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
ШАБЛОНИ «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»
Варіант: 25. Installer generator

Виконав:

Студент групи ІА-22

Птачик Р.С.

Перевірив:

Мягкий М.Ю.

Київ 2024

Зміст:

Тема: шаблони «abstract factory», «factory method», «memento», «observer», «decorator».

Мета: ознайомитися з шаблонами проектування «abstract factory», «factory method», «memento», «observer», «decorator». Реалізувати частину функціоналу програми за допомогою одного з розглянутих шаблонів для досягнення конкретних функціональних можливостей та забезпечення ефективної взаємодії між класами.

Хід роботи:

Тема:

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.

Короткі теоретичні відомості

Принципи проектування SOLID

Принципи SOLID — це п’ять основних правил об'єктно-орієнтованого проектування, сформульованих Робертом Мартіном. Вони дозволяють створювати масштабовані, зрозумілі та легкі для підтримки системи. Розглянемо їх докладніше:

Принцип єдиного обов'язку (SRP)

- Клас повинен виконувати лише одну задачу, що мінімізує складність та підвищує підтримуваність коду. Наприклад, якщо клас обробляє звіти і друкує їх, будь-яка зміна у звітності може спричинити помилки в логіці друку.

Принцип відкритості/закритості (OCP)

- Компоненти мають бути відкритими для розширення, але закритими для змін. Це досягається за допомогою використання інтерфейсів чи

абстрактних класів, що дозволяє змінювати функціональність через додавання нових класів, а не зміну наявних.

Принцип підстановки Барбари Лісков (LSP)

- Підкласи повинні заміняти базові класи без порушення логіки програми. Наприклад, якщо метод базового класу додає числа, підклас не повинен змінювати цю логіку, щоб виконувати множення.

Принцип розділення інтерфейсу (ISP)

- Інтерфейси повинні бути вузькоспеціалізованими. Замість великого універсального інтерфейсу краще створити кілька дрібніших, які реалізують лише необхідні функції для конкретного клієнта.

Принцип інверсії залежностей (DIP)

- Модулі верхнього рівня не повинні залежати від деталей, а деталі — від абстракцій. Наприклад, якщо програма залежить від конкретної бази даних, її важко масштабувати чи тестувати. Використання інтерфейсів вирішує цю проблему.

Шаблони проектування

Шаблон "Abstract Factory" (Абстрактна фабрика)

Призначення: забезпечує створення сімейств пов'язаних або залежних об'єктів без вказівки їх конкретних класів. Це дозволяє клієнту працювати з об'єктами, не знаючи їх реалізації, що знижує залежність між компонентами.

Проблема: необхідно створити об'єкт, але його тип відомий лише під час виконання програми.

Рішення: реалізація абстрактної фабрики, що визначає інтерфейси для створення об'єктів певного сімейства. Конкретні фабрики реалізують ці інтерфейси для створення потрібних об'єктів.

Приклад: для графічного редактора можна створити фабрики для Windows та MacOS, кожна з яких генерує інтерфейсні компоненти відповідно до платформи.

Переваги:

- Покращує розширюваність системи.
- Мінімізує залежність клієнтського коду від конкретних реалізацій.

- Дотримується принципу відкритості/закритості.

Недоліки:

- Збільшує складність коду через велику кількість класів і інтерфейсів.
- Надмірна абстракція може ускладнити розуміння.

Шаблон "Factory Method" (Фабричний метод)

Призначення: дозволяє створювати об'єкти без вказівки їх конкретного класу. Тип створюваного об'єкта визначається підкласом.

Проблема: як зберігати стан об'єкта для його відновлення без доступу до приватних даних.

Рішення: використання фабричного методу в абстрактному класі. Підкласи реалізують цей метод для створення об'єктів потрібного типу.

Приклад: у медіаплеєрі можна реалізувати фабричний метод для створення декодерів різних форматів (MP3, FLAC, WAV).

Переваги:

- Легке додавання нових типів об'єктів.
- Зменшує залежність від конкретних реалізацій.

Недоліки:

- Створення додаткових класів для кожного нового типу.
- Потребує підтримки додаткової абстракції.

Шаблон "Memento" (Мементо)

Призначення: дозволяє зберігати та відновлювати стан об'єкта без порушення інкапсуляції.

Проблема: як зберігати стан об'єкта для його відновлення без доступу до приватних даних.

Рішення: стан зберігається у спеціальному об'єкті "мементо", який може бути використаний для його відновлення.

Приклад: у текстовому редакторі можна використовувати мементо для реалізації функцій "скасування" та "повтор".

Переваги:

- Забезпечує інкапсуляцію даних.
- Зручний для реалізації історії змін.

Недоліки:

- Може вимагати багато пам'яті для складних об'єктів.
- Ускладнює код за рахунок створення окремих мементо.

Шаблон "Observer" (Спостерігач)

Призначення: створює залежність "один до багатьох", дозволяючи всім спостерігачам отримувати повідомлення про зміну стану об'єкта.

Проблема: необхідно оновлювати кілька об'єктів при зміні стану одного з них, уникнувши прямої залежності між ними.

Рішення: реалізація системи підписок. Спостерігачі "підписуються" на зміни, а об'єкт повідомляє їх про оновлення.

Приклад: у банківській системі зміна балансу клієнта оновлює відображення в інтерфейсі користувача.

Переваги:

- Полегшує управління залежностями.
- Легко додаються нові спостерігачі.

Недоліки:

- Можлива велика кількість оновлень.
- Ускладнене тестування через численні взаємодії.

Шаблон "Decorator" (Декоратор)

Призначення: динамічно додає нові функціональні можливості об'єктам без зміни їхньої структури.

Проблема: необхідно розширити функціональність об'єкта, зберігаючи існуючі можливості, без створення великої кількості підкласів.

Рішення: декоратор "обгортає" об'єкт, додаючи нові можливості. Декоратори можуть бути комбіновані.

Приклад: у редакторі зображень можна додавати нові фільтри або ефекти через декоратори, не змінюючи базовий код.

Переваги:

- Гнучке розширення функціональності.
- Збереження базової реалізації.

Недоліки:

- Може ускладнити розуміння через велику кількість декораторів.
- Додаткові витрати на обробку об'єктів.

Реалізувати не менше 3-х класів відповідно до обраної теми

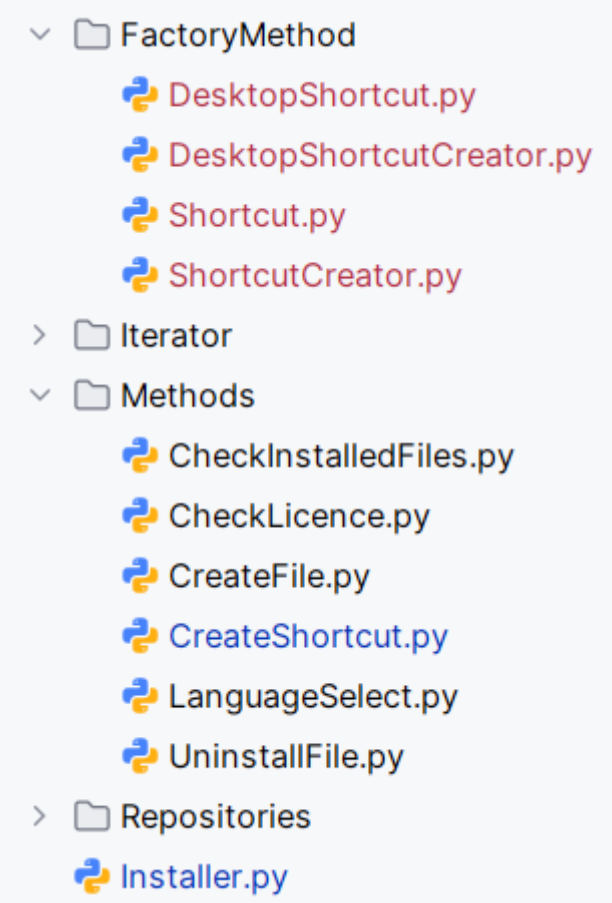

Рис. 1 – Структура проекту

В цій лабораторній роботі було реалізовано частину генератора інсталяційних пакетів з використанням шаблону factory method, а саме створення ярлика.

Реалізувати один з розглянутих шаблонів за обраною темою

Згідно з завданням, було використано та реалізовано шаблон factory method.

1. Shortcut:

- Визначає загальний інтерфейс через метод create, який є абстрактним і має бути реалізований у підкласах.
- Абстрактний клас забезпечує єдиний інтерфейс для всіх ярликів, але його не можна використовувати напряму.

2. DesktopShortcut:

- Є конкретною реалізацією абстрактного класу Shortcut.
- Реалізує метод create, який створює ярлик на робочому столі.

3. ShortcutCreator:

- Базовий клас для створення ярликів із використанням фабричного методу.
- Містить метод create_shortcut, який відповідає за створення об'єкта ярлика та виклик методу create.
- Містить метод factory_method, який є абстрактним і має бути перевизначений у підкласах.
- Цей клас виступає контекстом для створення об'єктів типу Shortcut.

4. DesktopShortcutCreator:

- Конкретна реалізація класу ShortcutCreator.
- Реалізує метод factory_method, який повертає об'єкт класу DesktopShortcut.
- Цей клас відповідає за створення конкретних об'єктів ярликів для робочого столу.

5. createShortcut():

- Виступає в ролі фасаду, що спрощує виклик процесу створення ярлика. Вона створює об'єкт DesktopShortcutCreator і викликає метод create_shortcut.

6. Installer:

- Використовує функцію createShortcut для створення ярликів у процесі установки.

- Метод create_shortcut(shortcut_name, shortcut_path): Приймає ім'я ярлика і шлях, передає ці параметри функції createShortcut.

Діаграма класів для шаблону factory method

Діаграма класів для шаблону factory method:

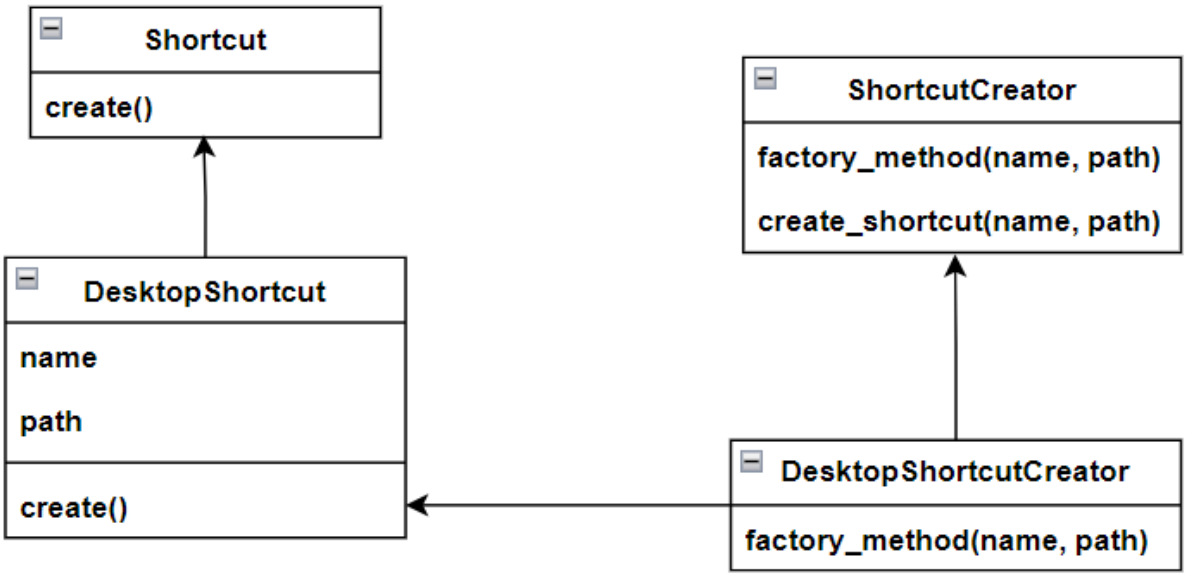

Рис. 7 – Діаграма класів для шаблону factory method

Проблема, яку допоміг вирішити шаблон factory method

Використання шаблону factory method у проекті "Installer Generator" вирішило проблему гнучкого та узгодженого створення об'єктів ярликів різних типів, таких як ярлики для робочого столу.

Завдяки шаблону factory method вдалося:

- Інкапсулювати логіку створення конкретних типів ярликів у підкласах, що зменшило залежність базового коду від конкретних реалізацій.
- Забезпечити розширюваність, дозволяючи додавати нові типи ярликів без змін у базовому коді.
- Спрощувати код шляхом винесення створення об'єктів у окремі класи, що зробило його більш читабельним і структурованим.
- Підвищити узгодженість і повторне використання коду, завдяки єдиному інтерфейсу для створення ярликів.

Це рішення зробило процес створення ярликів гнучким, модульним та легко розширюваним, що значно полегшує підтримку та масштабування проекту "Installer Generator".

Переваги використання шаблону factory method

Ізоляція створення об'єктів:

- Клієнтський код не потребує знань про конкретні класи, які створюються. Інтерфейс або абстрактний клас гарантують єдиний спосіб створення об'єктів.
- Логіка створення об'єктів ізольована в методі factory_method, що забезпечує простий і зрозумілий інтерфейс для роботи з продуктами.

Гнучкість:

- Можна легко додавати нові типи продуктів (наприклад, різні види ярликів), не змінюючи існуючий клієнтський код.
- Завдяки реалізації шаблону в підкласах, забезпечується можливість створення об'єктів із різними характеристиками або поведінкою.

Дотримання принципу відкритості/закритості (OCP):

- Клас ShortcutCreator закритий для модифікації, але відкритий для розширення. Нові продукти можуть додаватися через створення підкласів із реалізацією конкретного factory_method.

Виділення коду створення:

- Уся логіка створення конкретного продукту (наприклад, DesktopShortcut) зосереджена в одному місці — у відповідному підкласі. Це спрощує підтримку коду та зменшує його дублювання.

Узгодженість:

- Шаблон забезпечує єдиний підхід до створення об'єктів через абстракцію, що спрощує читання та розуміння коду.

Завдяки цим перевагам, шаблон Factory Method зробив процес створення об'єктів у проєкті "Installer Generator" більш структурованим, гнучким та адаптованим до змін і розширення.

Код та висновок

Код можна знайти за посиланням:
https://github.com/FryMondo/TRPZ/tree/master/InstallGenerator

Висновок: У цій лабораторній роботі реалізовано шаблон Factory Method для генератора інсталяційних пакетів, що забезпечило гнучке та масштабоване створення об'єктів ярликів. Реалізація включала класи Shortcut, DesktopShortcut, ShortcutCreator та функцію createShortcut(), які спростили логіку створення об'єктів. Використання шаблону дозволило інкапсулювати логіку створення, спростити підтримку коду та забезпечити розширюваність системи без модифікації базового коду.