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 №7
Технології розроблення програмного забезпечення
«Патерни проєктування»

Виконав
студент групи ІА–34:
Бородай А.С.

Київ 2025

Зміст

Теоретичні відомості

Шаблон «Mediator»

Призначення патерну «Mediator»: використовується для визначення взаємодії об'єктів за допомогою іншого об'єкта (замість зберігання посилань один на одного). Даний шаблон схожий на шаблон «команда», проте в даному випадку замість зберігання даних про конкретну дію, зберігаються дані про взаємодії між компонентами.

Даний шаблон зручно застосовувати у випадках, коли безліч об'єктів взаємодіє між собою деяким структурованим чином, однак складним для розуміння. У такому випадку вся логіка взаємодії виноситься в окремий об'єкт. Кожен із взаємодіючих об'єктів зберігає посилання на об'єкт «медіатор».

«Медіатор» нагадує диригента при управлінні оркестром. Диригент стежить за тим, щоб кожен інструмент грав в правильний час і в злагоді з іншими інструментами. Функції «медіатора» повністю це повторюють.

Проблема:

Ви розробляєте систему зі складною візуальною частино. Головна форма з якою працює користувач складається з великою кількості візуальних компонентів, які в свою чергу складаються з елементарних візуальних компонентів. Для такої складної форми, як правило, є багато логіки, коли дії в одному компоненті повинні впливати на інші компоненти.

Рішення:

В таких ситуаціях дуже добре підходить патерн «Mediator» (Посередник). При реалізації такого патерна, ми добавляємо новий клас посередник, і всі взаємодії між компонентами повинні відбуватися вже через нього. За рахунок цього компоненти вже не знають один про одного, а лише знають про посередника і для відпрацювання логіки звертаються вже до нього. А вже посередник взаємодіє з іншими компонентами для відпрацювання цієї логіки.

Переваги та недоліки:

+ Організація взаємодії між об’єктами лише через посередника спрощує розуміння та супроводження такого коду.

+ Додавання нових посередників без зміни існуючих компонентів дозволяє розширювати систему без зміни існуючого коду.

+ Зменшення залежностей між об’єктами підвищує гнучкість системи.

- Посередник, з часом, може перетворитися на дуже складний об’єкт, який робить все («God Object»).

Шаблон «Facade»

Призначення «Facade»: передбачає створення єдиного уніфікованого способу доступу до підсистеми без розкриття внутрішніх деталей підсистеми. Оскільки підсистема може складатися з безлічі класів, а кількість її функцій – не більше десяти, то щоб уникнути створення «спагеті-коду» (коли все тісно пов'язано між собою) виділяють один загальний інтерфейс доступу, здатний правильним чином звертатися до внутрішніх деталей. Це також відволікає користувачів від змін в підсистемі (внутрішня реалізація може змінюватися, а наданої послуги немає), що також скоротить кількість змін в використовуваних фасад класах (без фасаду довелося б змінювати вихідні коди в безлічі точок).

Проблема: Ви розробляєте компонент, який дозволяє відправляти запити на різні типи endpoint, а також працює з протоколами HTTP та TCP/IP.

Прототип компонента вже працює, але структура класів вийшла досить складна, а при налаштуванні на різні протоколи мають використовуватися різні класи.

Інструкція для використання також виходить досить складна та заплутана. Слід добавити, так як інші системи будуть знати про внутрішню будову вашого компонента, а тому при спробі змінити внутрішню структуру в наступних версіях, вам прийдеться повідомляти про це всіх користувачів вашого компонента і для них перехід на наступну версію вашого компонента буде достатньо складним.

Рішення: Тут краще використати патерн фасад. А саме, в даному випадку, створити один клас, наприклад, InternetClient, та набір методів у нього, які будуть використовуватися для налаштування цього підключення, та його подальшого використання. Цей клас єдиний буде позначено як public, і тільки його будуть бачити ваші клієнти. Таким чином, з точки зору зовнішнього коду вони будуть працювати з

набагато простішим інтерфейсом, а значить і інструкція використання буде значно простіша. Крім того, так як внутрішня структура повністю закрита, то в наступних версіях ви можете її змінювати, як вам буде зручно, а з точки зору користувачів компонента, перехід на нову версію буде просто переключенням на нову версію бібліотеки.

Переваги та недоліки:

+ Інкапсуляція внутрішньої структури від клієнтського коду.

+ Спрощується інтерфейс для роботи з модулем закритим фасадом.

- Зниження гнучкості в налаштуванні та використанні програмного коду закритого фасадом.

Шаблон «Bridge»

Призначення «Bridge»: використовується для поділу інтерфейсу і його реалізації. Це необхідно у випадках, коли може існувати кілька різних абстракцій, над якими можна проводити дії різними способами.

Проблема: Ви реалізовуєте графічний векторний редактор, який дозволяє рисувати круги, прямокутники, прямі та довільні лінії.

Ви маєте реалізувати функціонал відображення отриманого рисунку на екрані та друкувати на принтер.

Спростимо ситуацію: ваш редактор дозволяє рисувати лише лінії та круги.

При такому підході, нам потрібно будувати ієрархію фігур з різною реалізацією дочірніх класів: LinePrint, LineDraw, CirclePrint, CircleDraw. якщо добавити прямі, то добавиться ще два підкласи, і т.д. А як бути, коли нам потрібно буде ще і реалізувати збереження в bitmap форматі? добавляємо ще LineBinery, CircleBinery? При такому підході ми отримуємо дуже складну ієрархію класів.

Рішення: В даному випадку ми можемо використати патерн «Міст» (Bridge): робимо дві ієрархії – фігур (Shape) та рисування (DrawApi).

При такому підході DrawApi – це інтерфейс імплементації відображення (графічного драйвера), а Shape – інтерфейс абстракції фігур, яка має агрегацію з

об'єктом DrawApi. При такому підході фігури будуть делегувати рисування об'єкту DrawApi.

Лінія, коло, та інші будуть дочірніми класами до Shape, а WindowDrawApi та PrinterDrawApi – дочірні класи до DrawApi, які представляють графічні драйвери для відображення на екрані та принтері відповідно. Якщо нам потрібно буде добавити ще і збереження в bitmap форматі, то ми добавимо ще один підклас реалізації графічного драйвера BitmapDrawApi. Таким чином ми маємо дві різні ієрархії об’єктів і вони в нас не перетинаються і не збільшуються в геометричній прогресії при додаванні нових драйверів або фігур. Також слід відмітити, що DrawApi нічого не знає про фігури (абстракцію), а дочірні класи абстракції не залежать від реалізації графічного драйвера.

Переваги та недоліки:

+ Дозволяє змінювати ієрархії абстракції та реалізації незалежно одна від одної.

+ Розділивши абстракцію від реалізації отримуємо більшу гнучкість та простіший супровід такого коду.

- Підвищена гнучкість при використанні патерну отримується за рахунок більшої складності, введення додаткових проміжних рівнів.

Шаблон «Template Method»

Призначення «Template Method»: дозволяє реалізувати покроково алгоритм в абстрактному класі, але залишити специфіку реалізації підкласам. Можна привести в приклад формування веб-сторінки: 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.

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 добавляючи 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: «Шаблонний метод» 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.

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.

Тема: Патерни проектування.

Мета: Вивчити структуру шаблонів «Mediator», «Facade», «Bridge», «Template method» та навчитися застосовувати їх в реалізації програмної системи.

Завдання:

• Ознайомитись з короткими теоретичними відомостями.

• Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

• Реалізувати один з розглянутих шаблонів за обраною темою.

• Реалізувати не менше 3-х класів відповідно до обраної теми.

• Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє використання шаблону в реалізації системи, навести фрагменти коду по реалізації цього шаблону.

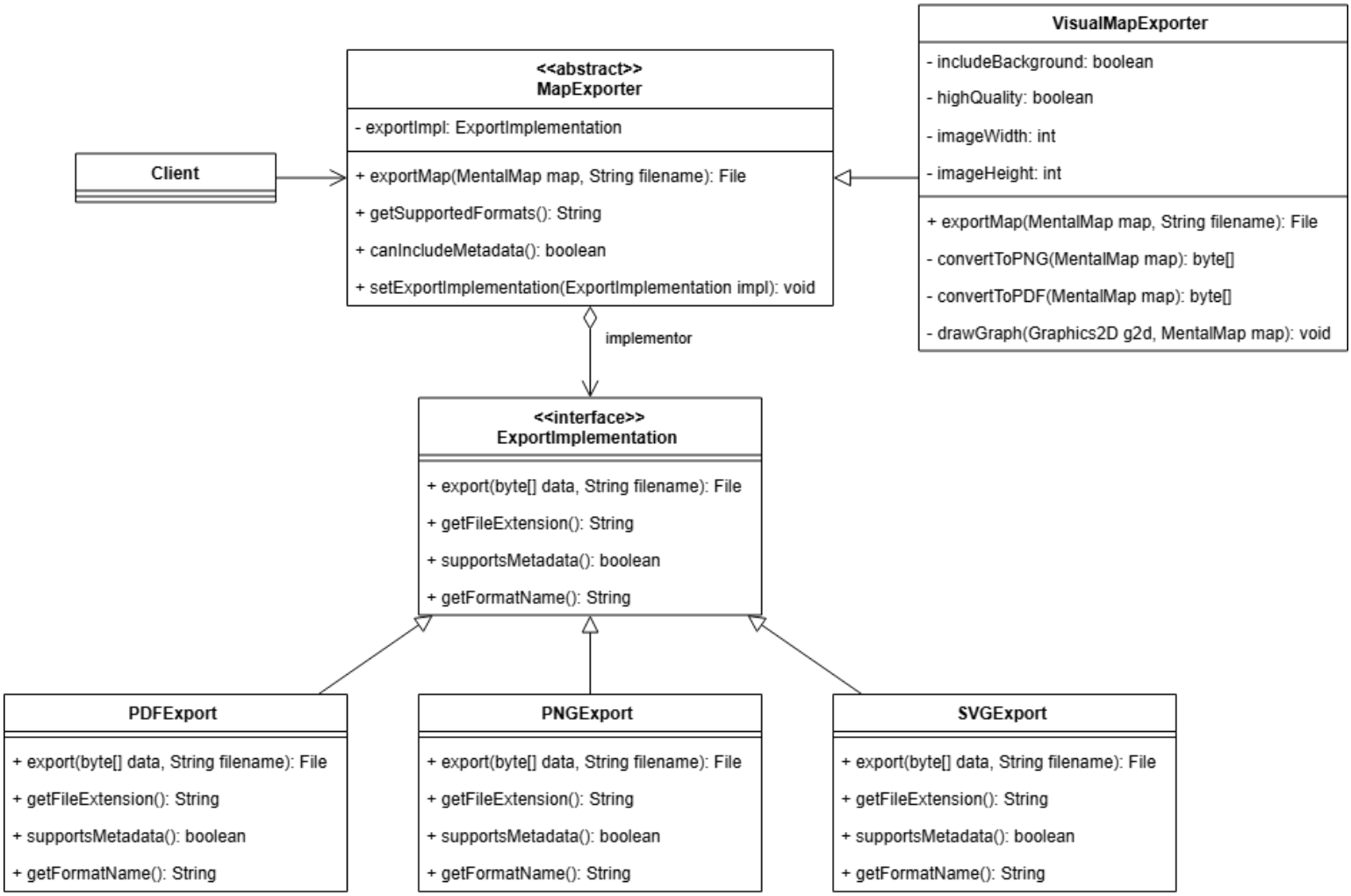
Хід роботи

20. **Mind-mapping software**

(strategy, prototype, abstract factory, bridge, composite, SOA)

Візуальний додаток для складання "карт пам'яті" з можливістю роботи з декількома картами (у вкладках), автоматичного промальовування ліній, додавання вкладених файлів, картинок, відеофайлів (попередній перегляд); можливість додавання значків категорій / терміновості, обведення областей карти (поділ пунктирною лінією).

ClassDiagram (Bridge)



VisualMapExporter

MapExporter

ExportImplementation

PDFExport

PNGExport

SVGExport

ExportException

Висновок

Під час виконання лабораторної роботи я ознайомився із шаблонами «Mediator», «Facade», «Bridge», «Template method». Мною були розроблені основні класи, які реалізують шаблон Bridge та його поведінку.

Використовуючи шаблон було розділено дві ієрархії (ментальні карти та експортери). Це зменшило кількість класів із 9 (3 ментальні карти по 3 експортери) до 6 (3 ментальні карти і 3 експортери). Також при додаванні нових типів карт чи експортерів необхідно реалізувати менше коду.

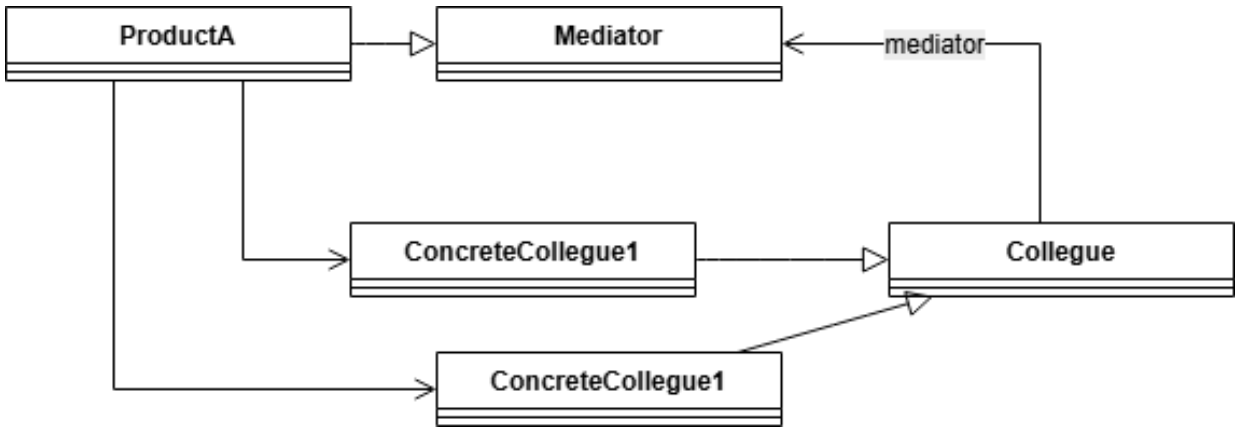
Нові шаблони будуть мною використані для подальшої реалізації системи та подальшого її розуміння.

Відповіді на питання для самоперевірки

1. Яке призначення шаблону «Посередник»?

Шаблон зменшує зв’язність між об’єктами, виступаючи вузлом комунікації. Замість того щоб об’єкти напряму викликали методи один одного, вони спілкуються через посередника. Це спрощує підтримку та розширення системи.

2. Нарисуйте структуру шаблону «Посередник».



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

Mediator – інтерфейс, який визначає методи спілкування між об’єктами.

ProductA – конкретна реалізація посередника.

Colleague – абстрактний клас або інтерфейс учасників.

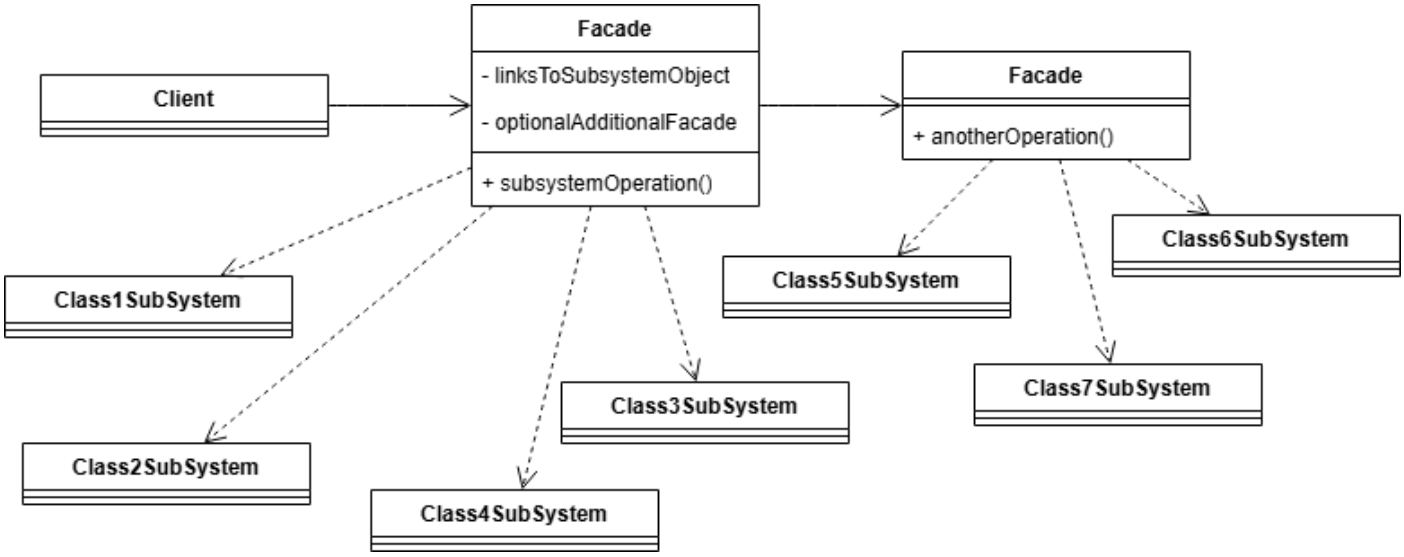
ConcreteColleague – конкретні об’єкти, що взаємодіють через посередника.

Взаємодія: колеги надсилають повідомлення посереднику, який визначає, кому і як їх передати.

4. Яке призначення шаблону «Фасад»?

Спростити доступ до складної системи, надаючи єдиний інтерфейс. Він приховує деталі реалізації та зменшує залежність клієнта від підсистем.

5. Нарисуйте структуру шаблону «Фасад».



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

Façade – надає простий інтерфейс до складної системи.

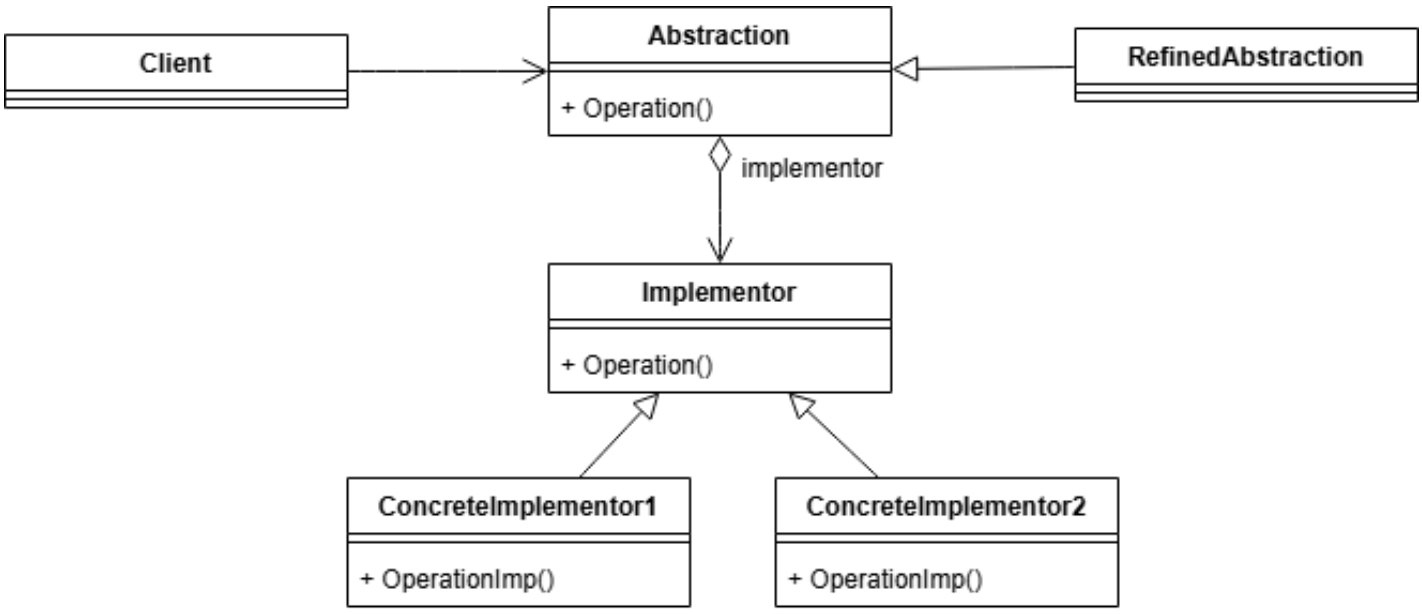
Subsystem classes – конкретні підсистеми з власними функціями.

Клієнт звертається лише до фасаду, а фасад делегує виклики відповідним підсистемам.

7. Яке призначення шаблону «Міст»?

Розділяє абстракцію і реалізацію, щоб їх можна було змінювати незалежно. Це дозволяє уникнути множинного наслідування і спрощує розширення системи.

8. Нарисуйте структуру шаблону «Міст».



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

Abstraction – визначає базовий інтерфейс абстракції.

RefinedAbstraction – розширює або деталізує абстракцію.

Implementor – інтерфейс реалізації.

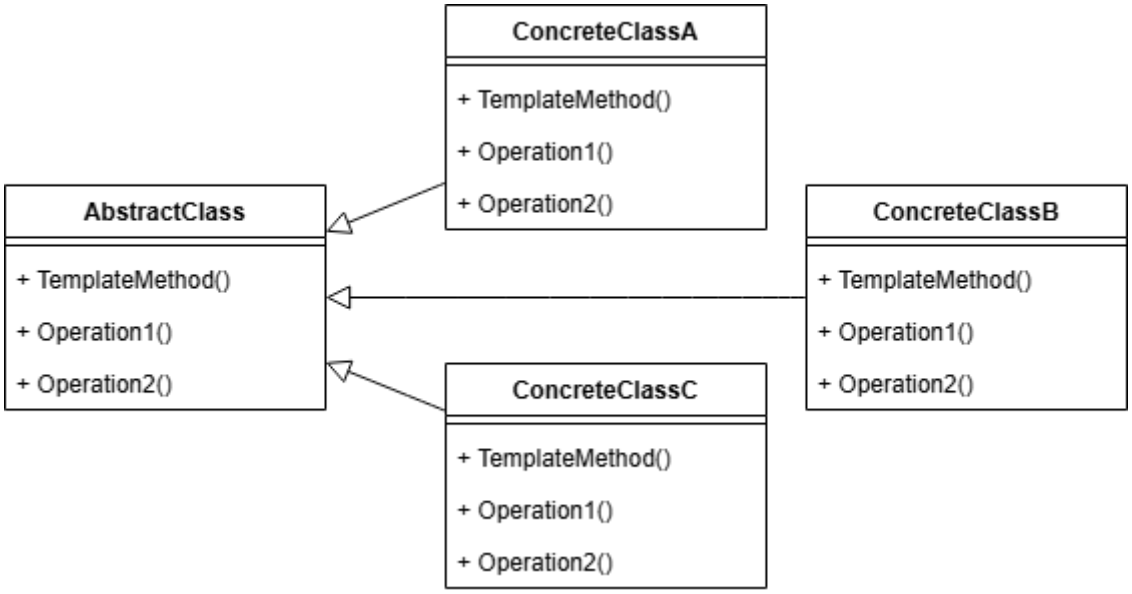
ConcreteImplementor – конкретна реалізація.

Абстракція делегує операції реалізатору через посилання на інтерфейс реалізації.

10. Яке призначення шаблону «Шаблоний метод»?

Задати каркас алгоритму в базовому класі та дозволити підкласам перевизначати окремі кроки, не змінюючи структуру всього алгоритму.

11. Нарисуйте структуру шаблону «Шаблоний метод».



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

AbstractClass – визначає шаблон методу (послідовність кроків).

ConcreteClass – реалізує конкретні кроки (операції).

Клієнт викликає шаблоний метод, який виконує фіксовану послідовність кроків, частково визначених у підкласах.

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

Шаблонний метод визначає алгоритм і дозволяє змінювати частини наслідуваного алгоритму (змінюються кроки алгоритму), тоді як фабричний метод наслідує створювача, визначає, який саме об’єкт створювати (змінюється типи об’єктів)

14. Яку функціональність додає шаблон «Міст»?

- Розділяє абстракцію від реалізації.
- Дозволяє незалежно розширювати обидві частини.
- Зменшує кількість класів, уникаючи комбінацій «абстракція × реалізація».
- Полегшує підтримку та тестування.