



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
**Технологія розроблення програмного
забезпечення**

“Використання шаблону Command, тема: Download manager.”
Варіант 26

Виконала
студентка групи ІА-13
Танасієнко Анастасія Вячеславівна

Перевірів:

Київ 2023р.

Мета: Дослідити шаблони ШАБЛОНИ «ADAPTER», «BUILDER», «COMMAND», «CHAIN OF RESPONSIBILITY», «PROTOTYPE»

Завдання.

1. Ознайомитися з короткими теоретичними відомостями.
2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3. Застосування одного з розглянутих шаблонів при реалізації програми.

1. ШАБЛОН «ADAPTER»

Шаблон "ADAPTER" є одним з базових шаблонів проектування в програмній інженерії. Його основна мета - дозволити існуючому інтерфейсу працювати з новим, несумісним інтерфейсом без модифікації вихідного коду.

Основні характеристики та особливості шаблону "ADAPTER":

- Цільовий інтерфейс (Target Interface): Визначає той інтерфейс, який відомий та очікується від клієнта.
- Адаптер (Adapter): Клас, який реалізує цільовий інтерфейс та містить посилання на адаптований об'єкт.
- Адаптований об'єкт (Adaptee): Об'єкт, чий інтерфейс несумісний з цільовим, і який потрібно адаптувати.
- Адаптація (Adaptation): Процес створення адаптера, який вирішує проблеми несумісності між інтерфейсами.
- Мост між інтерфейсами (Bridge between Interfaces): Адаптер служить мостом між цільовим інтерфейсом та адаптованим об'єктом.
- Конвертація запитів (Request Conversion): Адаптер перетворює запити від цільового інтерфейсу на запити, зрозумілі адаптованому об'єкту.

Зберігання адаптованого об'єкта (Holding the Adaptee): Адаптер має посилання на адаптований об'єкт для того, щоб можна було викликати його методи.

Шаблон "ADAPTER" дозволяє використовувати існуючий код та класи, які мають різний інтерфейс, в новому контексті без потреби їх змінювати. Він особливо корисний при роботі зі сторонніми бібліотеками, інтерфейси яких потрібно адаптувати під потреби програми.

2. ШАБЛОН «BUILDER»

Шаблон "BUILDER" є одним із базових шаблонів проектування, який використовується для конструювання складних об'єктів крок за кроком. Він дозволяє відокремити процес конструювання об'єкта від його представлення.

Основні характеристики та особливості шаблону "BUILDER":

- Директор (Director): Відповідає за взаємодію з будівельником для конструювання об'єкта. Він визначає послідовність кроків для створення об'єкта.
- Будівельник (Builder): Інтерфейс або абстрактний клас, що оголошує методи для конструювання окремих частин об'єкта.
- Конкретні будівельники (Concrete Builders): Реалізують інтерфейс будівельника для конструювання конкретних типів об'єктів.
- Продукт (Product): Результат конструювання, який представляє собою складний об'єкт.
- Додаткові деталі конструювання (Construction Details): Будівельник може мати методи для додавання додаткових деталей до об'єкта.
- Можливість отримати результат (Getting the Result): Після завершення конструювання, будівельник повертає готовий об'єкт.

Шаблон "BUILDER" дозволяє створювати різні варіанти одного об'єкта шляхом зміни будівельників. Він особливо корисний, коли існує багато можливих конфігурацій об'єкта, і важко чи незручно конструювати його без використання спеціального інтерфейсу.

3. ШАБЛОН «COMMAND»

Шаблон "COMMAND" є одним із базових шаблонів проектування, який дозволяє ізольовано обробляти запити чи команди як об'єкти. Він дозволяє реалізувати функціональність виклику методу в об'єкті без знання конкретної реалізації цього методу.

Основні характеристики та особливості шаблону "COMMAND":

- Команда (Command): Цей клас містить у собі інформацію про конкретну команду та об'єкт, на якому вона повинна виконатися. Він містить методи execute(), який викликається для виконання команди.
- Отримувач (Receiver): Це об'єкт, на якому виконується команда. Він містить фактичний код, який виконує дії, які вимагає команда.
- Викликач (Invoker): Цей клас викликає команди для виконання. Він має посилання на конкретну команду та може викликати її метод execute().
- Сполучник (Client): Цей клас створює об'єкти команд та прив'язує їх до відповідних отримувачів.
- Відміна операцій (Undo Operations): Шаблон "COMMAND" дозволяє легко реалізувати можливість відміни виконаних операцій.
- Композиція команд (Composite Commands): Команди можуть бути скомпоновані у складніше дерево, що дозволяє виконувати групу команд як єдину одиницю.

Шаблон "COMMAND" дозволяє відокремити ініціатора запиту від конкретної реалізації виконання команди. Це робить систему більш гнучкою та дозволяє легко розширювати набір команд. Він особливо корисний в ситуаціях, коли потрібно реалізувати відміну операцій або створити комплексні команди з різних простіших.

4. ШАБЛОН «CHAIN OF RESPONSIBILITY»

Шаблон "CHAIN OF RESPONSIBILITY" є одним з поведінкових шаблонів проектування, який дозволяє передавати запити через ланцюг обробників. Якщо один обробник не може обробити запит, він передає його на обробку наступному обробнику в ланцюжку.

Основні характеристики та особливості шаблону "CHAIN OF RESPONSIBILITY":

- Обробник (Handler): Абстрактний клас або інтерфейс, що описує метод обробки запиту та має посилання на наступний обробник в ланцюжку.
- Конкретні обробники (Concrete Handlers): Конкретні реалізації обробників, які спробують обробити запит. Якщо вони не можуть обробити його, вони передають його наступному обробнику в ланцюжку.
- Ланцюг обробників (Chain of Handlers): Обробники формують ланцюг, в якому кожен обробник може намагатися обробити запит, або передати його далі.
- Передача запиту (Request Propagation): Якщо обробник не може обробити запит, він передає його наступному обробнику в ланцюжку.
- Зупинка ланцюжка (Stopping the Chain): Обробник може вирішити не передавати запит далі, зупиняючи ланцюг.
- Динамічне додавання та видалення обробників (Dynamic Addition and Removal of Handlers): Обробники можуть динамічно додаватися та видалятися з ланцюжка під час виконання програми.

Шаблон "CHAIN OF RESPONSIBILITY" дозволяє побудувати логічний ланцюг для обробки запитів, при цьому кожен обробник вирішує, чи може він обробити запит, чи повинен передати його далі. Це робить систему більш гнучкою та дозволяє легко додавати чи змінювати обробників.

5. ШАБЛОН «PROTOTYPE»

Шаблон "PROTOTYPE" є одним з паттернів проектування, який дозволяє створювати нові об'єкти на основі вже існуючих прототипів. Він використовує копіювання об'єктів для генерації нових екземплярів.

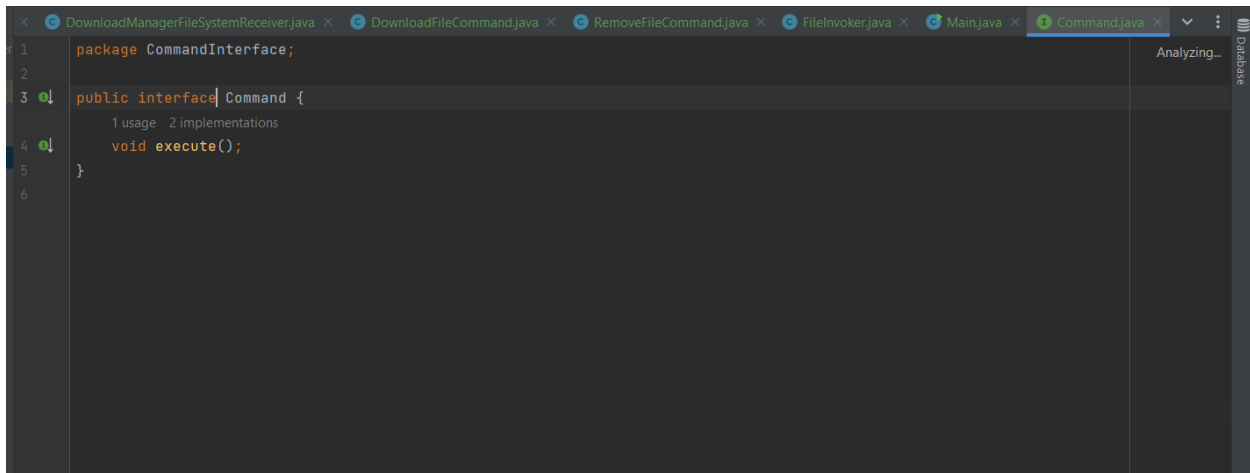
Основні характеристики та особливості шаблону "PROTOTYPE":

- Прототип (Prototype): Це абстрактний клас або інтерфейс, який визначає метод для копіювання самого себе.
- Конкретні прототипи (Concrete Prototypes): Конкретні реалізації прототипів, які реалізують метод копіювання.
- Клієнт (Client): Використовує прототип для створення нових об'єктів шляхом копіювання існуючого.
- Копіювання (Cloning): Прототип дозволяє копіювати себе, створюючи новий екземпляр.
- Глибоке та поверхневе копіювання (Deep and Shallow Copy): Прототип може здійснювати глибоке або поверхневе копіювання, залежно від потреб програми.
- Ефективне створення об'єктів (Efficient Object Creation): Шаблон "PROTOTYPE" дозволяє ефективно створювати нові об'єкти, особливо коли є необхідність у багатьох подібних екземплярах.
- Динамічна зміна класу об'єкта (Dynamic Class Change): Прототип дозволяє динамічно змінювати клас об'єкта, створюючи новий екземпляр на основі прототипу.

Шаблон "PROTOTYPE" дозволяє зекономити час та ресурси, оскільки нові об'єкти можна створювати на основі вже існуючих. Він особливо корисний, коли об'єкти мають складну ініціалізацію або коли потрібно генерувати велику кількість подібних екземплярів.

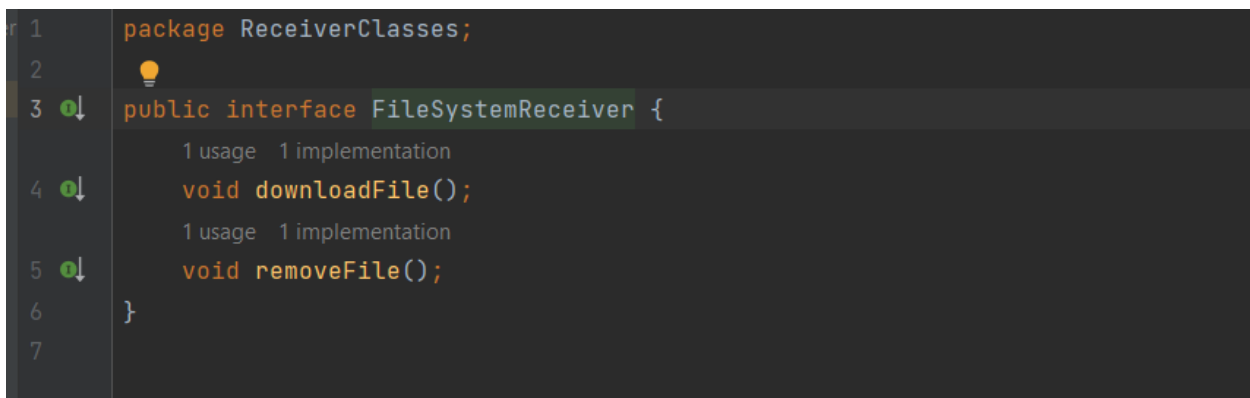
Результат роботи

-Наданий код - це інтерфейс Command в мові програмування Java. Він визначає лише один метод execute(), який має бути реалізований класами, що реалізують цей інтерфейс.



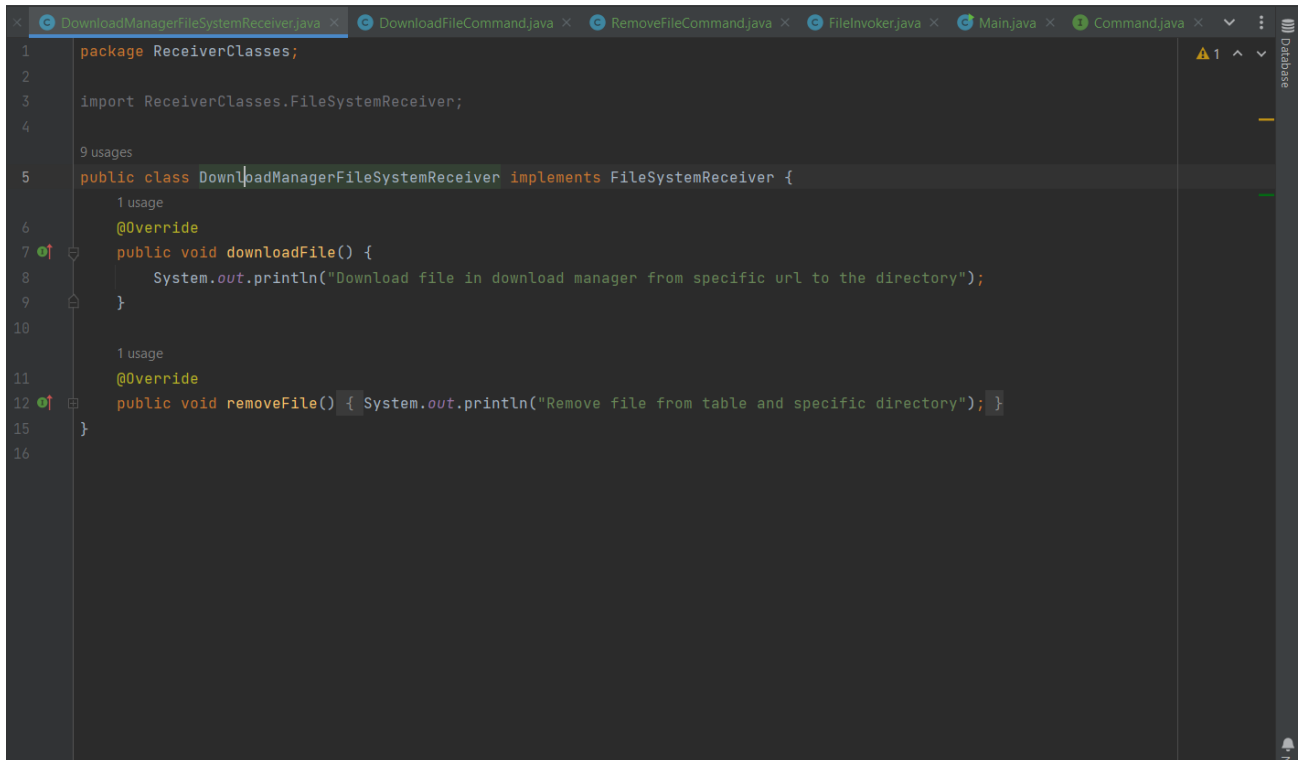
```
1 package CommandInterface;
2
3 public interface Command {
4     void execute();
5 }
6
```

-Даний код визначає інтерфейс FileSystemReceiver. У цьому інтерфейсі оголошені два методи: downloadFile() для завантаження файлу та removeFile() для видалення файлу. Класи, які реалізують цей інтерфейс, повинні надати конкретну реалізацію цих методів.



```
1 package ReceiverClasses;
2
3 public interface FileSystemReceiver {
4     void downloadFile();
5     void removeFile();
6 }
7
```

-Даний код визначає клас DownloadManagerFileSystemReceiver, який реалізує інтерфейс FileSystemReceiver. Цей клас містить методи downloadFile() і removeFile(), які виконують операції з завантаження та видалення файлів відповідно. Метод downloadFile() виводить повідомлення про завантаження файлу за конкретним URL до вказаного каталогу, а метод removeFile() виводить повідомлення про видалення файлу з таблиці та конкретного каталогу.



```
1 package ReceiverClasses;
2
3 import ReceiverClasses.FileSystemReceiver;
4
5 public class DownloadManagerFileSystemReceiver implements FileSystemReceiver {
6     @Override
7     public void downloadFile() {
8         System.out.println("Download file in download manager from specific url to the directory");
9     }
10
11     @Override
12     public void removeFile() { System.out.println("Remove file from table and specific directory"); }
13 }
14
15
16
```

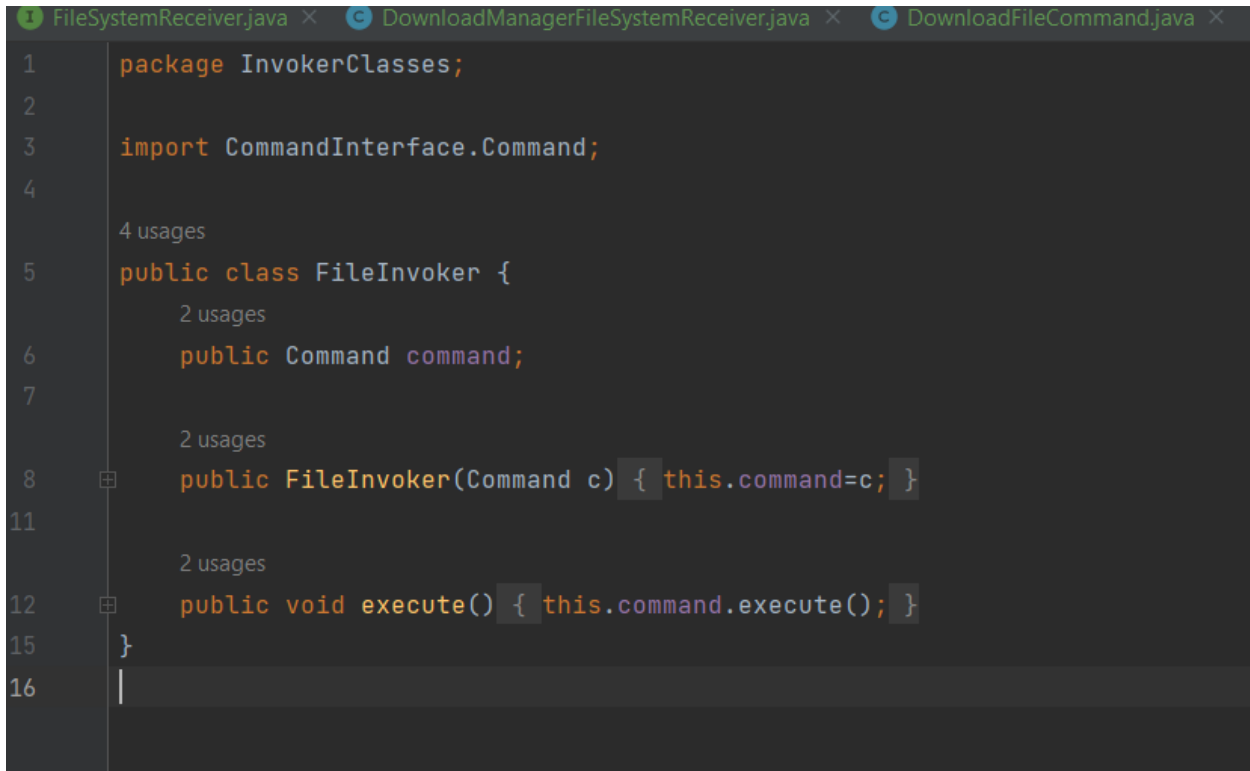

-Наданий код - це клас DownloadFileCommand, який реалізує інтерфейс Command. Клас призначений для виконання команди з завантаження файлу. Він містить посилання на об'єкт DownloadManagerFileSystemReceiver та реалізовує метод execute(), який викликає відповідний метод downloadFile() у об'єкта DownloadManagerFileSystemReceiver для виконання операції завантаження файлу.

```
1 package Commands;
2
3 import CommandInterface.Command;
4 import ReceiverClasses.DownloadManagerFileSystemReceiver;
5
6 public class DownloadFileCommand implements Command {
7     private DownloadManagerFileSystemReceiver downloadManager;
8
9     public DownloadFileCommand(DownloadManagerFileSystemReceiver dm) { this.downloadManager=dm; }
10
11     @Override
12     public void execute() {
13         //download command is forwarding request to downloadFile method
14         this.downloadManager.downloadFile();
15     }
16 }
17
18 }
```

-Наданий код - це клас RemoveFileCommand, який реалізує інтерфейс Command. Цей клас призначений для виконання команди з видалення файлу. Він містить посилання на об'єкт DownloadManagerFileSystemReceiver та реалізовує метод execute(), який викликає відповідний метод removeFile() у об'єкта DownloadManagerFileSystemReceiver для виконання операції видалення файлу.

```
1 package Commands;
2
3 import CommandInterface.Command;
4 import ReceiverClasses.DownloadManagerFileSystemReceiver;
5
6 3 usages
7 public class RemoveFileCommand implements Command {
8     2 usages
9     private DownloadManagerFileSystemReceiver downloadManager;
10
11     1 usage
12     public RemoveFileCommand(DownloadManagerFileSystemReceiver dm) { this.downloadManager=dm; }
13
14     1 usage
15     @Override
16     public void execute() {
17         //remove command is forwarding request to removeFile method
18         this.downloadManager.removeFile();
19     }
20 }
```

-Даний код представляє клас `FileInvoker`, який діє як інвокер (викликач) для команди. Він має поле `command` типу `Command`, яке зберігає команду для виклику. Конструктор класу приймає команду і зберігає її у поле. Метод `execute()` викликає відповідний метод `execute()` для виконання збереженої команди. Цей клас може використовуватися для виклику різних команд у системі.



```
1 package InvokerClasses;
2
3 import CommandInterface.Command;
4
5 4 usages
6 public class FileInvoker {
7     2 usages
8     public Command command;
9
10     2 usages
11     public FileInvoker(Command c) { this.command=c; }
12
13     2 usages
14     public void execute() { this.command.execute(); }
15 }
16
```

-Цей код використовує паттерн "Команда" для виконання дій з файлами. Завантажується файл та видаляється через відповідні команди, які асоціюються з викликачем дій.

```
FileSystemReceiver.java × DownloadManagerFileSystemReceiver.java × DownloadFileCommand.java × RemoveFileCommand.java
1  import Commands.DownloadFileCommand;
2      import Commands.RemoveFileCommand;
3      import InvokerClasses.FileInvoker;
4  import ReceiverClasses.DownloadManagerFileSystemReceiver;
5
6  public class Main {
7      public static void main(String[] args) {
8          DownloadManagerFileSystemReceiver fs = new DownloadManagerFileSystemReceiver();
9
10         //creating command and associating with receiver
11         DownloadFileCommand downloadFileCommand = new DownloadFileCommand(fs);
12
13         //Creating invoker and associating with Command
14         FileInvoker file = new FileInvoker(downloadFileCommand);
15
16         //perform action on invoker object
17         file.execute();
18
19
20         RemoveFileCommand removeFileCommand = new RemoveFileCommand(fs);
21         file = new FileInvoker(removeFileCommand);
22         file.execute();
23     }
24 }
25 |
```

Висновок:

Лабораторна робота була спрямована на дослідження та реалізацію п'яти важливих шаблонів проектування: "ADAPTER", "BUILDER", "COMMAND", "CHAIN OF RESPONSIBILITY" та "PROTOTYPE". Кожен із них має свою унікальну функціональність та може бути використаний для вирішення різноманітних завдань у програмній розробці.

- Шаблон проектування "ADAPTER": Цей шаблон дозволяє об'єктам з різними інтерфейсами працювати разом. Він надає проміжний інтерфейс, який перетворює один інтерфейс у інший, щоб вони могли взаємодіяти без проблем.
- Шаблон проектування "BUILDER": Цей шаблон дозволяє побудувати складні об'єкти крок за кроком. Він використовує декілька підприємств для конструювання об'єкта, щоб дати можливість створювати різні варіанти одного об'єкта.
- Шаблон проектування "COMMAND": Цей шаблон дозволяє ізольовано обробляти запити або команди як об'єкти. Він дозволяє реалізувати функціональність виклику методу в об'єкті без знання конкретної реалізації цього методу.
- Шаблон проектування "CHAIN OF RESPONSIBILITY": Цей шаблон дозволяє передавати запити через ланцюжок обробників. Якщо один обробник не може обробити запит, він передає його на обробку наступному обробнику в ланцюжку.
- Шаблон проектування "PROTOTYPE": Цей шаблон дозволяє створювати нові об'єкти на основі вже існуючих прототипів. Він використовує копіювання об'єктів для генерації нових екземплярів.

Під час виконання лабораторної роботи ми навчилися реалізовувати та досліджувати кожен з цих шаблонів. Кожен з них має свої конкретні випадки застосування та може бути корисним у різних сценаріях розробки програмного забезпечення. Робота над цими шаблонами надала нам глибше розуміння принципів об'єктно-орієнтованого програмування та паттернів проектування.