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.

В моїй лабораторній роботі розглянуто патерн Factory Method.

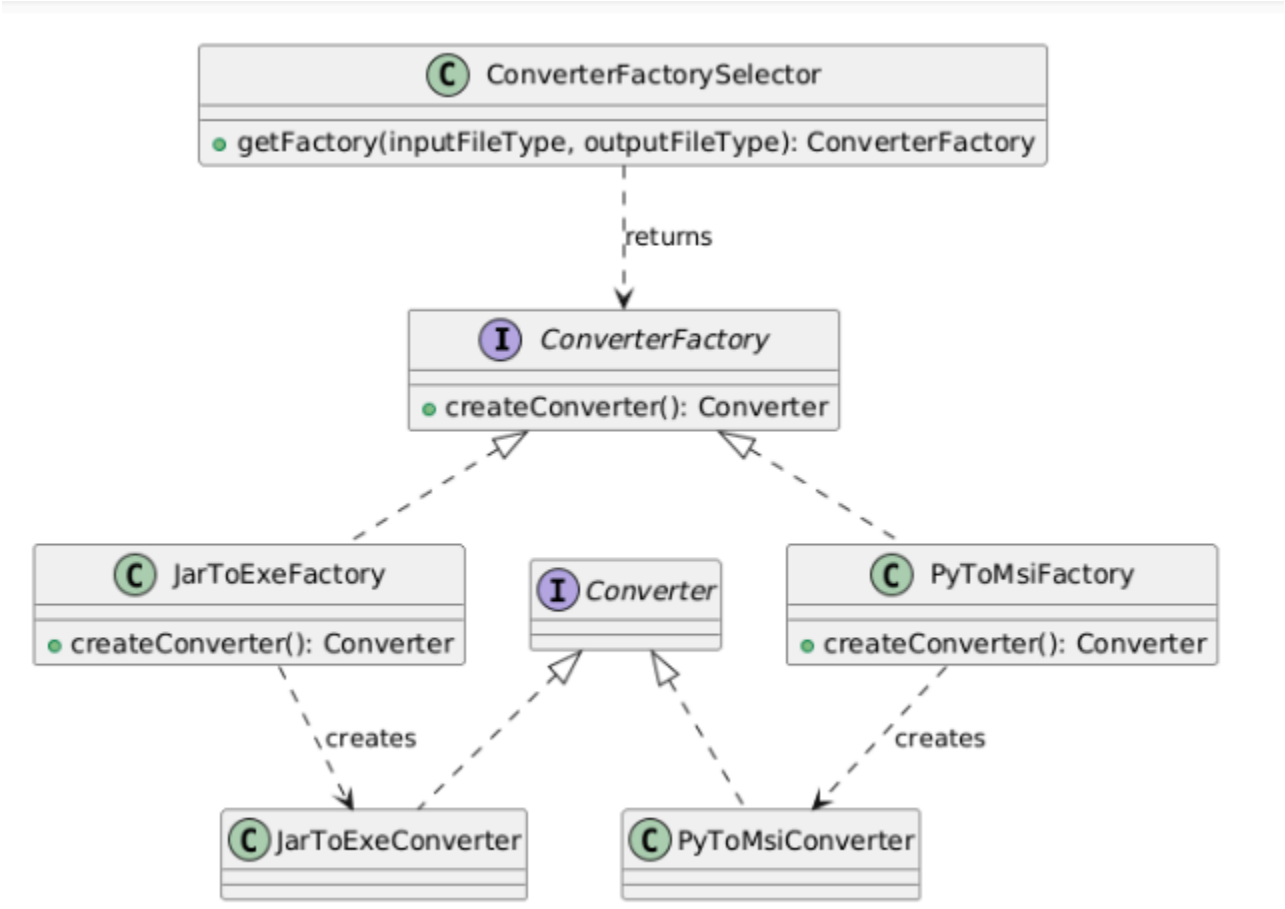


Рисунок 1. Структура патерну Factory Method.

Я вирішив використати патерн Factory Method для створення об’єктів-конвертерів різних типів файлів. Це дозволяє мені легко додавати нові типи конвертації (наприклад, з .jar у .exe, з .py у .msi тощо) без зміни основної логіки програми. Кожна фабрика (наприклад, JarToExeFactory, PyToMsiFactory) відповідає за створення конкретного конвертера, який реалізує потрібний функціонал.

Завдяки такому підходу створення об’єктів-конвертерів відбувається через спільний інтерфейс ConverterFactory, а вибір конкретної фабрики здійснюється динамічно залежно від типу вхідного та вихідного файлу. Це забезпечує гнучкість і розширюваність системи: для підтримки нового формату достатньо додати нову фабрику та конвертер, не змінюючи існуючий код.

Factory Method дозволяє ізолювати процес створення об’єктів, спростити підтримку та тестування, а також уникнути дублювання коду. Така структура робить систему простою для розширення і зрозумілою для розробників, що особливо важливо при роботі з великою кількістю різних форматів і сценаріїв конвертації.

Код:

```
You, 3 days ago | 1 author (You)
public interface ConverterFactory {
    Converter createConverter();
}
```

Рисунок 2.Клас ConverterFactory

```
public class ConverterFactorySelector {
    public static ConverterFactory getFactory(String inputFileType, String outputFileType) {
        if (inputFileType.equalsIgnoreCase(anotherString: "py") && outputFileType.equalsIgnoreCase(anotherString:
            return new PyToExeFactory();
        } else if (inputFileType.equalsIgnoreCase(anotherString: "py") && outputFileType.equalsIgnoreCase(another
            return new PyToMsiFactory();
        } else if (inputFileType.equalsIgnoreCase(anotherString: "jar") && outputFileType.equalsIgnoreCase(another
            return new JarToExeFactory();
        } else if (inputFileType.equalsIgnoreCase(anotherString: "jar") && outputFileType.equalsIgnoreCase(another
            return new CombinedJarToMsiFactory();
        }
        throw new IllegalArgumentException(s: "Unsupported conversion type");
    }
}
```

Рисунок 3.Клас ConverterFactorySelector

```
public class PyToExeFactory implements ConverterFactory {

    public Converter createConverter() {
        return new PyToExeConverter();
    }
}
```

Рисунок 4.Клас PyToExeFactory

```
public class PyToMsiFactory implements ConverterFactory {
  @Override
  public Converter createConverter() {
    return new PyToMsiConverter();
  }
}
```

Рисунок 5.Клас PyToMsiFactory

```
public  class JarToExeFactory implements ConverterFactory {
  @Override
  public Converter createConverter() {
    return new JarToExeConverter();
  }
}
```

Рисунок 6.Клас JarToExeFactory

```
public class CombinedJarToMsiFactory implements ConverterFactory {

  @Override
  public Converter createConverter() {
    return new CombinedJarToMsiConverter();
  }
}
```

Рисунок 7.Клас CompinedJarToMsiFactory

Висновок: У ході виконання лабораторної роботи було реалізовано патерн Factory Method для створення об’єктів-конвертерів різних типів файлів. Це дозволило організувати гнучку та розширювану систему, у якій додавання нових форматів конвертації не потребує змін у основній логіці програми. Кожна фабрика відповідає за створення конкретного конвертера, що спрощує підтримку та тестування коду.

Використання Factory Method забезпечило ізоляцію процесу створення об’єктів, зменшило дублювання коду та підвищило зрозумілість архітектури. Такий підхід дозволяє легко масштабувати систему, додаючи нові типи конвертації, і робить її зручною для подальшого розвитку. Реалізація патерну Factory Method у даній роботі є виправданою та ефективною для вирішення поставлених завдань.

.