

Національний технічний університет України  
«Київський політехнічний інститут імені Ігоря Сікорського»  
Факультет інформатики та обчислювальної техніки  
Кафедра інформаційних систем та технологій

## **ЛАБОРАТОРНА РОБОТА №7**

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Патерни проектування. FTP-server»

**Виконав:**

студент групи ІА-34

Сухоручкін Гліб

**Перевірив:**

Асистент кафедри ІСТ

Мягкий М.Ю.

## **Зміст:**

1. Вступ
2. Теоретичні відомості
3. Хід роботи
4. Діаграма класів
5. Вихідний код
6. Відповіді на питання до лабораторної роботи
7. Висновки

## Вступ

Метою цієї роботи є вивчити структуру шаблонів «Mediator», «Facade», «Bridge», «Template method» та навчитися застосовувати їх в реалізації програмної системи.

Проекторваною системою цієї лабораторної роботи є FTP-server (використовуючи state, builder, memento, template method, visitor, client-server).

FTP-сервер повинен вміти коректно обробляти і відправляти відповіді по протоколу FTP, з можливістю створення користувачів (з паролями) і доступних їм папок, розподілу прав за стандартною схемою (rwe), ведення статистики з'єднань, обмеження максимальної кількості підключень і максимальної швидкості поширення глобально і окремо для кожного облікового запису.

## Теоретичні відомості

### Шаблон «Template method»

**Призначення:** Шаблон «Template Method» (шаблонний метод) дозволяє реалізувати покроково алгоритм в абстрактному класі, але залишити специфіку реалізації підкласам. Можна привести в приклад формування вебсторінки: необхідно додати заголовки, вміст сторінки, файли, що додаються, і нижню частину сторінки. Код для додавання вмісту сторінки може бути абстрактним і реалізовуватися в різних класах – AspNetCompiler, HtmlCompiler, PhpCompiler і т.п. Додавання всіх інших елементів виконується за допомогою вихідного абстрактного класу з алгоритмом.

Даний шаблон дещо нагадує шаблон «Фабричний метод», однак область його використання абсолютно інша – для покрокового визначення конкретного алгоритму; більш того, даний шаблон не обов'язково створює нові об'єкти – лише визначає послідовність дій.

**Проблема:** Ви працюєте в команді, що займається розробкою застосунку для редагування відео-файлів. Застосунок вже працює з форматом відео MPEG-4, а саме дозволяє читати такі файли, виконувати попередню обробку даних для відображення в відео-редакторі.

Ви отримуєте нову задачу на реалізацію можливості роботи з більш старим форматом MPEG-2. Ви бачите два варіанта: зробити копію існуючого класу, що працює з MPEG-4, або вносити зміни в уже існуючий клас. Щоб прийняти рішення ви більш детально розбираєтеся з існуючим алгоритмом і бачите, що близько 70 відсотків коду має бути таким самим. Тому ви вирішуєте змінити вже існуючий клас для роботи з MPEG-4 додаючи в місцях де це потрібно умови з перевіркою, що якщо формат MPEG-2 то відпрацьовувати новий код, який ви добавили. Через деякий час, на запити від користувачів, вам на реалізацію приходить задача добавити підтримку ще більш старого формату MPEG-1. Ви вносите зміни так само в існуючий клас, тільки умови стали більш складними, тому що розгалуження логіки йде на три гілки.

Ще через деякий час приходить аналогічна задача на додавання читання даних з файлів формату H.262. Ви починаєте працювати над задачею і бачите, що код, який до цього був ще більш-менш зрозумілим стає зовсім важким для читання та внесення змін.

**Рішення:** Патерн «Шаблонний метод» (Template Method) пропонує загальний алгоритм винести в базовий клас, а частини алгоритма, які для різних задач виконуються по-різному, виділити в окремі методи. Ці методи будуть викликатися в алгоритмі, що реалізований в базовому класі. В дочірніх класах ці виділені методи будуть перевизначатися. Таким чином загальна логіка залишається в базовому класі, а специфічна частина реалізується в дочірніх класах.

Якщо подивитися на задачу з відео-редактором, то застосування «Шаблонного методу» наведе лад в коді і спростить його зміни.

Як це зробити: По перше, в алгоритмі всі блоки коду де є вибір гілки на основі типу формату виділяються в окремі методи. У випадку з відеоредактором, це скоріш за все будуть блоки коду пов'язані з читанням даних та розпакування їх в кадри, а також читання звукових доріжок. Далі створюється загальний базовий клас в який переноситься загальний алгоритм, а також об'являються віртуальні методи (фактично

беремо сигнатуру тих методів, що виділили на попередньому кроці). Далі створюємо дочірні класи під кожен формат файлу і перевизначаємо віртуальні методи. Фактично при цьому в кожному такому методі в дочірньому класі із реалізації цих методів, що була виділена на першому кроці, залишається код гілки який відповідав вибраному формату.

Після всіх цих змін ми маємо реалізацію патерна «Шаблонний метод»: в базовому класі реалізовано базовий алгоритм (по суті більша частина алгоритму) і в дочірніх класах перевизначені методи зі специфічною логікою.

Після таких змін, додати підтримку нового формату стає легше, тому що достатньо буде додати лише новий дочірній клас і перевизначити в ньому необхідні методи.

Слід зауважити, що якщо у вас алгоритми співпадають більше ніж на 50 відсотків, то застосування шаблонного методу буде доцільним, але якщо у вас алгоритми співпадають лише відсотків на 10 або 20, то скоріш за все, краще буде використати патерн «Стратегія».

### **Переваги та недоліки:**

- + Полегшує повторне використання коду.
- Ви жорстко обмежені скелетом існуючого алгоритму.
- Ви можете порушити принцип підстановки Барбари Лісков, змінюючи базову поведінку одного з кроків алгоритму через підклас.
- З ростом складності загального алгоритму шаблонний метод стає занадто складно підтримувати, особливо, коли є багато віртуальних методів для перевизначення в підкласах.

## Хід роботи

### Діаграма класів

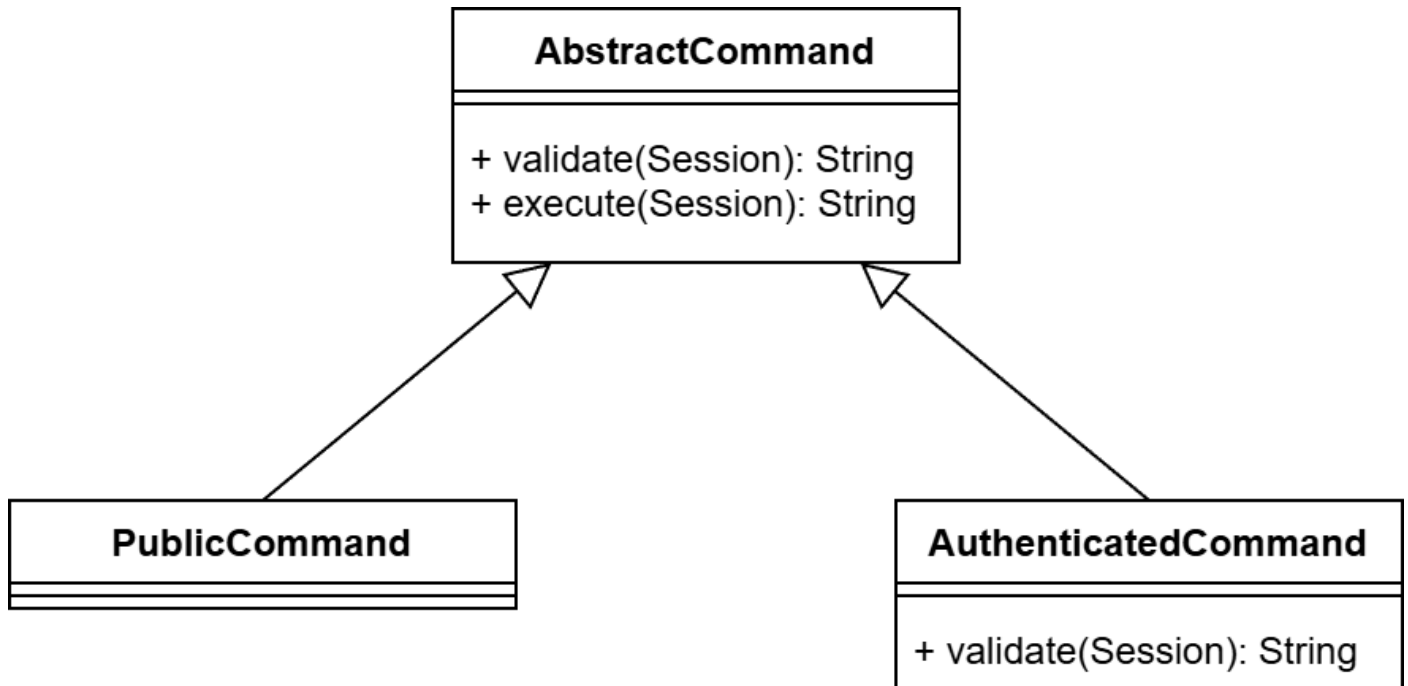


Рис 1. Патерн «Template method» у FTP-сервері.

### Пояснення діаграми класів

**AbstractCommand** – базовий абстрактний клас-шаблон. Містить шаблонний метод `final execute(Session): String`, який задає кістяк алгоритму: викликає гачок `validate(Session): String` (перевірки/попередні умови); якщо валідація повернула не `null`, одразу повертає це повідомлення; інакше викликає абстрактний крок `doExec(Session)` (реальна логіка команди).

**PublicCommand** – абстрактний підклас для публічних команд. Не перевизначає `validate(...)` (типово «по-ор»), тож такі команди виконуються без авторизації, але все одно проходять через однаковий шаблон `execute(...)`.

**AuthenticatedCommand** – абстрактний підклас для «захищених» команд. Перевизначає `validate(Session)` і повертає FTP-відповідь «530 Login required», якщо користувач не автентифікований. Далі, за успіху, керування переходить до `doExec(...)`.

### Вихідний код

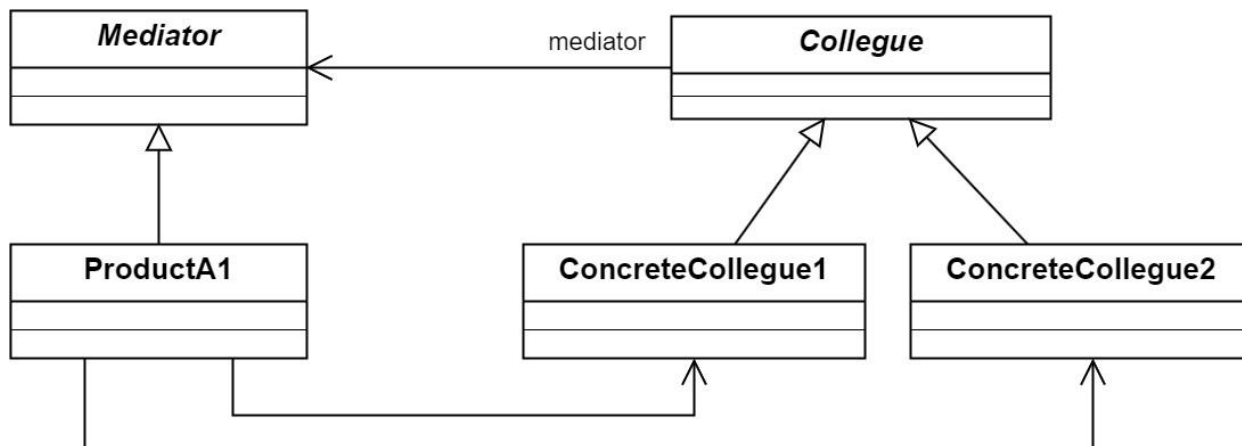
<https://github.com/GlebTiKsTRPZ/lab7>

## Відповіді на питання до лабораторної роботи

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

Визначати взаємодію об'єктів через окремий об'єкт-посередник (замість прямих посилань), що виносить логіку зв'язків у один центр.

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



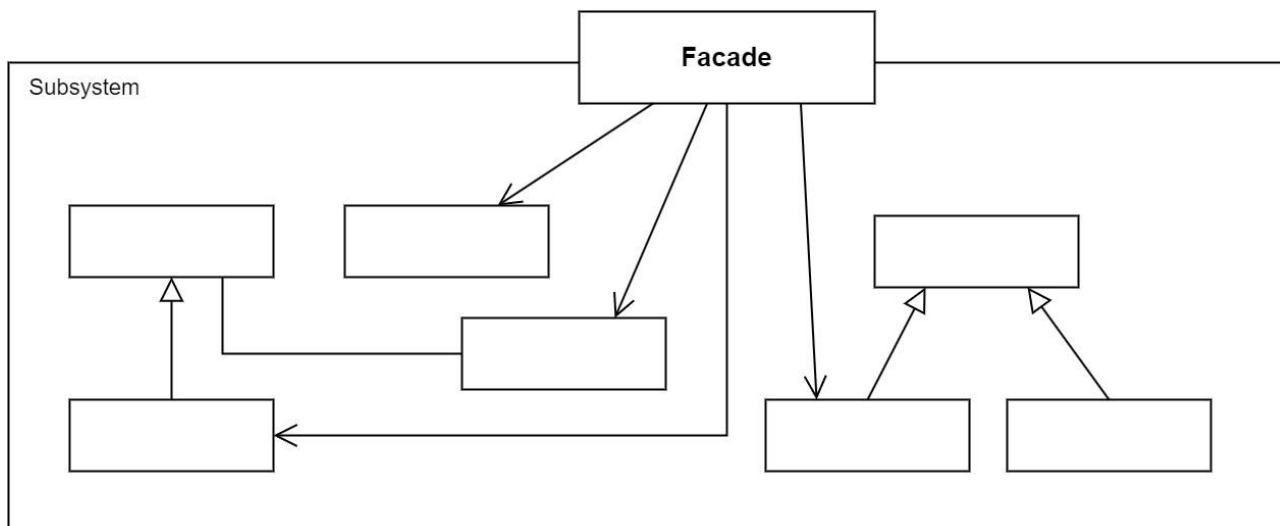
### 3. Які класи входять в шаблон «Посередник», та яка між ними взаємодія?

Є посередник та взаємодіючі компоненти (контролі), кожен компонент зберігає посилання на посередника і звертається до нього; посередник координує виклики між іншими компонентами; можливе використання спільного інтерфейсу посередника (його можна підміняти).

### 4. Яке призначення шаблону «Фасад»?

Надати єдиний уніфікований спосіб доступу до підсистеми без розкриття її внутрішніх деталей і зменшити «спагеті-код».

### 5. Нарисуйте структуру шаблону «Фасад».



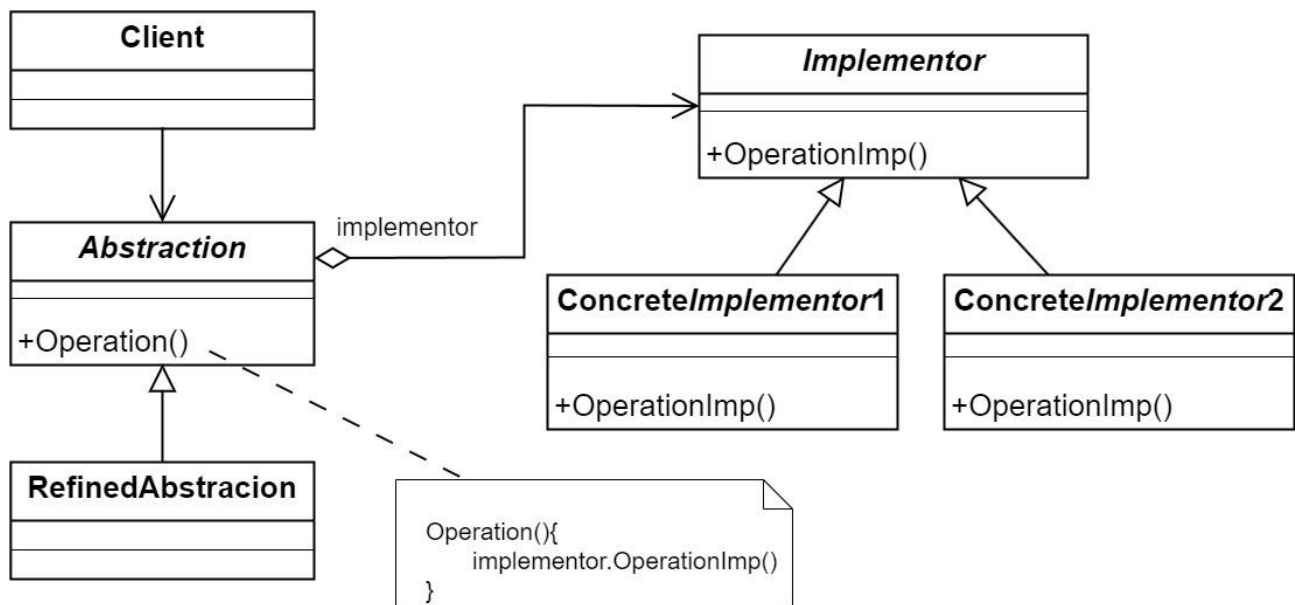
### 6. Які класи входять в шаблон «Фасад», та яка між ними взаємодія?

Один публічний клас-фасад (напр., InternetClient), який спрощує інтерфейс і делегує роботу внутрішнім класам підсистеми; клієнти взаємодіють тільки з фасадом, а внутрішня структура прихована (за потреби прямий доступ можливий, але не обов'язковий).

### 7. Яке призначення шаблону «Міст»?

Розділити абстракцію (інтерфейс) і її реалізації, щоб мати кілька абстракцій та виконувати дії різними способами.

## 8. Нарисуйте структуру шаблону «Міст».



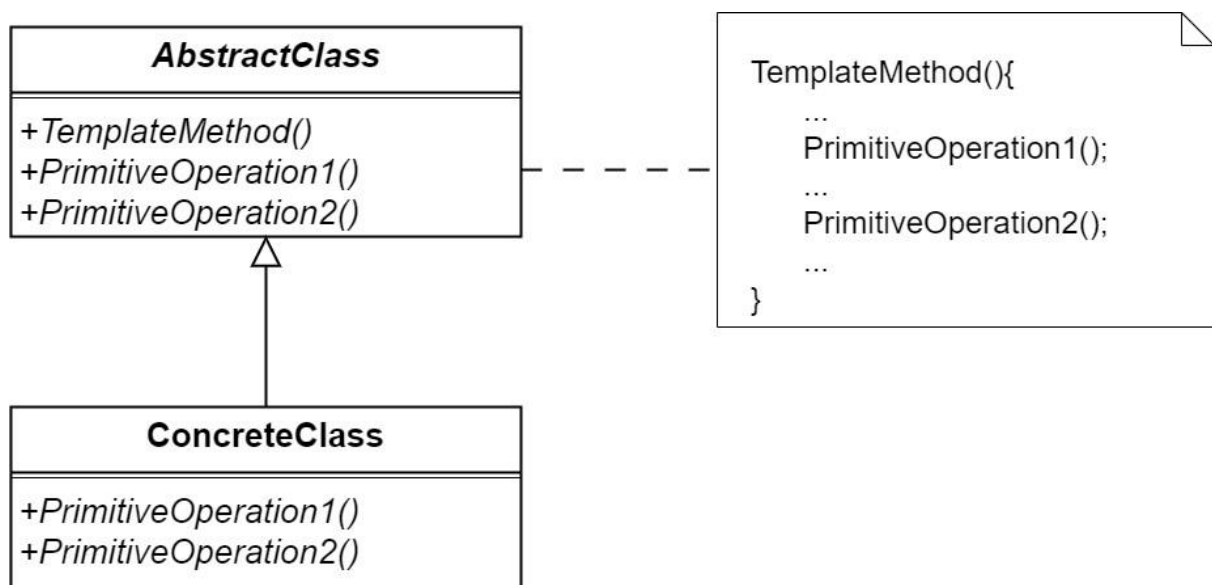
## 9. Які класи входять в шаблон «Міст», та яка між ними взаємодія?

Дві незалежні ієрархії – абстракцій (напр., Shape) та реалізацій (DrawApi). Абстракція агрегує DrawApi і делегує йому роботу; конкретні фігури є підкласами Shape, а конкретні драйвери (WindowDrawApi, PrinterDrawApi, BitmapDrawApi) – підкласами DrawApi; ієрархії розвиваються незалежно.

## 10. Яке призначення шаблону «Шаблонний метод»?

Зафіксувати кістяк алгоритму в базовому класі, а специфічні кроки залишити для перевизначення в підкласах; шаблон визначає послідовність дій і не обов'язково створює нові об'єкти.

## 11. Нарисуйте структуру шаблону «Шаблонний метод».



## 12. Які класи входять в шаблон «Шаблонний метод», та яка між ними взаємодія?

Базовий клас містить загальний алгоритм і виклики «гачків»/віртуальних методів; підкласи перевизначають ці методи, надаючи конкретну поведінку, тоді як



послідовність кроків залишається в базі.

### **13. Чим відрізняється шаблон «Шаблонний метод» від «Фабричного методу»?**

Попри схожість ідеї наслідування, «Шаблонний метод» призначений для покрокового визначення конкретного алгоритму (послідовності дій), тоді як «Фабричний метод» зосереджений на створенні об'єктів.

### **14. Яку функціональність додає шаблон «Міст»?**

Можливість незалежно варіювати та поєднувати абстракції й реалізації (напр., різні фігури x різні драйвери відображення), що підвищує гнучкість і спрощує супровід без роздування ієрархій.

## **Висновок**

У цій лабораторній роботі я опанував ключові ідеї шаблонів Mediator, Facade, Bridge та Template Method, та було застосовано шаблон Template Method у проєкті FTP-серверу, що дозволило уніфікувати виконання команд і підвищити зрозумілість архітектури.