



Міністерство освіти і науки України  
Національний технічний університет України  
«Київський політехнічний інститут» імені Ігоря Сікорського  
Факультет інформатики та обчислювальної техніки  
Кафедра Інформаційних Систем та Технологій

## **Лабораторна робота № 5**

з дисципліни: «Технології розроблення програмного забезпечення»

Виконав:

Тимчук Владислав

ІА-34

Перевірив:

Мягкий М. Ю.

**Тема:** Патерни проектування.

**Мета:** Вивчити структуру шаблонів «Adapter», «Builder», «Command», «Chain of responsibility», «Prototype» та навчитися застосовувати їх в реалізації програмної системи.

**Тема Лабораторного Практикуму:**

**Музичний програвач** (iterator, command, memento, facade, visitor, clientserver)

Музичний програвач становить собою програму для програвання музичних файлів або відтворення потокової музики з можливістю створення, запам'ятовування і редагування списків програвання, перемішування/повторення (shuffle/repeat), розпізнавання різних аудіоформатів, еквалайзер.

## **Вступ**

Метою цієї роботи є реалізація ще одної частини функціональності програмного забезпечення «Музичний програвач» з використанням шаблону проектування Command. У цій роботі попрацюємо на моделюванні користувацьких дій (відтворення треку, пауза, додавання треку до плейлиста) у вигляді окремих об'єктів-команд.

## **Зміст**

1. Діаграма класів
2. Код
  - 2.1. шаблон Command
  - 2.2. Фрагменти коду реалізації шаблону

**ВИСНОВОК**

**КОНТРОЛЬНІ ЗАПИТАННЯ**

## Хід роботи

### 1. ДІАГРАМА КЛАСІВ

Буде реалізовано невеликий модуль, який відповідає за виконання команд у музичному програвачі. Основна ідея:

- Користувацькі дії представляються як **об'єкти-команди**.
- Є інтерфейс **Command** з методом `execute()`.
- Класи **PlayTrackCommand**, **PauseCommand**,

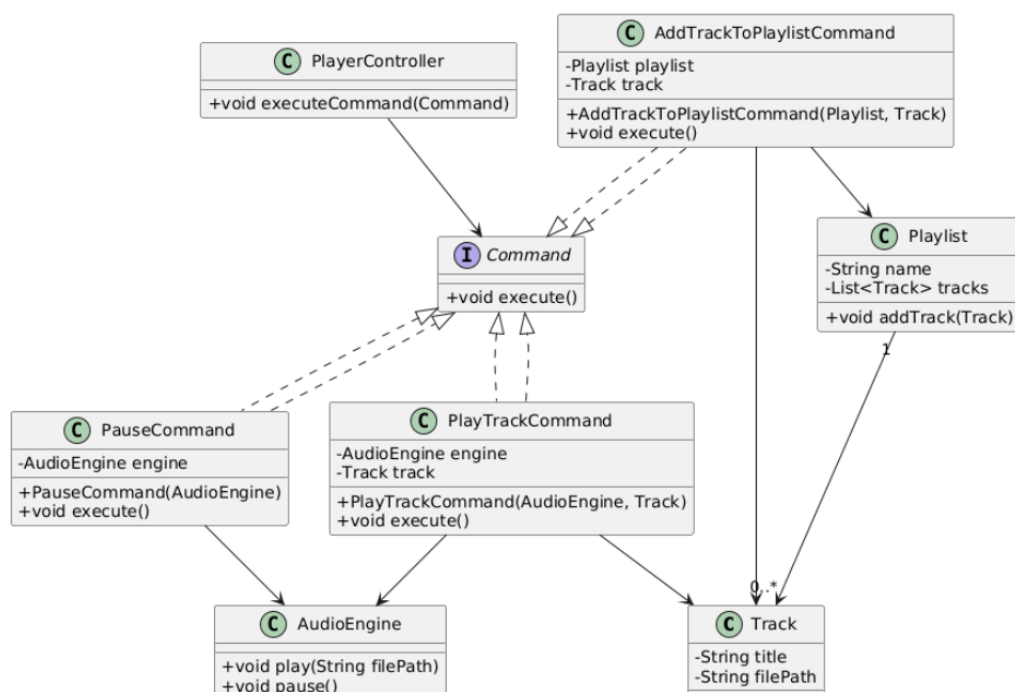
**AddTrackToPlaylistCommand** реалізують конкретні команди:

- **PlayTrackCommand** — відтворення конкретного треку.
- **PauseCommand** — пауза відтворення.
- **AddTrackToPlaylistCommand** — додавання треку до плейлиста.

• Є **Invoker** — клас **PlayerController**, який отримує команди й виконує їх, не знаючи деталей реалізації.

- **Receiver'и:**

- **AudioEngine** — відповідає за реальну логіку відтворення (`play/pause`).
- **Playlist** — колекція треків, в яку ми додаємо трек.



## 2. КОД

### 2.1. Шаблон Command

Шаблон **Command** дозволяє інкапсулювати запит (дію) в окремий об'єкт, відокремити відправника команди від її виконавця й спростити розширення системи новими діями (новими командами). Це особливо зручно в музичному плеєрі, де користувач викликає різні операції над треками й плейлистами (Play, Pause, Add, Remove, Next тощо).

### 2.2. Фрагменти коду реалізації шаблону

Увесь код знаходиться на віддаленому репозиторії:  
<https://github.com/fromz67/TRPZ/tree/main/lab5>

## ВИСНОВОК

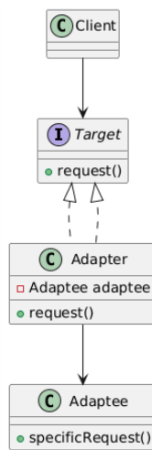
У ході лабораторної роботи було реалізовано частину функціональності музичного програвача з використанням шаблону проєктування Command. Користувацькі дії (відтворення треку, пауза, додавання треку до плейлиста) були представлені у вигляді окремих класів-команд, що інкапсулюють запити до відповідних об'єктів.

## КОНТРОЛЬНІ ЗАПИТАННЯ

### 1. Яке призначення шаблону «Адаптер»?

Шаблон «Адаптер» призначений для узгодження двох несумісних інтерфейсів: він «обгортає» об'єкт з одним інтерфейсом і надає інший інтерфейс, очікуваний клієнтом, щоб можна було використовувати існуючий код без його зміни.

### 2. Нарисуйте структуру шаблону «Адаптер».



### 3. Які класи входять в шаблон «Адаптер», та яка між ними взаємодія?

До шаблону «Адаптер» входять: **Client** (клієнтський код), **Target** (інтерфейс, який очікує клієнт), **Adaptee** (існуючий клас з «незручним» інтерфейсом) та **Adapter** (клас-перехідник); клієнт працює з Target, Adapter реалізує Target і всередині делегує виклики до Adaptee, перетворюючи один інтерфейс в інший.

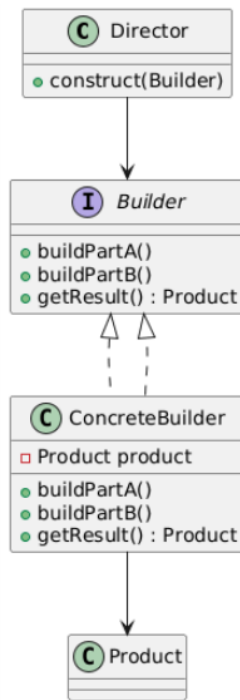
### 4. Яка різниця між реалізацією «Адаптера» на рівні об'єктів та на рівні класів?

Об'єктний адаптер використовує композицію: зберігає всередині посилання на Adaptee й делегує йому виклики, що гнучкіше й дозволяє адаптувати підкласи; класовий адаптер зазвичай базується на множинному наслідуванні (наслідує Target і Adaptee) і «перевизначає» поведінку, але жорсткіше прив'язаний до конкретних класів та можливостей мови.

### 5. Яке призначення шаблону «Будівельник»?

«Будівельник» призначений для поетапного конструювання складних об'єктів, коли створення має багато кроків і варіацій; він відділяє процес побудови від представлення, дозволяючи створювати різні варіанти об'єкта, використовуючи один і той самий процес.

### 6. Нарисуйте структуру шаблону «Будівельник».



7. Які класи входять в шаблон «Будівельник», та яка між ними взаємодія?

У «Будівельник» входять: **Director** (керує порядком виклику кроків побудови), **Builder** (інтерфейс кроків побудови), **ConcreteBuilder** (конкретний будівельник, що реалізує кроки й накопичує результат) та **Product** (кінцевий об'єкт); Director викликає методи Builder у певній послідовності, ConcreteBuilder поступово формує Product і повертає його через getResult().

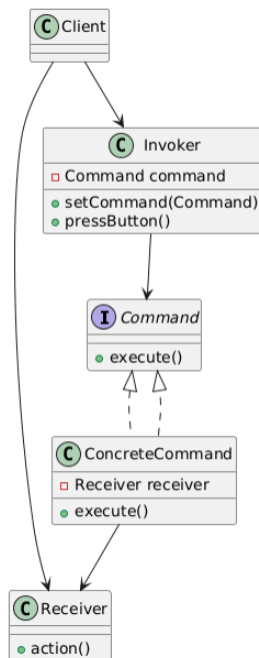
8. У яких випадках варто застосовувати шаблон «Будівельник»?

«Будівельник» доцільно застосовувати, коли об'єкт має багато параметрів або складну структуру, коли потрібні різні варіанти створення одного й того ж продукту, коли конструктори з великою кількістю параметрів стають незручними та коли треба чітко відокремити логіку побудови від готового об'єкта.

9. Яке призначення шаблону «Команда»?

Шаблон «Команда» інкапсулює запит (дію) в окремий об'єкт, щоб відокремити відправника команди від її виконавця, дозволити ставити команди в чергу, логувати, відмінити (undo) та комбінувати їх, не змінюючи клієнтський код.

10. Нарисуйте структуру шаблону «Команда».



### 11. Які класи входять в шаблон «Команда», та яка між ними взаємодія?

У шаблон «Команда» входять: **Command** (інтерфейс з методом `execute()`), **ConcreteCommand** (конкретна команда, зберігає посилання на **Receiver** і реалізує дію), **Receiver** (об'єкт, який безпосередньо виконує роботу), **Invoker** (ініціатор, який викликає команди), **Client** (створює команди і налаштовує зв'язки); **Client** створює **ConcreteCommand**, передає її **Invoker**'у, **Invoker** викликає `execute()`, а команда делегує дію **Receiver**'у.

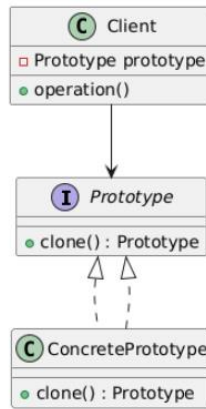
### 12. Розкажіть як працює шаблон «Команда».

Клієнтський код створює об'єкт-команду, прив'язуючи його до конкретного отримувача (**Receiver**), потім передає цю команду викликачеві (**Invoker**), який нічого не знає про внутрішню реалізацію й просто викликає метод `execute()`; усі параметри й інформація про те, як саме виконати дію, зберігаються всередині об'єкта-команди, що дозволяє легко додавати нові команди, логувати, ставити в чергу й реалізовувати `undo`.

### 13. Яке призначення шаблону «Прототип»?

«Прототип» призначений для створення нових об'єктів шляхом клонування існуючих екземплярів, а не через конструктор, що корисно, коли створення об'єкта «з нуля» дороге або складне, а також коли потрібно копіювати об'єкти з уже налаштованим станом.

### 14. Нарисуйте структуру шаблону «Прототип»



**15. Які класи входять в шаблон «Прототип», та яка між ними взаємодія?**

До шаблону «Прототип» входять: **Prototype** (інтерфейс з методом `clone()`), **ConcretePrototype** (конкретні реалізації, які вміють копіювати самі себе) та **Client** (використовує прототипи для створення нових об'єктів через клонування); клієнт зберігає посилання на прототип(и) і замість виклику конструктора просить `prototype.clone()`, отримуючи новий об'єкт з таким самим станом.

**16. Які можна привести приклади використання шаблону «Ланцюжок відповідальності»?**

Приклади: обробка HTTP-запиту в веб-фреймворках через послідовність фільтрів/ `middleware`; система логування з різними рівнями (`debug/info/error`), де кожен обробник вирішує, чи обробляти повідомлення; обробка подій у GUI, де подія «піднімається» по ієрархії елементів; перевірка прав доступу, коли кілька об'єктів по черзі вирішують, чи можуть обробити запит користувача.