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 №7

Тема **«Шаблони «Mediator», «Facade», «Bridge», «Template Method»»**

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 «Mediator», «Facade», «Bridge» та «Template Method», 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, розглянуті у цій лабораторній роботі, дозволяють ефективно вирішувати типові проблеми розробки програмного забезпечення, дотримуючись принципів, які забезпечують простоту, структурованість та гнучкість коду. Їх основою є ключові програмні філософії:

- Принцип DRY (Don’t Repeat Yourself) спрямований на усунення повторень у коді, що дозволяє уникати дублювання функціональності. Це сприяє легкості підтримки, знижує ризик помилок і полегшує масштабування, оскільки зміни в одному місці автоматично поширюються на всю систему. У контексті шаблонів цей принцип реалізується через повторне використання загальних структур, наприклад, фасаду чи посередника.
- Принцип KISS (Keep It Simple, Stupid!) наголошує на необхідності створення простих і зрозумілих конструкцій. У складних системах кожен компонент має виконувати лише одну конкретну функцію, що полегшує розуміння та підтримку. Шаблони, як-от "Міст" або "Фасад", ілюструють це, дозволяючи приховати складність через спрощені інтерфейси або розділення абстракції та реалізації.
- Принцип YOLO (You Only Load It Once!) допомагає оптимізувати продуктивність системи, мінімізуючи повторні звернення до ресурсів. Наприклад, при використанні шаблону "Фасад" можна створити єдиний об'єкт, який відповідатиме за ініціалізацію даних, забезпечуючи ефективність виконання запитів.
- Принцип Парето (80/20) нагадує про важливість фокусування на ключових аспектах системи. Наприклад, при реалізації шаблону "Посередник" слід приділити основну увагу найбільш критичним комунікаціям між об'єктами, які забезпечують основну функціональність програми.

- Принцип YAGNI (You Ain’t Gonna Need It) закликає уникати зайвої складності, створюючи лише те, що необхідно в поточний момент. Це проявляється в реалізації шаблону "Шаблонний метод", де забезпечується базова структура алгоритму, а додаткові деталі вводяться лише тоді, коли вони дійсно потрібні.

Шаблон "Mediator"

Шаблон "Посередник" (Mediator) вирішує проблему хаотичних зв'язків між об'єктами в програмній системі. Коли кожен компонент напряму взаємодіє з іншими, виникає сильна залежність між ними, що ускладнює підтримку та розширення системи. "Посередник" дозволяє уникнути цієї плутанини, вводячи єдиний центральний об'єкт, через який координується вся взаємодія. Об'єкти, які використовують посередника, називаються колегами; вони більше не звертаються напряму одне до одного, а передають запити через посередника. Таким чином, кожен компонент знає лише про існування цього посередника, що значно знижує зв'язаність системи. Проте, хоча це спрощує логіку взаємодії компонентів, складність може перенестися в самого посередника, якщо він починає координувати надто багато процесів.

Шаблон "Facade"

Шаблон "Фасад" (Facade) створений для того, щоб приховувати складність програмних підсистем за простим і зрозумілим інтерфейсом. Уявіть систему з численними класами та складною структурою, які потрібно використовувати для виконання певного завдання. Фасад пропонує єдиний вхідний пункт для доступу до цієї системи, спрощуючи її використання. Він слугує своєрідним "обличчям" системи, яке відображає тільки те, що потрібно користувачам, приховуючи зайві деталі. Однак, важливо пам'ятати, що фасад сам по собі не додає нових функцій; він лише спрощує доступ до вже існуючої функціональності.

Шаблон "Bridge"

Шаблон "Міст" (Bridge) розв'язує проблему залежності між абстракцією та її реалізацією. Уявіть, що в системі є кілька абстракцій, кожна з яких може мати різні реалізації. Якщо ці аспекти не розділені, кожна нова абстракція або реалізація потребуватиме значної переробки існуючого коду. "Міст" розділяє ці два рівні, дозволяючи їм розвиватися незалежно. Реалізація цього шаблону включає два окремих ієрархічних дерева: одне для абстракції, а інше для реалізації. Абстракція делегує виклики своїй реалізації, що дозволяє легко змінювати реалізацію без

необхідності переписувати клієнтський код. Це забезпечує велику гнучкість, хоча й додає певної складності на етапі проектування.

Шаблон "Template Method"

Шаблон "Шаблонний метод" (Template Method) допомагає структурувати алгоритми таким чином, щоб загальна логіка залишалася незмінною, а конкретні деталі могли змінюватися підкласами. Він визначає основу алгоритму у вигляді кроків у базовому класі, залишаючи можливість підкласам уточнювати чи перевизначати ці кроки. Це дозволяє забезпечити єдність структури алгоритму для всіх підкласів, зберігаючи при цьому їхню гнучкість у реалізації окремих частин. Наприклад, уявіть алгоритм обробки документа: спільні кроки, як зчитування та збереження, визначаються в базовому класі, а специфічні деталі обробки документа — у підкласах. Шаблонний метод гарантує послідовність виконання основних кроків, водночас дозволяючи змінювати їхню реалізацію там, де це необхідно.

Хід роботи

1. Ознайомитися з короткими теоретичними відомостями.
2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3. Застосування одного з розглянутих шаблонів при реалізації програми.

Тема: Архіватор

Реалізація шаблону Facade

Шаблон «Facade» був обраний мною для спрощення взаємодії з комплексною підсистемою архівування файлів. Завдяки цьому шаблону я зміг створити єдиний високорівневий інтерфейс (`ArchiveFacade`), який інкапсулює деталі роботи з архіватором (`Archiver`). Це зробило код більш читабельним та зрозумілим, оскільки приховало складну логіку створення, додавання файлів та розпакування архівів, дозволяючи взаємодіяти з цими функціями через спрощений інтерфейс.

src > archive_facade.py > ...

```python
from archiver import Archiver

class ArchiveFacade:
    def __init__(self, archive_type):
        self.archiver = Archiver(archive_type)
        print(f"Initializing ArchiveFacade with {archive_type} strategy")

    def create_archive(self, file_name, files_to_add):
        print("Facade: Creating archive...")
        self.archiver.create_archive(file_name)
        for file_path in files_to_add:
            print(f"Facade: Adding {file_path} to archive...")
            self.archiver.add_file(file_path)
        self.archiver.close()
        print("Facade: Archive creation completed.")

    def extract_archive(self, archive_path, destination_folder):
        print("Facade: Extracting archive...")
        self.archiver.extract_archive(archive_path, destination_folder)
        print("Facade: Archive extraction completed.")

    def add_files_to_archive(self, archive_path, files_to_add):
        print("Facade: Adding files to existing archive...")
        self.archiver = Archiver(archive_path.split('.')[-1])
        for file_path in files_to_add:
            print(f"Facade: Adding {file_path} to archive...")
            self.archiver.add_file(file_path)
        self.archiver.close()
        print("Facade: Files added to archive.")

```

Рис 1. - archive_facade.py створює клас ArchiveFacade, який забезпечує спрощений інтерфейс для роботи з архівами (створення архіву, додавання файлів до архіву, розпакування архіву). Він інкапсулює складність взаємодії з класом Archiver.

rc > gui.py > ArchiverApp > get_archive_type

```python
import tkinter as tk
from tkinter import filedialog, messagebox
from tkinter import ttk
from archive_facade import ArchiveFacade  # Імпортуйте новий фасад
from archive_process_observer import ArchiveProcessObserver  # Додайте цей імпорт
import tkinter.simpledialog

class ArchiverApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Архіватор")
        self.root.geometry("600x400")
        self.create_buttons()
        self.facade = None
        self.observers = []

    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 add_observer_to_archiver(self, observer):
        self.observers.append(observer)
        if self.facade is not None:
            self.facade.archiver.add_observer(observer)

    def notify_observers(self, message):
        for observer in self.observers:
            observer.update(message)

    def create_archive(self):
        archive_type = self.get_archive_type()
        if archive_type:
            archive_name = filedialog.asksaveasfilename(defaultextension=f".{archive_type}",
                                                        filetypes=[(f"{archive_type.upper()} файли", f"*.{archive_type}")])
            if archive_name:
                try:
                    files_to_add = filedialog.askopenfilenames(title="Виберіть файли для додавання в архів")
                    if files_to_add:
                        self.facade = ArchiveFacade(archive_type)
                        observer = ArchiveProcessObserver()
                        self.add_observer_to_archiver(observer)
                        self.facade.create_archive(archive_name, files_to_add)
                        messagebox.showinfo("Успіх", f"Архів {archive_name} успішно створений.")
```

src > gui.py > ArchiverApp > get_archive_type

```
class ArchiverApp:
    def create_archive(self):
                            self.facade.create_archive(archive_name, files_to_add)
                            messagebox.showinfo("Успіх", f"Архів {archive_name} успішно створений.")
                    except Exception as e:
                        self.notify_observers(f"Error: {str(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:
                    archive_type = file_path.split('.')[-1]
                    self.facade = ArchiveFacade(archive_type)
                    observer = ArchiveProcessObserver()
                    self.add_observer_to_archiver(observer)
                    self.facade.extract_archive(file_path, destination_folder)
                    messagebox.showinfo("Успіх", f"Архів успішно розпаковано до {destination_folder}.")
                except Exception as e:
                    self.notify_observers(f"Error: {str(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:
                    self.facade = ArchiveFacade(archive_path.split('.')[-1])
                    observer = ArchiveProcessObserver()
                    self.add_observer_to_archiver(observer)
                    self.notify_observers(f"Adding files to archive {archive_path}")
                    self.facade.add_files_to_archive(archive_path, files_to_add)
                    messagebox.showinfo("Успіх", f"Файли успішно додано до архіву {archive_path}.")
                except Exception as e:
                    self.notify_observers(f"Error: {str(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 2-3. - gui.py змінено для використання нового фасаду (ArchiveFacade). Тепер взаємодія з архівами здійснюється через фасад, що спрощує логіку GUI.

src > archiver.py > ...

```python
from archive_factory import ArchiveFactory
from observable import Observable
import os

class Archiver(Observable):
    def __init__(self, archive_type):
        super().__init__()
        self.adapter = ArchiveFactory.create_archiver(archive_type)
        self.files_added = set()
        self.archive_initialized = False

    def create_archive(self, file_name):
        if self.adapter:
            self.adapter.create_archive(file_name)
            self.archive_initialized = True
            self.notify_observers(f"Archive {file_name} created")
        else:
            self.notify_observers("Error: Adapter not initialized")

    def add_file(self, file_path):
        if not self.archive_initialized:
            self.notify_observers("Error: Archive not initialized")
            return

        if os.path.exists(file_path):
            if file_path not in self.files_added:
                self.adapter.add_file(file_path)
                self.files_added.add(file_path)
                self.notify_observers(f"File {file_path} added to archive")
            else:
                self.notify_observers(f"File {file_path} already added to archive")
        else:
            self.notify_observers(f"Error: File {file_path} not found")

    def close(self):
        if not self.archive_initialized:
            self.notify_observers("Error: Archive not initialized")
            return

        if hasattr(self.adapter, 'close'):
            self.adapter.close()
            self.archive_initialized = False
            self.notify_observers("Archive process completed")

    def extract_archive(self, archive_path, destination_folder):
        if self.adapter:
            self.adapter.extract_archive(archive_path, destination_folder)
            self.notify_observers(f"Archive {archive_path} successfully extracted to {destination_folder}")
        else:
            self.notify_observers("Error: Adapter not initialized")
```

Рис 4. - archiver.py змінено для підтримки роботи з фасадом та спостерігачами. Клас Archiver забезпечує логіку створення, додавання та розпакування архівів, а також сповіщення спостерігачів.

```
(venv) PS C:\Users\lubch\Desktop\TRPZ_labs_Liubchenko_IA24\lab7trpz> python src\main.py
Initializing ArchiveFacade with zip strategy
Facade: Creating archive...
Creating ZIP archive...
Notification received: Archive C:/Users/lubch/Desktop/1.zip created
Facade: Adding C:/Users/lubch/Desktop/1.txt to archive...
Adding C:/Users/lubch/Desktop/1.txt to ZIP archive
Notification received: File C:/Users/lubch/Desktop/1.txt added to archive
ZIP archive closed
Notification received: Archive process completed
Facade: Archive creation completed.
```

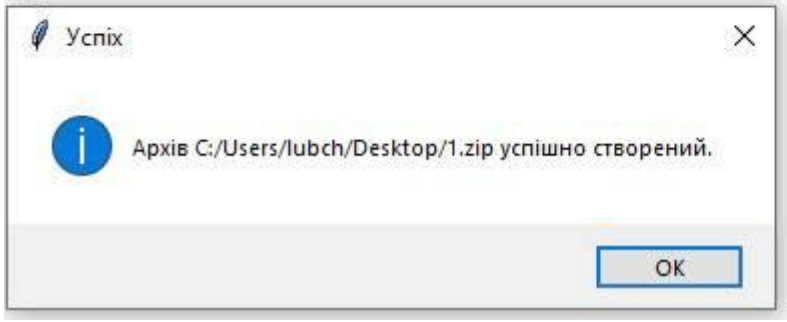

Рис 5. – Створення архіву із шаблоном “Façade”

```
(venv) PS C:\Users\lubch\Desktop\TRPZ_labs_Liubchenko_IA24\lab7trpz> python src\main.py
Initializing ArchiveFacade with zip strategy
Facade: Extracting archive...
Extracting ZIP archive...
Notification received: Archive C:/Users/lubch/Desktop/1.zip successfully extracted to C:/Users/lubch/Desktop
Facade: Archive extraction completed.
```

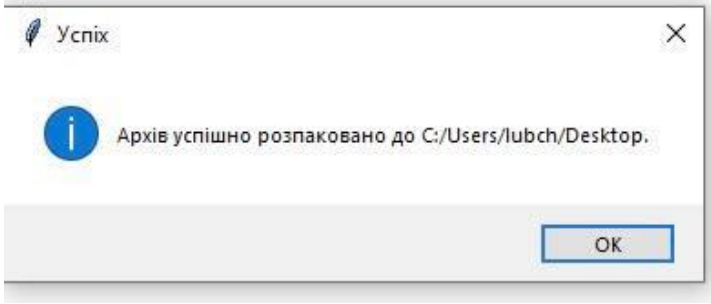

Рис 6. – Розпакування архіву із шаблоном “Facade”

Висновок: У ході виконання цієї лабораторної роботи особливу увагу було приділено вивченню підходів до вирішення типових задач, що виникають під час розробки складних систем. Завдяки використанню шаблонів, таких як "Посередник", "Фасад", "Міст" та "Шаблонний метод", стає можливим зменшення залежностей між компонентами, покращення модульності та спрощення підтримки коду. Ці шаблони демонструють практичне застосування принципів програмування, таких як DRY, KISS та YAGNI, які спрямовані на збереження простоти, уникнення дублювання та ефективність рішень.

Під час роботи я зосередився на імплементації шаблону "Фасад" (Facade). Цей шаблон дозволяє створити єдиний високорівневий інтерфейс, який інкапсулює складну логіку підсистеми, забезпечуючи спрощений доступ до її функціоналу. Це сприяє поліпшенню читабельності коду, спрощує його підтримку і розвиток, дозволяючи зменшити зв’язаність між компонентами системи.

Реалізація шаблону "Фасад" особливо показала важливість дотримання принципу Open-Closed, що дозволяє розширювати функціональність без зміни існуючих модулів. Крім того, я на практиці переконався у важливості простоти в дизайні, адже правильне застосування шаблону дозволяє уникати дублювання та зберігати чітку структуру. В результаті, робота з "Фасадом" стала корисним досвідом, який допоміг краще зрозуміти, як створювати масштабовані рішення та ефективно структурувати залежності.