Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Лабораторна робота №7

Технології розроблення програмного забезпечення

«*Патерни проектування. Паттерн «Facad»*»

«10. VCS all-in-one»

Виконав
студент групи ІА–32:
Варивода К. С.

Перевірив:

Мягкий Михайло Юрійович

Київ 2025

Зміст

Варіант

10. VCS all-in-one (iterator, adapter, factory method, facade, visitor, p2p)

Клієнт для всіх систем контролю версій повинен підтримувати основні команди і дії (commit, update, push, pull, fetch, list, log, patch, branch, merge, tag) для 3-х основних систем управління версіями (svn, git, mercurial), а також мати можливість вести реєстр репозиторіїв (і їх типів) і відображати дерева фіксації графічно.

Теоретичні відомості

Будь-який патерн проєктування, використовуваний при розробці інформаційних систем, являє собою формалізований опис, який часто зустрічається в завданнях проєктування, вдале рішення даної задачі, а також рекомендації по застосуванню цього рішення в різних ситуаціях. Крім того, патерн проєктування обов'язково має загальновживане найменування. Правильно сформульований патерн проєктування дозволяє, відшукавши одного разу вдале рішення, користуватися ним знову і знову.

Призначення патерну: Шаблон «Facade» (фасад) 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). Звичайно, твердої умови повного закриття внутрішніх класів підсистеми не стоїть – при необхідності можна звертатися до окремих класів безпосередньо, минаючи об'єкт фасад.

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.

Релевантність до проекту:

В моєму проекті цей патерн є корисним, так як він дає можливість приховати логіку конкретних класів, зробити єдину вхіжну точку та дати той набір функцій, який я хочу давати наверх, наприклад для ui. Також інколи є потреба надбудувати логіку над вже існуючим методом, тоді цей патерн також є гарним рішенням

Хід роботи

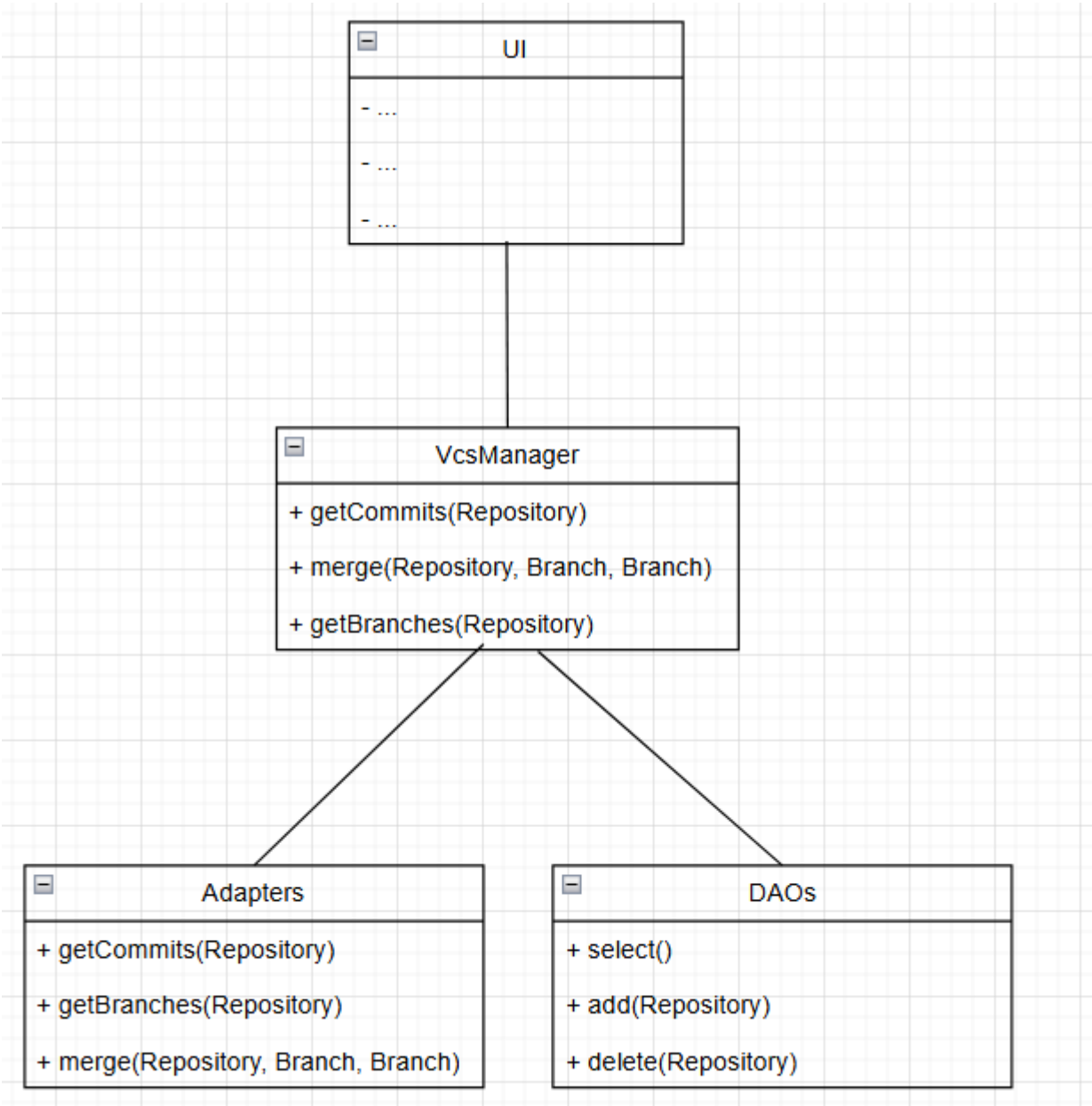


Рис 1 – Діаграма класів модулю з використанням паттерну «Фасад»

```java
public class VcsManager {  6 usages  mavr1iq

    private final DaoFactory daoFactory;  7 usages
    private final AdapterFactory adapterFactory;  2 usages

    public VcsManager() {  1 usage  mavr1iq
        this.daoFactory = new DaoFactory(new DatabaseContext());

        this.adapterFactory = new AdapterFactory(this.daoFactory);
    }

    public void createRepository(String name, String path, VcsType type, int ownerId, String description) {...}


    public void cloneRepository(int sourceRepoId, String targetPath, int ownerId) {...}

    public List<Repository> getAllRepositories() { return daoFactory.getRepositoryDao().getAll(); }

    public Repository getRepository(int id) { return getRepoOrFail(id); }

    public Commit commit(int repoId, int userId, String message) {...}

    public void update(int repoId, String revision) {...}

    public void checkout(int repoId, String branchName) {...}

    public boolean push(int repoId, String remotePath) {...}

    public boolean pull(int repoId, String remotePath) {...}

    public void fetch(int repoId, String remotePath) {...}

    public List<String> listFiles(int repoId, String ref) {...}

    public List<Commit> getLog(int repoId) {...}

    public Iterable<Commit> getCommitHistoryIterator(int startCommitId) {...}

    public List<Branch> getBranches(int repoId) {...}

    public void createBranch(int repoId, String branchName, int creatorId) {...}

    public MergeResults merge(int repoId, String sourceBranchName, String targetBranchName) {...}

    public void createTag(int repoId, String tagName, String message) {...}

    public void applyPatch(int repoId, File patchFile) {...}

    private Repository getRepoOrFail(int repoId) {...}

    public String getRepoGraph(int repoId) {...}

    private VcsAdapter getAdapter(VcsType type) { return adapterFactory.getAdapter(type); }
}
```

Рис 2 – клас VcsManager

Клас VcsManager якраз є вхідною точкою та реалізацією патерну Фасад, так як він має в собі всі потрібні методи, щоб UI не йшла кудись ще, а мало лише цей клас для роботи зі всім застосунком

```
21  public class MainController {  1 usage  mavr1iq
22
23      @FXML private ListView<Repository> repoListView;
24      @FXML private ListView<String> fileListView;
25      @FXML private TextArea commitMessageArea;
26      @FXML private TableView<Commit> historyTable;
27      @FXML private TableColumn<Commit, Integer> colCommitId;
28      @FXML private TableColumn<Commit, String> colMessage;
29      @FXML private TableColumn<Commit, Integer> colAuthor; // Поки ID автора
30      @FXML private Label statusLabel;
31      @FXML private ListView<Branch> branchListView;
32      @FXML private TextField branchNameField;
33      @FXML private TextField tagNameField;
34      @FXML private TextField tagMessageField;
35      @FXML private TextArea graphTextArea;
36
37      private VcsManager vcsManager;  14 usages
38      private Repository currentRepo;  24 usages
39      private final int CURRENT_USER_ID = 1;  3 usages
40
41      @FXML  mavr1iq
42      public void initialize() {
43          // 1. Отримуємо менеджер
```

Рис 3 – декларація змінних у контролері під UI

Видно, що ми використовумо лише VcsManager для доступу до беку

```
public class VcsManager {  6 usages  mavr1iq

    private final DaoFactory daoFactory;  7 usages
    private final AdapterFactory adapterFactory;  2 usages

    public VcsManager() {  1 usage  mavr1iq
        this.daoFactory = new DaoFactory(new DatabaseContext());

        this.adapterFactory = new AdapterFactory(this.daoFactory);
    }
```

Рис 4 – декларація змінних у фасаді

В свою чергу клас з реалізацією Фасаду має всі доступні фабрики для створення адаптерів та ДАО для роботи

Висновок

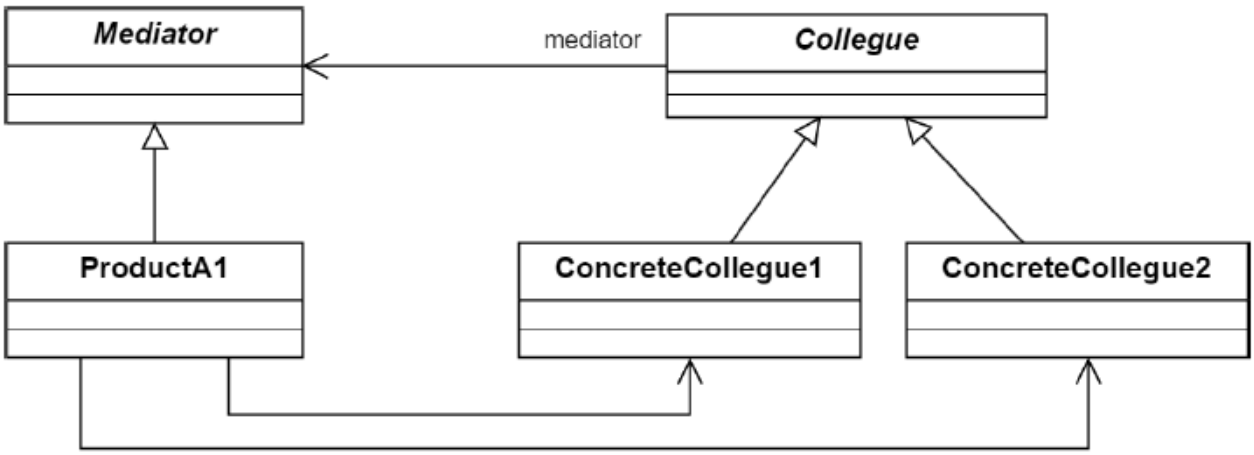
Отже, після виконання даної лабораторної роботи було вивчено паттерн програмування «Фасад», його сильні та слабкі сторони, тонкощі імплементації та use кейси. Для закріплення отриманих знань було розроблено паттерн у існуючому проекті. Було проведено аналіз потреб проекту та визначено потребу в імплементації вказаного патерну для можливості мати 1 точку доступу до застосунку, приховати логіку під адаптерами та ДАО та імплементації додаткової логіки зверху.

Відповіді на питання

1. Яке призначення шаблону «Посередник»?

Призначення шаблону «Посередник» (Mediator) — зменшити зв'язність між класами, винісши їхню взаємодію в окремий об'єкт-посередник. Замість того, щоб компоненти (колеги) спілкувалися один з одним напряму, вони надсилають повідомлення посереднику, який перенаправляє їх іншим компонентам. Це спрощує підтримку коду і дозволяє змінювати логіку взаємодії без зміни самих компонентів.

2. Нарисуйте структуру шаблону «Посередник».



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

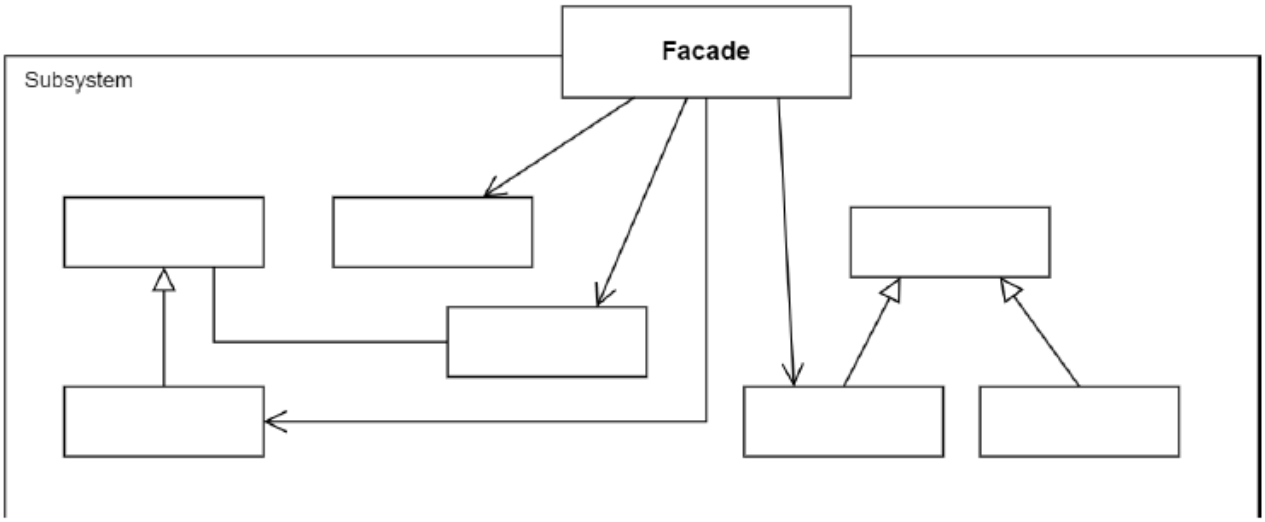
- Mediator (Посередник): Інтерфейс, що визначає методи для комунікації з компонентами (наприклад, notify()).
- ConcreteMediator (Конкретний Посередник): Клас, що реалізує інтерфейс і знає про всі конкретні компоненти. Він координує їхню роботу, отримуючи сповіщення від одних і передаючи команди іншим.
- Component (або Colleague - Колега): Класи компонентів, що виконують бізнес-логіку. Вони мають посилання на Mediator.

Взаємодія: Коли в ComponentA стається подія (наприклад, натиснули кнопку), він викликає метод mediator.notify(this, "event"). ConcreteMediator отримує це повідомлення, вирішує, що робити, і викликає відповідний метод у ComponentB або ComponentC. ComponentA нічого не знає про існування B і C.

4. Яке призначення шаблону «Фасад»?

Призначення шаблону «Фасад» (Facade) — надати простий та уніфікований інтерфейс до складної системи класів, бібліотеки або фреймворку. Фасад приховує складність внутрішньої структури системи від клієнта, дозволяючи виконувати складні операції одним викликом методу.

5. Нарисуйте структуру шаблону «Фасад».



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

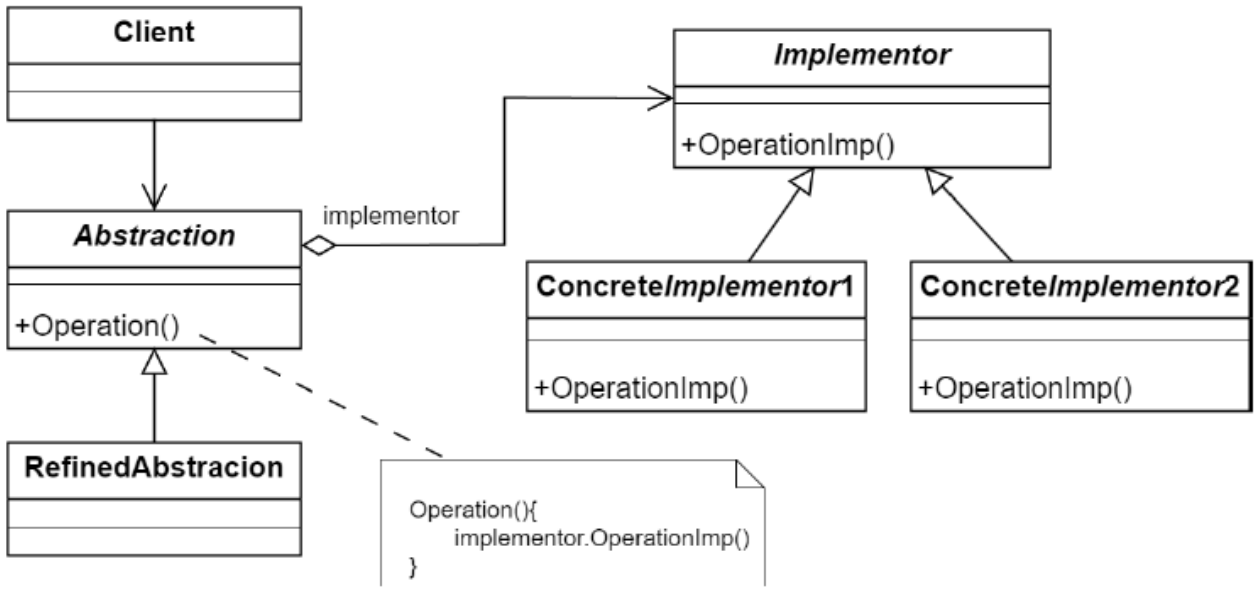
- Facade (Фасад): Клас, який знає, яким класам підсистеми адресувати запит клієнта, і делегує їм роботу.
- Subsystem classes (Класи підсистеми): Набір класів, що реалізують складну функціональність (наприклад, відео-конвертер, кодек, аудіо-мікшер). Вони виконують реальну роботу, але нічого не знають про Фасад.
- Client (Клієнт): Користувач, який взаємодіє тільки з Фасадом.

Взаємодія: Клієнт викликає метод фасаду (наприклад, convertVideo()). Фасад всередині цього методу послідовно викликає методи багатьох класів підсистеми: FileLoader.load(), CodecFactory.extract(), AudioMixer.fix(). Клієнт отримує готовий результат, не знаючи про ці внутрішні виклики.

7. Яке призначення шаблону «Міст»?

Призначення шаблону «Міст» (Bridge) — розділити абстракцію та її реалізацію так, щоб вони могли змінюватися незалежно одна від одної. Це вирішує проблему експоненційного росту кількості класів при використанні спадкування, коли клас має кілька незалежних вимірів змін (наприклад, "Форма" та "Колір").

8. Нарисуйте структуру шаблону «Міст».



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

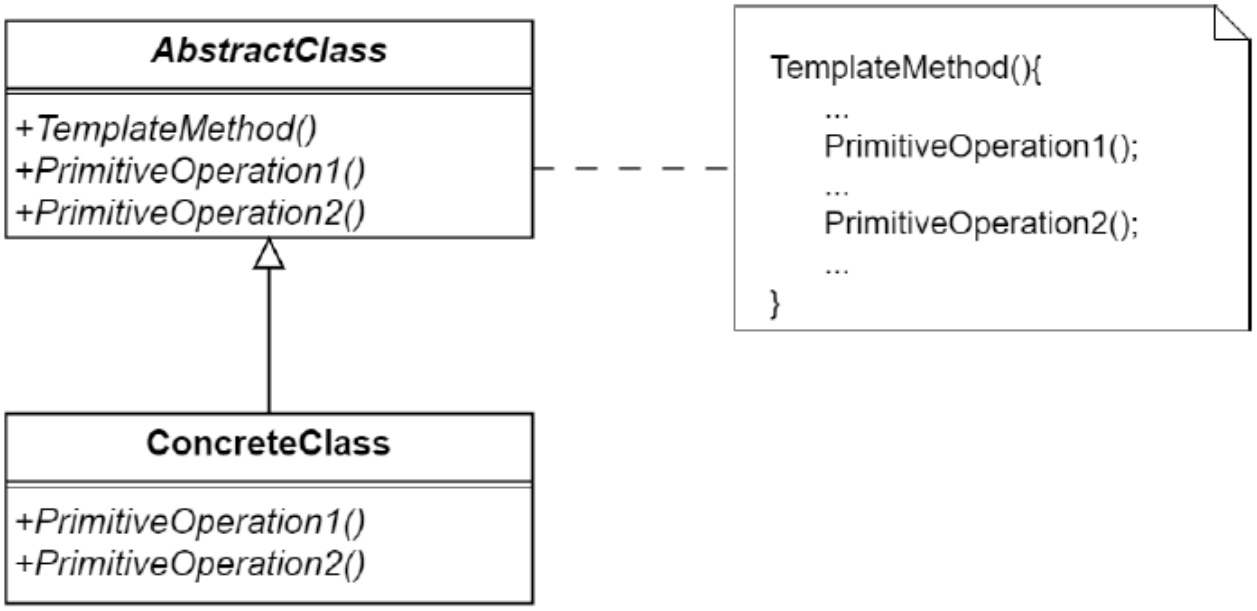
- Abstraction (Абстракція): Визначає інтерфейс "високого рівня" (наприклад, Shape). Зберігає посилання на об'єкт Implementation.
- RefinedAbstraction (Уточнена абстракція): Розширює інтерфейс абстракції (наприклад, Circle, Square).
- Implementation (Реалізація): Інтерфейс для "низькорівневих" операцій (наприклад, Color або DrawingAPI з методами fill(), drawBytes()).
- ConcreteImplementation (Конкретна реалізація): Конкретні класи реалізації (наприклад, RedColor, BlueColor).

Взаємодія: Клієнт працює з Abstraction. Коли він викликає метод (наприклад, circle.draw()), Abstraction делегує роботу об'єкту Implementation (наприклад, color.fill()), який у ньому зберігається. Це дозволяє комбінувати будь-яку фігуру з будь-яким кольором без створення класів RedCircle, BlueCircle тощо.

10. Яке призначення шаблону «Шаблонний метод»?

Призначення шаблону «Шаблонний метод» (Template Method) — визначити скелет алгоритму в базовому класі, але дозволити підкласам перевизначати певні кроки цього алгоритму без зміни його загальної структури.

11. Нарисуйте структуру шаблону «Шаблонний метод».



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

- AbstractClass (Абстрактний клас): Оголошує шаблонний метод (зазвичай final, щоб не можна було змінити структуру), який викликає набір кроків. Деякі кроки мають реалізацію за замовчуванням, інші — абстрактні.
- ConcreteClass (Конкретний клас): Успадковує AbstractClass і реалізує абстрактні кроки, специфічні для цього підкласу.

Взаємодія: Клієнт викликає шаблонний метод у ConcreteClass. Цей метод починає виконуватися за сценарієм, прописаним у AbstractClass. Коли доходить черга до специфічного кроку, викликається його реалізація з ConcreteClass.

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

- Призначення:
  - Фабричний метод: Сфокусований на створенні об'єктів. Він делегує процес створення екземпляра підкласам.
  - Шаблонний метод: Сфокусований на поведінці (алгоритмі). Він визначає послідовність дій, дозволяючи підкласам змінювати окремі кроки.
- Рівень деталізації: Фабричний метод часто є *частиною* великого Шаблонного методу (кроком, який створює об'єкт).
- Результат: Фабричний метод повертає новий об'єкт. Шаблонний метод виконує дію.

14. Яку функціональність додає шаблон «Міст»?

Шаблон «Міст» додає функціональність ортогонального розширення системи. Він дозволяє розвивати ієрархію класів у двох (і більше) незалежних вимірах.

Наприклад, якщо ви розробляєте кросплатформний UI, «Міст» дозволяє додавати підтримку нових операційних систем (Windows, Linux, macOS — *Реалізації*) незалежно від додавання нових віджетів (Кнопка, Вікно, Меню — *Абстракції*). Без «Моста» вам довелося б створювати окремий клас для кожної комбінації (LinuxButton, WindowsButton, LinuxWindow...).