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

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Патерни проектування»

Тема роботи: 7. Редактор зображень

Виконав
студент групи ІА–33
Марченко Вадим Олександрович

Київ 2025

Тема: Патерни проектування

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

Посилання на репозиторій: https://github.com/nwu1015/ImageEditor

Короткі теоретичні відомості

Посередник (Mediator)

Шаблон "Посередник" забезпечує централізоване управління взаємодією між об'єктами, зменшуючи кількість прямих залежностей між ними. Використовується, коли потрібно уникнути складності, що виникає через безпосередні зв'язки між об'єктами в системі. У Java цей шаблон зазвичай реалізується шляхом створення посередника, який координує обмін даними або подіями між об'єктами, що реєструються у ньому. Це спрощує модифікацію і підтримку компонентів, зменшуючи зв'язаність між ними.

Фасад (Facade)

Шаблон "Фасад" надає єдиний інтерфейс до групи взаємопов'язаних класів або підсистем, спрощуючи їх використання. Використовується для зменшення складності системи, приховуючи її деталі реалізації та надаючи зручний API. У Java цей шаблон зазвичай реалізується через клас, який містить методи для основних операцій, що викликають функціонал внутрішніх класів або підсистем. Це дозволяє зменшити залежності між клієнтським кодом і складними підсистемами, спрощуючи розробку і підтримку.

Міст (Bridge)

Шаблон "Міст" розділяє абстракцію та її реалізацію, дозволяючи змінювати їх незалежно одна від одної. Використовується для уникнення жорсткого зв'язку між абстракцією та її конкретними реалізаціями. У Java цей шаблон реалізується через створення інтерфейсу або абстрактного класу для абстракції, а також окремих класів для реалізацій, які

пов'язуються через композицію. Це забезпечує більшу гнучкість у розширенні функціональності системи.

Шаблонний метод (Template Method)

Шаблон "Шаблонний метод" визначає загальну структуру алгоритму, делегуючи реалізацію деяких його кроків підкласам. Використовується для забезпечення гнучкості та уникнення дублювання коду, коли алгоритм має спільні етапи для різних реалізацій. У Java цей шаблон реалізується через абстрактний клас із визначеним методом-шаблоном, який викликає конкретні реалізації абстрактних методів, що задаються підкласами. Це дозволяє стандартизувати алгоритм, зберігаючи варіативність його частин.

Хід роботи

7. **Редактор зображень** (state, prototype, memento, facade, composite, client-server)

Редактор зображень має такі функціональні можливості: відкриття/збереження зображень у найпопулярніших форматах (5 на вибір студента), застосування ефектів, наприклад поворот, розтягування, стиснення, кадрування зображення, можливість створення колажів шляхом «нашарування» зображень.

У цій роботі я використовую патерн "Фасад" (Facade). Цей патерн було обрано як архітектурне рішення для спрощення взаємодії зі складною підсистемою завантаження, обробки та збереження зображень.

У процесі роботи над проєктом стало зрозуміло, що такі операції, як рендеринг фінального колажу або завантаження нового файлу, вимагають цілого набору складних, низькорівневих дій. Наприклад, метод renderAndSaveCollage у CollageService спочатку був змушений сам маніпулювати BufferedImage та Graphics2D, генерувати унікальні імена UUID, працювати з Paths та ImageIO для запису файлу на диск, а потім вручну створювати сутність Image.

Без використання патерну "Фасад", ця складна логіка була б "розмазана" по CollageService, порушуючи його основну відповідальність — керування логікою колажів. Це створило б сильну зв'язаність між CollageService та низькорівневими бібліотеками Java (java.nio.file, javax.imageio). Будь-яка зміна у способі збереження файлів (наприклад, додавання підтримки JPEG) вимагала б модифікації CollageService.

Патерн "Фасад" вирішує цю проблему елегантно. Він надає єдиний, спрощений інтерфейс до цієї складної підсистеми. У моєму проєкті роль "Фасаду" виконує клас ImageService. Він інкапсулює (приховує) всю складність роботи з Files, Paths, ImageIO, BufferedImage та Graphics2D у приватних методах або всередині своїх публічних методів.

Клас CollageService (Клієнт) тепер не знає нічого про те, як файл зберігається на диск або як генерується його ім'я. Він просто викликає один метод фасаду, наприклад, imageService.saveRenderedCollage(canvas, collage, user). ImageService бере на себе всю складну роботу, від запису байтів до створення запису в ImageRepository. Завдяки цьому CollageService залишається чистим і сфокусованим виключно на своїй бізнес-логіці, а вся робота з файлами ізольована в одному місці.

Реалізація паттерну:

```java
/**
 * Приховує всю складність збереження згенерованого BufferedImage.
 * Обробляє іменування, збереження файлу та створення сутності в БД.
 */
@Transactional  1 usage  new *
public Image saveRenderedCollage(BufferedImage canvas, Collage collage, User user) throws IOException {
    String fileExtension = "png";
    String uniqueFilename = "collage-" + UUID.randomUUID() + "." + fileExtension;
    Path destinationFile = this.rootLocation.resolve(uniqueFilename).normalize().toAbsolutePath();

    try {
        ImageIO.write(canvas, fileExtension, destinationFile.toFile());
    } catch (IOException e) {
        throw new RuntimeException("Failed to save rendered collage", e);
    }

    Image finalImage = new Image();
    finalImage.setFileName(uniqueFilename);
    finalImage.setPath(destinationFile.toString());
    finalImage.setFileFormat(fileExtension);
    finalImage.setOwner(user);
    finalImage.setWidth(canvas.getWidth());
    finalImage.setHeight(canvas.getHeight());
    finalImage.setTitle("Результат колажу: " + collage.getName());

    return imageRepository.save(finalImage);
}
```

Рисунок 1. - Програмний код методу saveRenderesCollage() у класі ImageService

```java
@Transactional  1 usage  nwu1015 *
public Image renderAndSaveCollage(Long collageId, User user) throws IOException {
    Collage collage = findCollageById(collageId);

    BufferedImage canvas = new BufferedImage(
            collage.getCanvasWidth(),
            collage.getCanvasHeight(),
            BufferedImage.TYPE_INT_ARGB
    );
    Graphics2D g2d = canvas.createGraphics();

    for (ImageLayer layer : collage.getLayers()) {
        // Фасадний метод
        BufferedImage transformedLayerImage =
                imageService.applyTransformationsToLayer(layer.getId());

        g2d.drawImage(transformedLayerImage, layer.getPositionX(),
                layer.getPositionY(),  observer: null);
    }
    g2d.dispose();

    // Фасадний метод
    return imageService.saveRenderedCollage(canvas, collage, user);
}
```

Рисунок 2. - Програмний код методу renderAndSaveCollage() CollageService

Попередня реалізація методу saveRenderesCollage() на момент виконання лабораторної роботи №6:

```
163     @Transactional  1 usage   nwu1015
164     public Image renderAndSaveCollage(Long collageId, User user) throws IOException {
165         Collage collage = findCollageById(collageId);
166
167         BufferedImage canvas = new BufferedImage(
168                 collage.getCanvasWidth(),
169                 collage.getCanvasHeight(),
170                 BufferedImage.TYPE_INT_ARGB // Тип, що підтримує прозорість
171         );
172         Graphics2D g2d = canvas.createGraphics();
173
174         for (ImageLayer layer : collage.getLayers()) {
175             BufferedImage transformedLayerImage =
176                     imageService.applyTransformationsToLayer(layer.getId());
177
178             g2d.drawImage(transformedLayerImage, layer.getPositionX(),
179                     layer.getPositionY(),  observer: null);
180         }
181         g2d.dispose();
182
183         String finalFileName = "collage-" + UUID.randomUUID() + ".png";
184         Path finalPath = Paths.get( first: "uploads").resolve(finalFileName);
185         ImageIO.write(canvas,  formatName: "png", finalPath.toFile());
186
187         Image finalImage = new Image();
188         finalImage.setFileName(finalFileName);
189         finalImage.setPath(finalPath.toString());
190         finalImage.setFileFormat("png");
191         finalImage.setWidth(canvas.getWidth());
192         finalImage.setHeight(canvas.getHeight());
193         finalImage.setOwner(user);
194         finalImage.setTitle("Результат колажу: " + collage.getName());
195
196         return imageService.saveFinalImage(finalImage);
197     }
```

Рисунок 3. - Програмний код методу saveRenderesCollage() у класі ImageService до реалізації паттерну Facade.

Діаграма класів

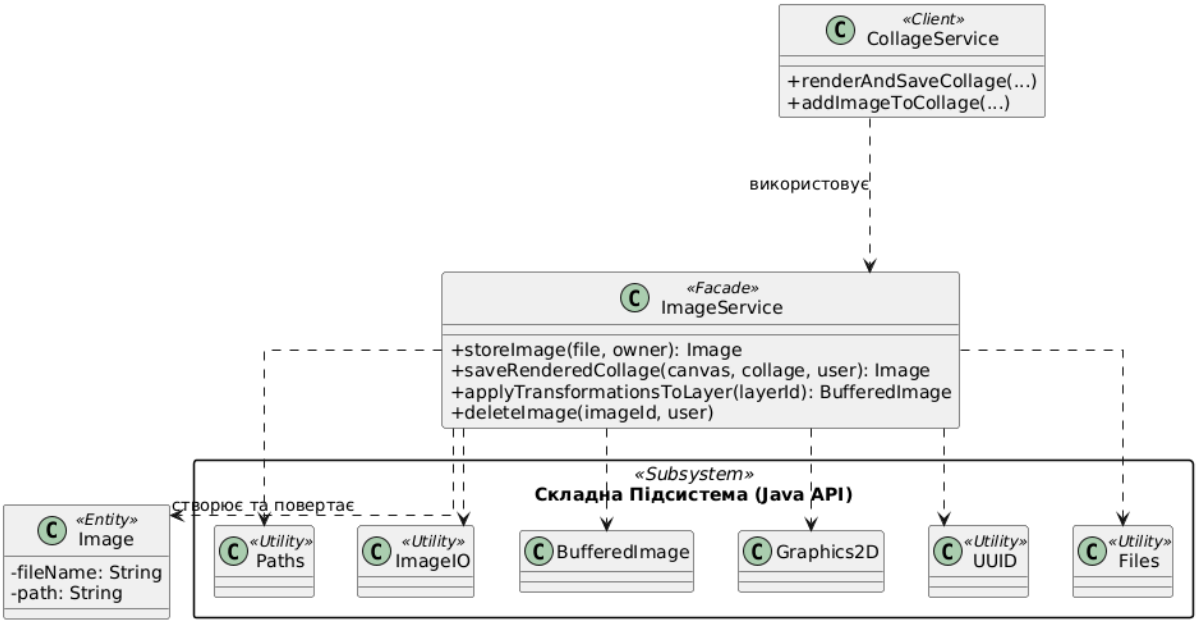


Рисунок 4. - Діаграма класів реалізації паттерну

Висновок: Виконуючи цю лабораторну роботу, я ознайомився з такими патернами, як «Mediator», «Facade», «Bridge», «Template method».

Особливу увагу я приділив реалізації шаблону "Фасад" (Facade), детально описавши його основну логіку та функціональність у контексті мого проєкту. Цей патерн було обрано як архітектурне рішення для спрощення взаємодії зі складною підсистемою завантаження, обробки та збереження файлів зображень.

Під час реалізації шаблону "Фасад" я зрозумів, наскільки цей патерн елегантно вирішує проблему складності, інкапсулюючи цілий набір низькорівневих операцій за єдиним, чистим інтерфейсом. Це дозволяє "Клієнту" виконувати складні дії, наприклад, renderAndSaveCollage, не маючи жодного уявлення про внутрішню реалізацію роботи з java.nio.file.Files, ImageIO чи BufferedImage. Це робить код значно чистішим та забезпечує чітке розділення відповідальності: "Фасад"

(ImageService) бере на себе всю складну роботу з файлами, а сервіс фокусується виключно на своїй бізнес-логіці (керуванні колажем).

Завдяки цьому досвіду я краще зрозумів переваги патерну "Фасад" у контексті роботи зі складними, багатокомпонентними підсистемами. Я усвідомив його ключову роль у зниженні зв'язаності між різними частинами програми та в підтримці гнучкої, слабко зв'язаної архітектури програмної системи.

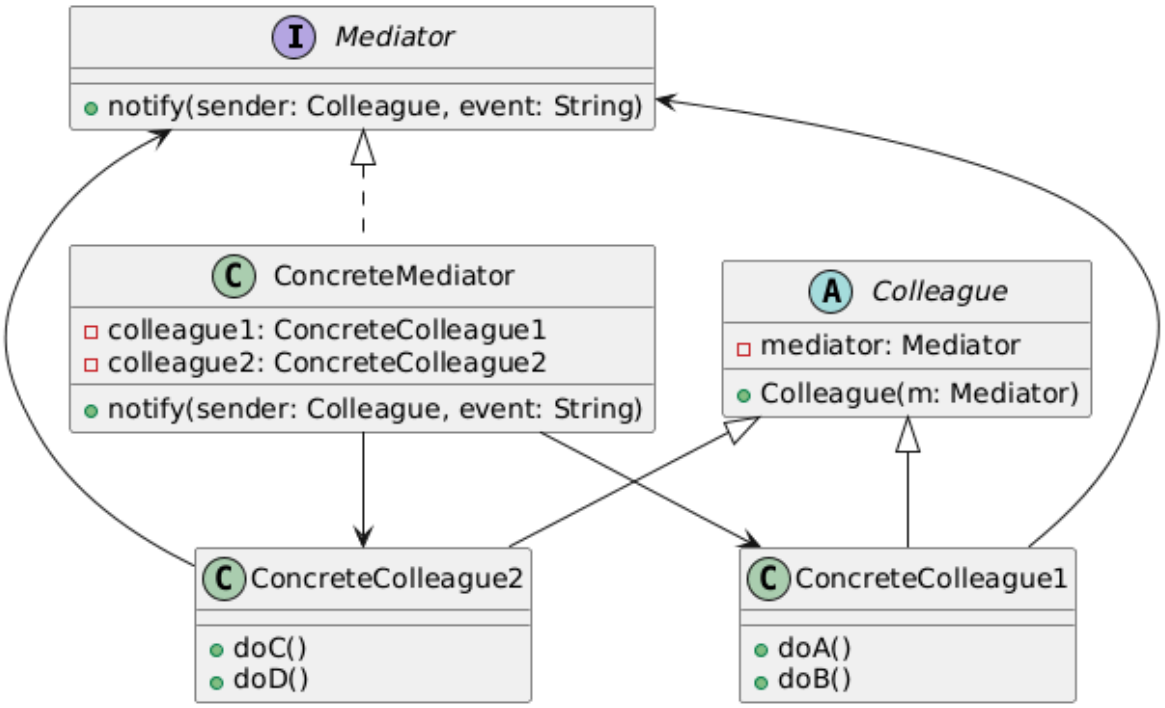
Контрольні запитання

1. Яке призначення шаблону «Посередник»?

Шаблон «Посередник» (Mediator) призначений для спрощення взаємодії між об’єктами у системі. Замість того, щоб об’єкти напряму обмінювалися повідомленнями та створювали заплутані зв’язки, усі вони спілкуються через єдиний об’єкт-посередник.

Посередник контролює обмін даними між компонентами, координує їхню роботу та зменшує кількість залежностей у програмі. Завдяки цьому система стає більш гнучкою та легкою в супроводі, оскільки зміни в одному компоненті не потребують змін в інших.

2. Нарисуйте структуру шаблону «Посередник».



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

До шаблону «Посередник» (Mediator) входять такі основні класи:

- Mediator (Посередник) — це інтерфейс або абстрактний клас, який визначає методи для обміну інформацією між об’єктами.

- ConcreteMediator (Конкретний посередник) — реалізує інтерфейс посередника та координує взаємодію між конкретними об’єктами (колегами). Він знає, які об’єкти беруть участь у взаємодії, і спрямовує повідомлення між ними.

- Colleague (Колега) — це базовий клас або інтерфейс для об’єктів, які взаємодіють через посередника.

- ConcreteColleague (Конкретний колега) — об’єкти, які виконують певні дії, але не спілкуються напряму між собою. Вони надсилають повідомлення посереднику, а той уже вирішує, кому їх передати.

Коли один із колег хоче повідомити інший об’єкт, він не викликає його метод безпосередньо, а звертається до посередника. Посередник приймає повідомлення та вирішує, який інший колега повинен отримати цю

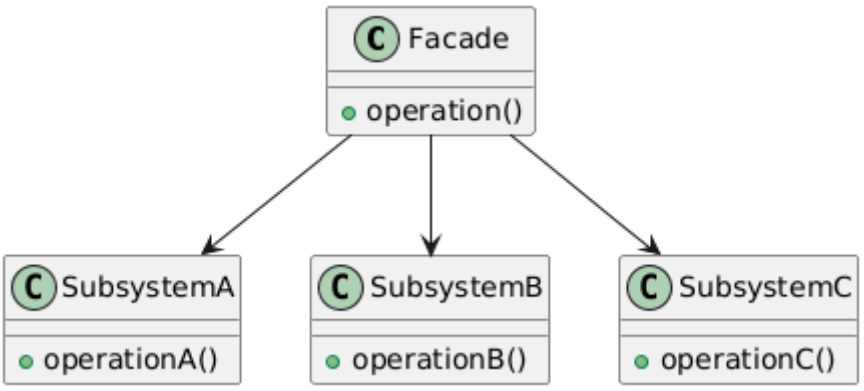
інформацію. Таким чином, усі зв’язки проходять через посередника, що робить систему більш керованою й менш залежною.

4. Яке призначення шаблону «Фасад»?

Шаблон «Фасад» (Facade) призначений для спрощення роботи зі складною системою. Він надає єдиний узагальнений інтерфейс для взаємодії з групою класів або підсистем, приховуючи їхню внутрішню складність.

Завдяки фасаду клієнт може виконувати складні операції за допомогою одного простого виклику методу, не знаючи, як саме усе реалізовано всередині. Це робить код зрозумілішим, зручнішим у використанні та легшим у супроводі.

5. Нарисуйте структуру шаблону «Фасад».



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

До шаблону «Фасад» (Facade) входять такі основні класи:

- Facade (Фасад) — головний клас, який надає спрощений інтерфейс для клієнта. Він містить посилання на класи підсистеми та викликає їхні методи у потрібному порядку.

- Subsystem classes (Класи підсистеми) — це реальні компоненти системи, які виконують основну роботу. Вони не знають про фасад і можуть використовуватися незалежно від нього.

- Client (Клієнт) — це код, який використовує фасад, щоб отримати доступ до функціональності підсистеми через простий інтерфейс.

Клієнт звертається лише до фасаду, не взаємодіючи безпосередньо з класами підсистеми. Фасад приймає запит, викликає необхідні методи підсистеми та повертає результат. Таким чином, фасад приховує складність внутрішньої структури системи й спрощує роботу з нею.

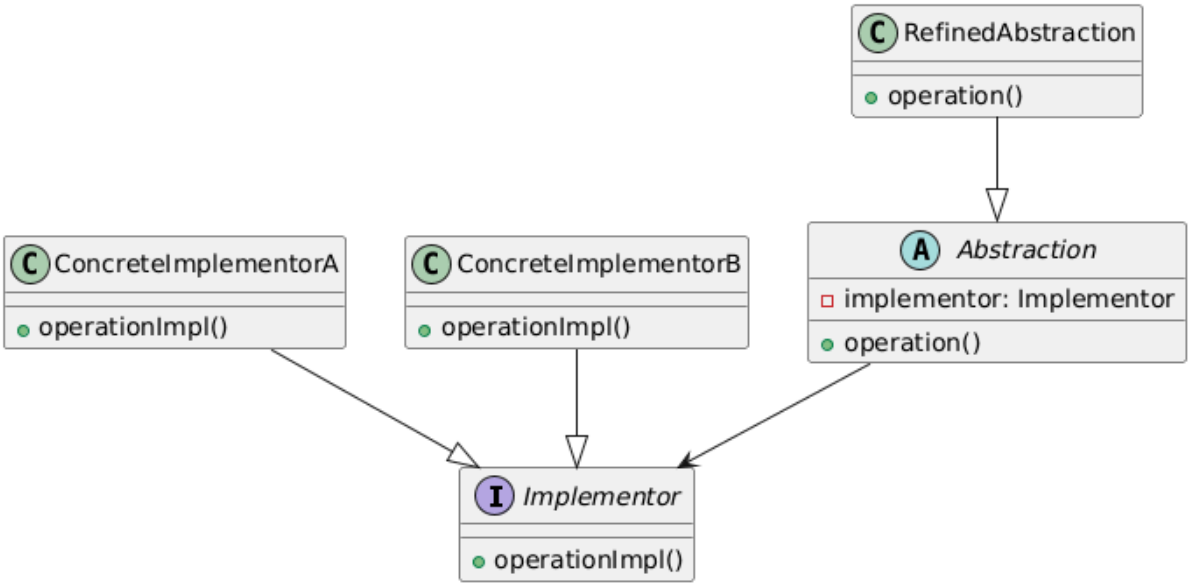
7. Яке призначення шаблону «Міст»?

Шаблон «Міст» (Bridge) призначений для відокремлення абстракції від її реалізації, щоб вони могли розвиватися незалежно одна від одної.

Замість того, щоб жорстко пов’язувати абстрактний клас із конкретною реалізацією, «Міст» створює між ними окремий зв’язок через інтерфейс. Це дозволяє легко змінювати або розширювати як абстракцію, так і реалізацію, не впливаючи одна на одну.

У результаті код стає більш гнучким і масштабованим, особливо коли потрібно підтримувати різні варіанти реалізацій або платформ.

8. Нарисуйте структуру шаблону «Міст».



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

До шаблону «Міст» (Bridge) входять такі основні класи:

- Abstraction (Абстракція) — визначає базовий інтерфейс для користувачів і містить посилання на об’єкт реалізації. Вона делегує частину роботи об’єкту реалізації, замість того щоб виконувати її самостійно.

- RefinedAbstraction (Уточнена абстракція) — розширює функціональність базової абстракції, але все одно використовує реалізацію через міст.

- Implementor (Реалізатор) — це інтерфейс, який визначає методи, що мають бути реалізовані в конкретних реалізаторах.

- ConcreteImplementor (Конкретний реалізатор) — надає специфічну реалізацію методів, оголошених в інтерфейсі Implementor.

Абстракція зберігає посилання на об’єкт типу Implementor і викликає його методи для виконання конкретних дій. Клієнт працює лише з абстракцією, не знаючи про деталі реалізації. Таким чином, абстракція та реалізація можуть змінюватися незалежно, що забезпечує гнучкість і розширюваність системи.

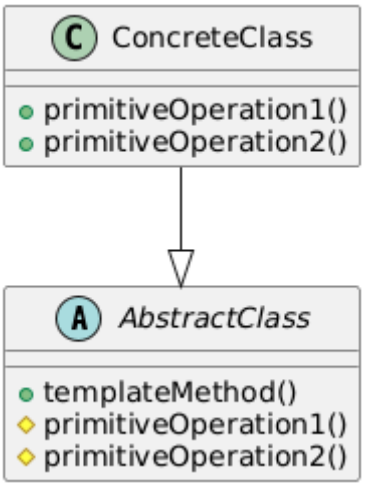
10.Яке призначення шаблону «Шаблоний метод»?

Шаблон «Шаблонний метод» (Template Method) призначений для визначення загальної послідовності дій алгоритму в базовому класі, залишаючи реалізацію окремих кроків підкласам.

Це дозволяє створити спільний “каркас” алгоритму, який не потрібно переписувати в кожному класі, а лише перевизначати окремі частини, що відрізняються.

Завдяки цьому шаблон забезпечує повторне використання коду, підтримує єдину структуру процесу та дозволяє легко змінювати поведінку алгоритму через наслідування.

11.Нарисуйте структуру шаблону «Шаблонний метод».



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

До шаблону «Шаблонний метод» (Template Method) входять такі основні класи:

- AbstractClass (Абстрактний клас) — визначає шаблонний метод, який описує загальну послідовність виконання алгоритму. Деякі кроки

алгоритму реалізовані тут, а деякі оголошені як абстрактні методи, які мають бути реалізовані підкласами.

- ConcreteClass (Конкретний клас) — наслідує абстрактний клас і реалізує абстрактні методи, тобто визначає конкретну поведінку окремих кроків алгоритму.

Клієнт викликає шаблонний метод абстрактного класу. Шаблонний метод послідовно викликає внутрішні методи: частина з них реалізована в абстрактному класі, частина — у підкласах. Підклас відповідає лише за деталі окремих кроків, не змінюючи загальну структуру алгоритму.

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

Шаблон «Шаблонний метод» використовується для визначення загальної структури алгоритму в базовому класі, залишаючи підкласам реалізацію окремих кроків. Він дозволяє повторно використовувати код і змінювати лише деталі алгоритму без зміни його загальної структури.

Шаблон «Фабричний метод» застосовується для створення об’єктів без жорсткого зв’язку з конкретними класами. Підкласи вирішують, який конкретний об’єкт створювати, а клієнт користується лише базовим інтерфейсом або класом.

Основна відмінність полягає в тому, що «Шаблонний метод» визначає послідовність дій, а «Фабричний метод» — спосіб створення об’єктів.

14.Яку функціональність додає шаблон «Міст»?

Шаблон «Міст» (Bridge) додає функціональність для незалежного розділення абстракції та її реалізації. Він дозволяє змінювати або розширювати абстракцію і реалізацію окремо, не впливаючи одна на одну. Завдяки цьому система стає гнучкішою, масштабованішою та легшою для

підтримки, особливо коли потрібно підтримувати кілька варіантів реалізацій або платформ.