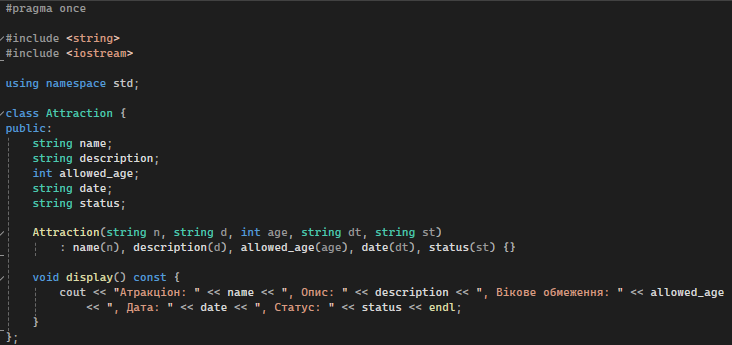


Рис.2.2.1.4 – Attraction.h

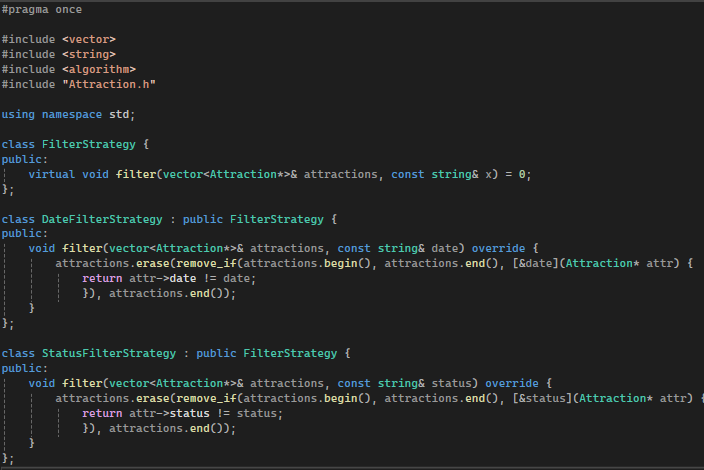


Рис.2.2.1.5 – FilterStrategy.h

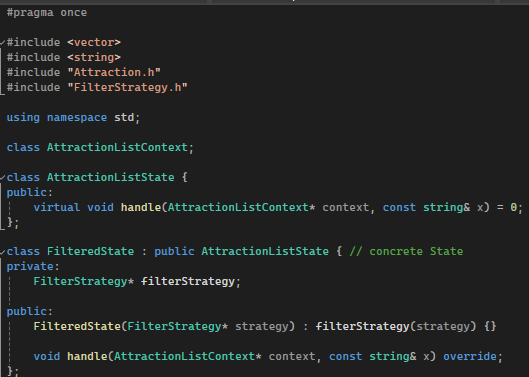


Рис.2.2.1.6 – AttractionListState.h

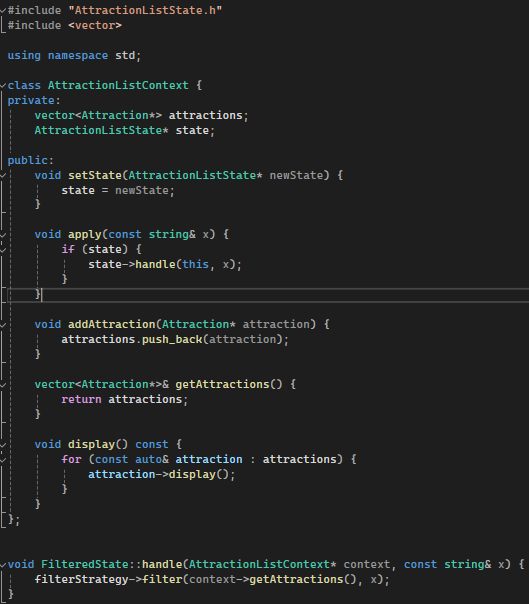


Рис.2.2.1.7 – AttractionListContext.h

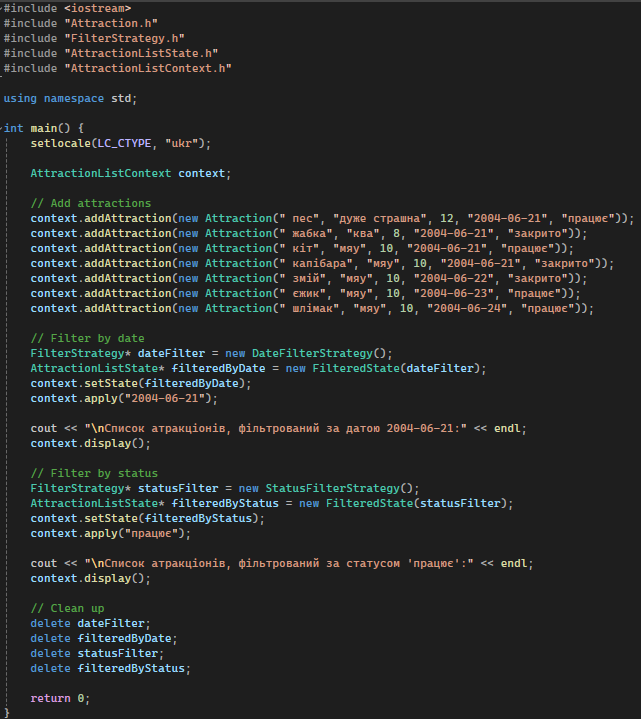


Рис.2.2.1.8 – main.cpp

* + 1. **Реалізація сумісного використання паттернів Template Method та Command**

Паттерни Темплейт Метод та Команду можна реалізувати для керування об’єктами для мапи, для варіанту використання та юзер сторі “Перегляд мапи парку”.

Так як суть Темплейт Метода, в тому щоб визначити кістяк алгоритму і дозволити падкласам змінювати окремі кроки цього алгоритму не змінюючи його основну структуру, ми можемо використати його для загального алгоритму відображення об’єктів на мапі.

Комманд можемо використати для реалізації команд додавання, оновлення і видалення точок на мапі.

Для реалізації Темплейта, задамо мапі кістяк: кожна точка яка є на мапі має відмальовуватись та виводитись(умовно в консоль).

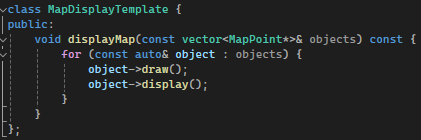


Рис.2.2.2.1 – застосування паттерну Темплейт Метод

Для реалізації Команд, реалізуємо видалення та додавання точок

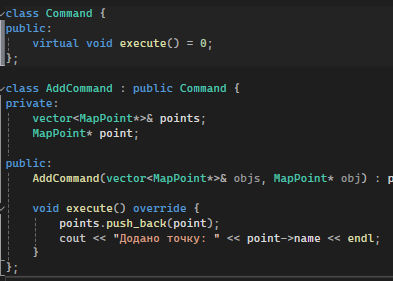


Рис.2.2.2.2 – застосування паттерну Комманд

Тепер потрібно протестувати роботу. Додамо одну точку. Виведемо мапу. Додамо ще одну точку. Виведемо мапу. Видалимо точку. Виведемо мапу

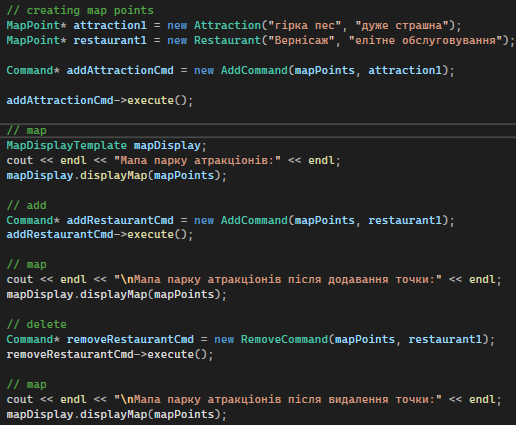


Рис.2.2.2.3 – застосування обох паттернів

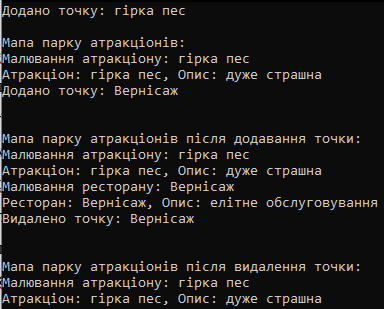


Рис.2.2.2.4 – результат роботи паттернів Template Method + Command

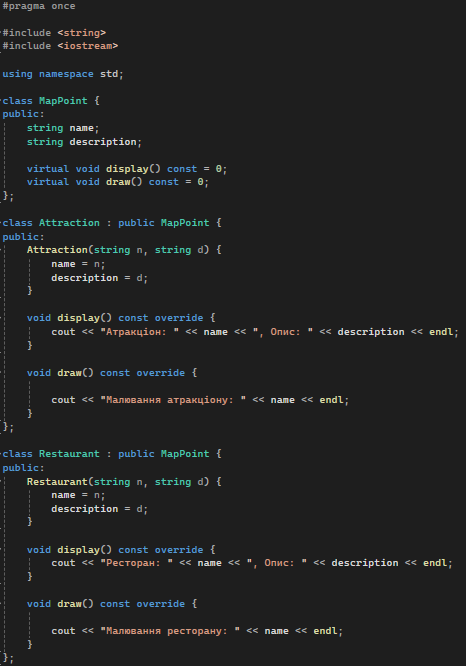


Рис.2.2.2.5 - MapPoint.h

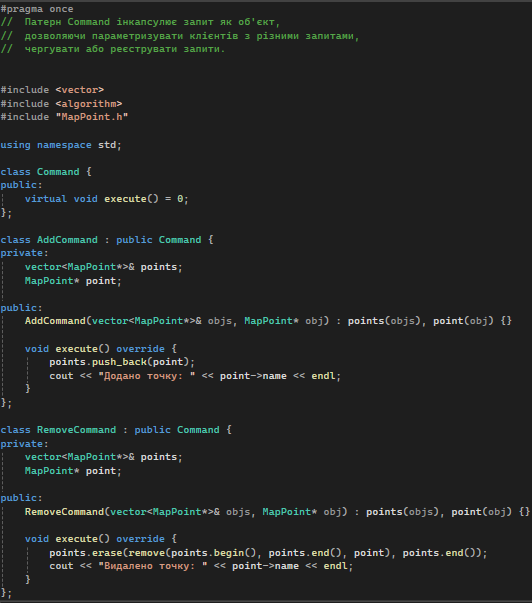


Рис.2.2.2.6 –Command.h

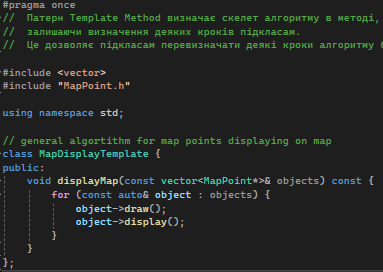


Рис.2.2.2.7 – TemplateMethod.h

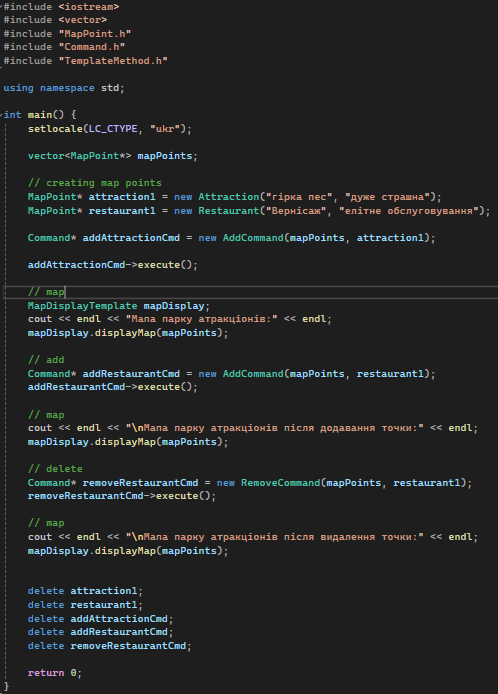


Рис.2.2.2.8 – main.cpp

1. **Програмний код.**

Програмний код наведено на рисунках.

Посилання на гітхаб:  
<https://github.com/ffructose/Lab10_OOAP>

**Висновки:**

В ході виконання даної лабораторної роботи було засвоєно застосування паттернів проєктування при розробці об’єктно- орієнтованого програмного забезпечення