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

ШАБЛОНИ «ADAPTER», «BUILDER», «COMMAND», «CHAINOFRESPONSIBILITY», «PROTOTYPE»

Виконав
студент групи ІА-31
Даценко Влад

Перевірив:
Мягкий Михайло
Юрійович

Київ 2025р.

Зміст

Тема: 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 їх взаємодій для досягнення конкретних функціональних можливостей.

3. Застосування одного з розглянутих шаблонів при реалізації програми.

Короткі теоретичні відомості

Adapter (Адаптер) – це структурний шаблон, який дозволяє об'єктам з різними інтерфейсами працювати разом. Адаптер перетворює інтерфейс одного класу на інтерфейс, який очікує інший клас. Це дозволяє інтегрувати різні компоненти в систему, не змінюючи їх внутрішню реалізацію. Адаптери часто використовуються для сумісності з існуючими бібліотеками чи старими системами.

Builder (Будівельник) – це створювальний шаблон, який розділяє процес створення складних об'єктів на кілька етапів. Це дозволяє створювати різні варіанти об'єкта за допомогою однакових кроків, але з різними параметрами. Шаблон Builder корисний для побудови об'єктів, які мають багато складових, і дозволяє керувати їх створенням поетапно.

Command (Команда) – це поведінковий шаблон, який інкапсулює запит як об'єкт, що дозволяє параметризувати методи виклику, ставити їх у чергу або записувати для подальшого виконання. Шаблон Command дозволяє розділяти запити та їх виконання, а також легко реалізовувати операції скасування (undo) чи повторення (redo). Це дозволяє зручно працювати з виконанням команд у додатках.

Chain of Responsibility (Цепочка відповідальності) – це поведінковий шаблон, який дозволяє передавати запит по ланцюгу обробників. Кожен обробник може або обробити запит, або передати його наступному обробнику в ланцюгу. Це дає

змогу динамічно змінювати порядок обробки запитів, а також зменшує зв'язність між клієнтом і обробниками.

Prototype (Прототип) – це створювальний шаблон, який дозволяє копіювати існуючі об'єкти замість їх створення з нуля. За допомогою шаблону Prototype об'єкт може створити нові екземпляри на основі себе, зменшуючи накладні витрати на створення складних об'єктів. Цей шаблон корисний, коли потрібно створювати багато подібних об'єктів, але зі змінами, які важко досягти через звичайне конструювання.

Хід роботи

..25 Installer generator (iterator, builder, factory method, bridge, interpreter, client-server)

Генератор інсталяційних пакетів повинен мати якийсь спосіб налаштування файлів, що входять в установку, установки вікон з інтерактивними можливостями (галочка - створити ярлик на робочому столі; ввести в текстове поле деякі дані, наприклад, ліцензійний ключ і т.д.). Генератор повинен вивести один файл .exe або .msi.

У своїй лабораторній я використовую патерн Builder тому що він ідеально підходить для задачі створення складних об'єктів (інсталяторів) із багатьма параметрами, які можуть варіюватися в залежності від потреб.

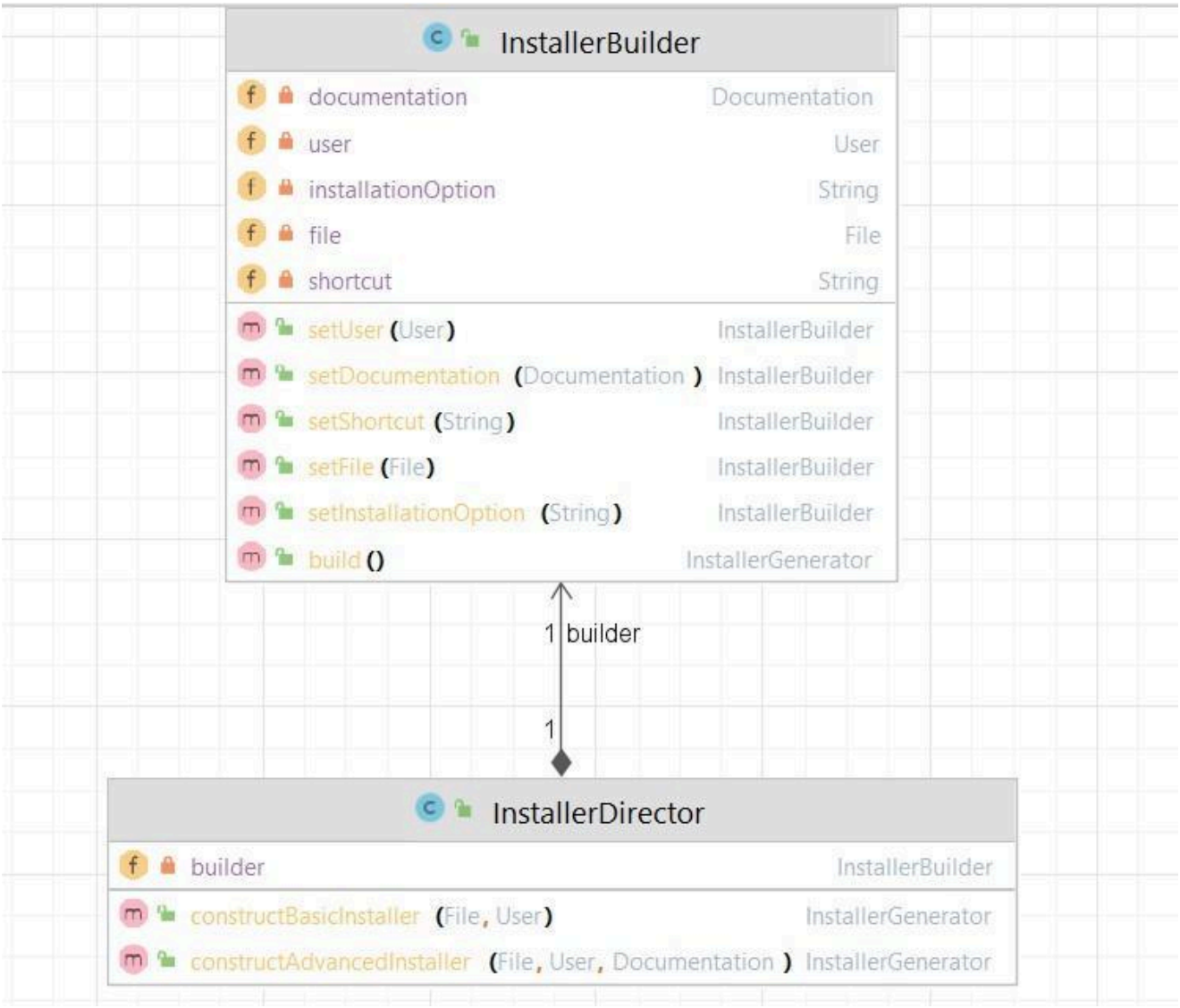
Інсталятори мають багато параметрів, які можна налаштувати:

- Основний файл інсталяції (File).
- Документація (Documentation).
- Додаткові опції встановлення (InstallationOption).
- Створення ярликів, введення ліцензійних ключів тощо.

Кожен параметр може бути необов’язковим або відрізнятися в залежності від сценарію. Патерн **Builder** дозволяє поетапно налаштовувати всі ці параметри, що робить створення об'єкта більш гнучким і зрозумілим.

Якщо в майбутньому я захочу додати нові можливості до інсталяторів (наприклад, підтримку додаткових файлів або спеціальних налаштувань), то не доведеться змінювати код базового класу InstallerGenerator. Достатньо буде додати нові методи в InstallerBuilder, що дотримується принципу **SOLID (Open/Closed Principle)**

Структура Builder



```
You, 3 hours ago | 1 author (You)
public class InstallerBuilder {

  private File file;
  private User user;
  private String installationOption;
  private String shortcut;
  private Documentation documentation;

  public InstallerBuilder setFile(File file) {
    this.file = file;
    return this;
  }

  public InstallerBuilder setUser(User user) {
    this.user = user;
    return this;
  }

  public InstallerBuilder setInstallationOption(String installationOption) {
    this.installationOption = installationOption;
    return this;
  }

  public InstallerBuilder setShortcut(String shortcut) {
    this.shortcut = shortcut;
    return this;
  }

  public InstallerBuilder setDocumentation(Documentation documentation) {
    this.documentation = documentation;
    return this;
  }
```

```
  public InstallerGenerator build() {
    InstallerGenerator generator = new InstallerGenerator();
    generator.setFile(this.file);
    generator.setUser(this.user);
    generator.setInstallationOption(this.installationOption);
    generator.setShortcut(this.shortcut);
    generator.setDocumentation(this.documentation);
    return generator;
  }
}
```

Рисунок 1. Клас InstallerBuilder

```
You, 3 hours ago | 1 author (You)
public class InstallerDirector {
  private InstallerBuilder builder;

  public InstallerDirector(InstallerBuilder builder) {
    this.builder = builder;
  }

  public InstallerGenerator constructBasicInstaller(File file, User user) {
    return builder
        .setFile(file)
        .setUser(user)
        .setInstallationOption(installationOption: "Default Option")
        .setShortcut(shortcut: "No Shortcut")
        .build();
  }

  public InstallerGenerator constructAdvancedInstaller(File file, User user, Documentation documentati
    return builder
        .setFile(file)
        .setUser(user)
        .setDocumentation(documentation)
        .setInstallationOption(installationOption: "Install for all users")
        .setShortcut(shortcut: "Create Desktop Shortcut")
        .build();
  }
}
```

Рисунок 2. Клас InstallerDirector

InstallerDirector управляє процесом створення інсталяторів. Він використовує InstallerBuilder для побудови інсталяторів з конкретними налаштуваннями.

Методи constructBasicInstaller(File file, User user): Створює базовий інсталятор з мінімальними налаштуваннями.

constructAdvancedInstaller(File file, User user, Documentation documentation): Створює розширений інсталятор з додатковими параметрами.

Висновок: Завдяки використанню шаблону **Builder** створення об'єкта інсталятора стало зрозумілим і зручним. Це дозволяє модифікувати процес побудови інсталяторів без необхідності змінювати існуючий код базового класу InstallerGenerator. Додаткові можливості, такі як нові параметри чи опції, можуть бути додані просто через нові методи в класі InstallerBuilder, що відповідає принципу SOLID (зокрема, принципу відкритості/закритості).

.