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 №6
Технологія розроблення програмного забезпечення

“Патерни проектування”

Виконав
студент групи ІА-31
Даценко Влад

Перевірив:
Мягкий Михайло
Юрійович

Київ 2025

Тема: Патерни проектування.

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.

Короткі теоретичні відомості

Factory Method (Фабричний метод) – це створювальний шаблон, який визначає спільний інтерфейс для створення об'єкта, але дозволяє підкласам змінювати тип створюваного об'єкта. Це дає змогу делегувати логіку створення екземплярів спадкоємцям батьківського класу. Шаблон Factory Method корисний, коли системі не потрібно знати про конкретні класи об'єктів, з якими вона працює, що полегшує додавання нових типів продуктів у програму.

Abstract Factory (Абстрактна фабрика) – це створювальний шаблон, який надає інтерфейс для створення сімейств взаємопов'язаних або залежних об'єктів без вказівки їхніх конкретних класів. Це дозволяє гарантувати, що створені об'єкти (наприклад, вікна, кнопки та чекбокси) будуть належати до одного стилю і коректно взаємодіяти між собою. Абстрактна фабрика використовується, коли система повинна бути незалежною від способу створення та компонування своїх складових.

Memento (Знімок) – це поведінковий шаблон, який дозволяє зберігати та

відновлювати попередній стан об'єкта, не розкриваючи подробиць його реалізації (не порушуючи інкапсуляцію). Шаблон пропонує зберігати стан у спеціальному об'єкті-знімку, до якого має повний доступ лише його творець. Memento є стандартним рішенням для реалізації функцій «скасувати» (undo) та «повторити» (redo) у редакторах та інших складних додатках.

Observer (Спостерігач) – це поведінковий шаблон, який створює механізм підписки для сповіщення декількох об'єктів про події, що відбуваються з іншим об'єктом. Це формує залежність «один-до-багатьох», де при зміні стану одного об'єкта (видавця) всі залежні від нього об'єкти (підписники) автоматично отримують повідомлення про це. Шаблон часто застосовується в розробці інтерфейсів користувача та системах, керованих подіями.

Decorator (Декоратор) – це структурний шаблон, який дозволяє динамічно додавати об'єктам нову функціональність, загортаючи їх у спеціальні об'єкти-обгортки (wrappers). Декоратор має такий самий інтерфейс, як і декорований об'єкт, але розширює його поведінку, виконуючи додаткові дії до або після виклику основного методу. Це забезпечує гнучку альтернативу наслідуванню для розширення можливостей класів без зміни їхнього коду.

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.

Використані патерни:
-Factory Method/Abstract Factory

-Memento

-Observer

-Decorator

Код:

```
package com.generator_installer.project.example;



import java.util.*;



public class ExamplePatterns {

    // Factory Method

    interface Installer {

        void install();

    }

    static class WindowsInstaller implements Installer {

        public void install() {

            System.out.println("Installing on Windows platform");

        }

    }

    abstract static class InstallerFactory {

        public abstract Installer createInstaller();

    }

    static class WindowsInstallerFactory extends InstallerFactory {
```

```
        public Installer createInstaller() {
            return new WindowsInstaller();
        }
    }

    // Memento
    static class InstallerMemento {
        private final String state;
        public InstallerMemento(String state) { this.state = state; }
        public String getState() { return state; }
    }
    static class InstallerOriginator {
        private String state;
        public void setState(String state) { this.state = state; }
        public String getState() { return state; }
        public InstallerMemento saveStateToMemento() { return new
InstallerMemento(state); }
        public void getStateFromMemento(InstallerMemento memento) { this.state
= memento.getState(); }
    }
    static class InstallerCaretaker {
        private final List<InstallerMemento> mementoList = new ArrayList<>();
        public void add(InstallerMemento state) { mementoList.add(state); }
        public InstallerMemento get(int index) { return
mementoList.get(index); }
```

```
    }


    // Observer
    interface InstallerEventListener {
        void update(String event);
    }
    static class InstallerEventManager {
        private final List<InstallerEventListener> listeners = new
ArrayList<>();
        public void subscribe(InstallerEventListener listener) {
listeners.add(listener); }
        public void notifyEvent(String event) {
            for (InstallerEventListener listener : listeners) {
                listener.update(event);
            }
        }
    }


    // Decorator
    static class InstallerGenerator {
        public String getCurrentState() { return "Base Installer"; }
    }
    static abstract class InstallerDecorator extends InstallerGenerator {
        protected InstallerGenerator installerGenerator;
```

```
        public InstallerDecorator(InstallerGenerator installerGenerator) {
            this.installerGenerator = installerGenerator;
        }
        public abstract String getDescription();
    }
    static class LoggingInstallerDecorator extends InstallerDecorator {
        public LoggingInstallerDecorator(InstallerGenerator
installerGenerator) {
            super(installerGenerator);
        }
        public String getDescription() {
            return installerGenerator.getCurrentState() + " + Logging";
        }
    }


    // Main method for demonstration
    public static void main(String[] args) {
        // Factory Method
        InstallerFactory factory = new WindowsInstallerFactory();
        Installer installer = factory.createInstaller();
        installer.install();


        // Memento
        InstallerOriginator originator = new InstallerOriginator();
```

```
        InstallerCaretaker caretaker = new InstallerCaretaker();

        originator.setState("State1");

        caretaker.add(originator.saveStateToMemento());

        originator.setState("State2");

        System.out.println("Current state: " + originator.getState());

        originator.getStateFromMemento(caretaker.get(0));

        System.out.println("Restored state: " + originator.getState());



        // Observer

        InstallerEventManager eventManager = new InstallerEventManager();

        eventManager.subscribe(event -> System.out.println("Received event: "
+ event));

        eventManager.notifyEvent("Install started");



        // Decorator

        InstallerGenerator base = new InstallerGenerator();

        InstallerDecorator decorated = new LoggingInstallerDecorator(base);

        System.out.println(decorated.getDescription());

    }

}
```

Висновок: У лабораторній роботі реалізовано та інтегровано чотири патерни проектування. Діаграма класів і фрагменти коду демонструють їхню взаємодію у системі.

.