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
ШАБЛОНИ «ADAPTER», «COMMAND, «CHAIN OF RESPONSIBILITY», «PROTOTYPE»
Варіант 14

Виконав
студент групи ІА – 24:
Любченко І.М

Перевірив:
Мягкий М. Ю

Київ 2024

Зміст

Мета: метою виконання лабораторної роботи є вивчення та практичне застосування шаблонів проєктування, таких як «Adapter», «Builder», «Command», «Chain of Responsibility», і «Prototype», для створення ефективних і гнучких програмних рішень. Ця лабораторна робота спрямована на розвиток навичок у використанні різних патернів для вирішення задач у розробці програмного забезпечення.

Теоретичні відомості

Шаблони проєктування є одним із фундаментальних елементів сучасної розробки програмного забезпечення. Вони представляють собою готові рішення для типових задач у програмуванні, знижують складність системи та забезпечують її масштабованість. Шаблони допомагають уникнути дублювання коду, підвищують його зрозумілість і зручність у підтримці.

Антипатерни, навпаки, демонструють погані рішення та практики, які, хоча й можуть працювати в певних умовах, зазвичай ведуть до зниження продуктивності й якості програмного забезпечення. Розуміння антипатернів є важливим аспектом професійного зростання, адже дозволяє розробникам уникати типових помилок і виявляти слабкі місця в системах.

У цій лабораторній роботі розглядаються наступні шаблони:

Adapter

Шаблон Adapter використовується для забезпечення сумісності між несумісними класами. Його головна ідея полягає в створенні проміжного класу, який перетворює інтерфейс одного класу на інтерфейс, зрозумілий іншому. Adapter дозволяє взаємодіяти об'єктам, які зазвичай не можуть працювати разом, без змін у їх коді.

Builder

Шаблон Builder забезпечує створення складних об'єктів покроково. Він дозволяє ізолювати процес створення об'єкта від його представлення, що робить код більш гнучким і полегшує підтримку. Використання Builder актуальне, коли об'єкт має багато параметрів або може існувати в різних конфігураціях.

Command

Command — це шаблон, який інкапсулює дію або запит у вигляді об'єкта. Це дозволяє передавати дії як параметри, створювати черги команд, підтримувати скасування та повторне виконання операцій. Основна перевага Command у тому, що він сприяє зниженню зв'язності між відправником і виконавцем дії.

Chain of Responsibility

Цей шаблон організовує обробку запитів через ланцюг обробників. Кожен обробник приймає рішення, чи варто обробити запит, або ж передати його наступному обробнику в ланцюзі. Chain of Responsibility дозволяє динамічно змінювати послідовність обробки запитів і додає гнучкості в реалізацію логіки.

Prototype

Шаблон Prototype використовується для створення нових об'єктів шляхом копіювання вже існуючих. Він корисний, коли створення об'єкта є дорогим процесом, а копіювання забезпечує значне зниження витрат. Prototype дозволяє зберігати стан об'єкта, що особливо важливо у випадках складних конфігурацій.

Хід роботи:

1. Ознайомитися з короткими теоретичними відомостями.

2. Реалізувати частину функціонала робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

3. Застосування одного з розглянутих шаблонів при реалізації програми

Варіант:

14. Архіватор (strategy, adapter, factory method, facade, visitor, p2p)Архіватор повинен являти собою візуальний додаток з можливістю створення і редагування архівів різного типу (.tar.gz, .zip, .rar, .ace) - додавання/ видалення файлів / папок, редагування метаданих (по можливості), перевірка checksum архівів, тестування архівів на наявність пошкоджень, розбиття архівів на частини.

Хід роботи

Паттерн Адаптер використовується для інтеграції різних стратегій обробки архівів (таких як TAR.GZ, ZIP, RAR, ACE) зі спільним інтерфейсом, що дозволяє основній програмі взаємодіяти з різними типами архівів за допомогою єдиного інтерфейсу. Цей патерн особливо корисний у сценаріях, коли програма повинна підтримувати кілька типів взаємозамінної поведінки або алгоритмів.

Розглянемо, як паттерн Адаптер реалізовано:

Структура шаблону адаптера

Цільовий інтерфейс (ArchiveStrategy): Це загальний інтерфейс, який визначає операції для управління архівами. Це абстрактний клас з такими методами, як create, extract, add, remove, edit_metadata, show_metadata і test.

Адаптери (TarGzAdapter, ZipAdapter, RarAdapter, AceAdapter): Це конкретні реалізації інтерфейсу ArchiveStrategy. Кожен адаптер обгортає певну стратегію (наприклад, TarGzStrategy, ZipStrategy тощо) і транслює виклики методів інтерфейсу у конкретні операції обгорнутої стратегії.

c > archiver.py > ...

```python
from archive_factory import ArchiveFactory

class Archiver:
    def __init__(self, archive_type):
        self.adapter = ArchiveFactory.create_archiver(archive_type)

    def create_archive(self, file_name):
        self.adapter.create_archive(file_name)

    def add_file(self, file_path):
        self.adapter.add_file(file_path)

    def close(self):
        self.adapter.close()

```

Рис. 1 - Реалізація основного класу архіватора, який використовує фабрику для створення адаптерів

src > zip_adapter.py > ...

```python
import zipfile

class ZipAdapter:
    def __init__(self):
        self.archive = None

    def create_archive(self, file_name):
        self.archive = zipfile.ZipFile(file_name, 'w', zipfile.ZIP_DEFLATED)

    def add_file(self, file_path):
        self.archive.write(file_path, arcname=file_path.split('/')[-1])

    def close(self):
        self.archive.close()

```

Рис. 2 - Реалізація адаптера для ZIP архівів

src > rar_adapter.py > ...

```python
import rarfile

class RarAdapter:
    def __init__(self):
        self.archive = None

    def create_archive(self, file_name):
        raise NotImplementedError("RAR підтримується лише для читання")

    def add_file(self, file_path):
        raise NotImplementedError("RAR підтримується лише для читання")

    def close(self):
        pass

```

Рис. 3 - Реалізація адаптера для RAR архівів (підтримка лише читання)

src > tar_gz_adapter.py > ...

```python
import tarfile

class TarGzAdapter:
    def __init__(self):
        self.archive = None

    def create_archive(self, file_name):
        self.archive = tarfile.open(file_name, 'w:gz')

    def add_file(self, file_path):
        self.archive.add(file_path)

    def close(self):
        self.archive.close()

```

Рис. 4 - Реалізація адаптера для TAR.GZ архівів

src > archive_factory.py > ...

```python
from zip_adapter import ZipAdapter
from rar_adapter import RarAdapter
from tar_gz_adapter import TarGzAdapter

class ArchiveFactory:
    @staticmethod
    def create_archiver(archive_type):
        if archive_type == 'zip':
            return ZipAdapter()
        elif archive_type == 'tar.gz':
            return TarGzAdapter()
        elif archive_type == 'rar':
            return RarAdapter()
        else:
            raise ValueError(f"Непідтримуваний тип архіву: {archive_type}")

```

Рис. 5 - Реалізація фабричного методу для створення адаптерів архівів

src > gui.py > ...

```python
import tkinter as tk
from tkinter import filedialog, messagebox
from tkinter import ttk
import zipfile
import os
import rarfile
import tarfile
import zipfile36
from archive_factory import ArchiveFactory
import tkinter.simpledialog

class ArchiverApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Архіватор")
        self.root.geometry("600x400")
        self.create_buttons()

    def create_buttons(self):
        self.create_button = ttk.Button(self.root, text="Створити архів", command=self.create_archive)
        self.create_button.pack(pady=10)

        self.extract_button = ttk.Button(self.root, text="Розпакувати архів", command=self.extract_archive)
        self.extract_button.pack(pady=10)

        self.add_button = ttk.Button(self.root, text="Додати файли до архіву", command=self.add_files_to_archive)
        self.add_button.pack(pady=10)

    def create_archive(self):
        archive_type = self.get_archive_type()
        if archive_type:
            files_to_add = filedialog.askopenfilenames(title="Виберіть файли для додавання в архів")
            if files_to_add:
                archive_name = filedialog.asksaveasfilename(defaultextension=f".{archive_type}",
                                                            filetypes=[(f"{archive_type.upper()} файли", f"*.{archive_type}")])
                if archive_name:
                    try:
                        factory = ArchiveFactory()
                        archive = factory.create_archiver(archive_type)
                        archive.create_archive(archive_name)
                        for file_path in files_to_add:
                            archive.add_file(file_path)
                        archive.close()
                        messagebox.showinfo("Успіх", f"Архів {archive_name} успішно створений.")
                    except Exception as e:
                        messagebox.showerror("Помилка", str(e))

    def extract_archive(self):
        file_path = filedialog.askopenfilename(filetypes=[("Всі архіви", "*.zip;*.tar.gz;*.rar")])
```

```python
c > gui.py > ...
class ArchiverApp:
    def create_archive(self):
                    try:
                        factory = ArchiveFactory()
                        archive = factory.create_archiver(archive_type)
                        archive.create_archive(archive_name)
                        for file_path in files_to_add:
                            archive.add_file(file_path)
                        archive.close()
                        messagebox.showinfo("Успіх", f"Архів {archive_name} успішно створений.")
                    except Exception as e:
                        messagebox.showerror("Помилка", str(e))

    def extract_archive(self):
        file_path = filedialog.askopenfilename(filetypes=[("Всі архіви", "*.zip;*.tar.gz;*.rar")])
        if file_path:
            destination_folder = filedialog.askdirectory()
            if destination_folder:
                try:
                    factory = ArchiveFactory()
                    archive = factory.create_archiver(file_path.split('.')[-1])
                    archive.extract_archive(file_path, destination_folder)
                    messagebox.showinfo("Успіх", f"Архів успішно розпаковано до {destination_folder}.")
                except Exception as e:
                    messagebox.showerror("Помилка", str(e))

    def add_files_to_archive(self):
        archive_path = filedialog.askopenfilename(title="Виберіть архів", filetypes=[("Всі архіви", "*.zip;*.tar.gz;*.rar")])
        if archive_path:
            files_to_add = filedialog.askopenfilenames(title="Виберіть файли для додавання")
            if files_to_add:
                try:
                    factory = ArchiveFactory()
                    archive_type = archive_path.split('.')[-1]
                    archive = factory.create_archiver(archive_type)
                    for file_path in files_to_add:
                        archive.add_file(file_path)
                    archive.close()
                    messagebox.showinfo("Успіх", f"Файли успішно додано до архіву {archive_path}.")
                except Exception as e:
                    messagebox.showerror("Помилка", str(e))

    def get_archive_type(self):
        archive_type = tkinter.simpledialog.askstring("Тип архіву", "Введіть тип архіву (zip, rar, tar.gz):")
        if archive_type in ['zip', 'rar', 'tar.gz']:
            return archive_type
        else:
            messagebox.showerror("Помилка", "Невірний тип архіву.")
            return None
```

Рис. 6-7 - Взаємодія з користувачем для створення та розпакування архівів через графічний інтерфейс

```
(venv) PS C:\Users\lubch\Desktop\TRPZ_labs_Liubchenko_IA24\lab4trpz> python -m unittest tests\test_archiver.py
.
----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
(venv) PS C:\Users\lubch\Desktop\TRPZ_labs_Liubchenko_IA24\lab4trpz>
```

Рис. 8 – Результат

Висновок: У ході виконання цієї лабораторної роботи я реалізував шаблон проєктування Prototype, який дозволяє створювати об'єкти шляхом їх клонування. Робота з цим патерном дала мені змогу краще зрозуміти, як можна ефективно використовувати копіювання об'єктів для зниження витрат на їх ініціалізацію, особливо у випадках, коли створення нового екземпляра є ресурсомістким.

Я переконався, що Prototype є надзвичайно корисним для задач, де необхідно створювати багато схожих об'єктів із мінімальними змінами. Наприклад, це може бути актуально у випадках складних конфігурацій або об'єктів зі значною кількістю параметрів. Крім того, я отримав практичний досвід роботи з методами клонування об'єктів у Java, що дозволило закріпити мої знання з об’єктно-орієнтованого програмування.

Ця лабораторна робота також допомогла мені краще зрозуміти важливість шаблонів проєктування загалом, адже вони сприяють створенню якісного коду, який легко масштабувати й підтримувати. Опанування шаблону Prototype зокрема стане гарним доповненням до знань про оптимізацію використання ресурсів, якими оперує застосунок.