



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

Тема «Шаблони «**Abstract Factory**», «**Factory Method**», «**Memento**»,
«**Observer**», «**Decorator**»»

Варіант 14

Виконав
студент групи ІА – 24:
Любченко І.М

Перевірив:
Мягкий М. Ю

Київ 2024

Зміст

МЕТА.....	3
Теоретичні відомості.....	3
Хід роботи.....	5
Реалізація шаблону Prototype.....	5
ВИСНОВОК.....	10

Мета: метою виконання лабораторної роботи є вивчення та практичне застосування шаблонів проектування, таких як «Abstract Factory», «Factory Method», «Memento», «Observer» і «Decorator», 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 з даної лабораторної роботи допомагають вирішувати типові задачі у розробці програмного забезпечення, забезпечуючи структурованість, масштабованість та легкість підтримки. Основою їх застосування є принципи SOLID:

- **Single Responsibility Principle:** клас має виконувати лише одну логічну задачу, спрощуючи зміну або розширення його функціональності.
- **Open-Closed Principle:** класи повинні бути відкритими для розширення, але закритими для змін, тобто нову поведінку додають через спадкування чи композицію.
- **Liskov Substitution Principle:** підкласи мають коректно працювати у будьякому контексті, де використовується базовий клас.
- **Interface Segregation Principle:** інтерфейси слід розбивати на дрібні, вузько спрямовані частини, щоб уникати надлишкової реалізації.
- **Dependency Inversion Principle:** модулі вищого рівня не повинні залежати від модулів нижчого рівня, обидва мають залежати від абстракцій.

Розуміння цих принципів є основою правильного застосування шаблонів і створення якісного програмного забезпечення. **Abstract Factory**

Шаблон Abstract Factory дозволяє створювати цілі сімейства взаємопов'язаних об'єктів без вказання їхніх конкретних класів. Його основною ідеєю є надання інтерфейсу, який приховує процес створення об'єктів і робить систему гнучкішою до змін. Наприклад, цей шаблон можна використовувати для розробки графічного інтерфейсу, який має підтримувати 4 різні операційні системи. У цьому випадку фабрика створює набори компонентів, таких як кнопки чи текстові поля, які виглядають і поведуться відповідно до конкретної платформи. Ця гнучкість дозволяє легко замінювати реалізацію без змін у бізнес-логіці програми.

Factory Method

Factory Method забезпечує можливість створювати об'єкти через виклики фабричних методів, які визначаються у підкласах. Це дозволяє класам делегувати створення об'єктів, не прив'язуючись до конкретних реалізацій. Наприклад, у додатку для роботи з повідомленнями можуть бути різні типи повідомлень, такі як SMS, електронна пошта або push-сповіщення. Factory Method дозволяє створювати ці повідомлення залежно від контексту, не змінюючи клієнтський код. Це значно полегшує додавання нових типів повідомлень і робить код більш масштабованим.

Memento

Шаблон Memento використовується для збереження стану об'єкта, щоб його можна було відновити без порушення інкапсуляції. Його основна ідея полягає в тому, щоб створити "знімок" поточного стану об'єкта, який може бути збережений і використаний пізніше для відновлення. Наприклад, у текстових редакторах реалізується функціональність скасування і повтору дій за допомогою цього шаблону. Стан документа зберігається перед виконанням змін, і якщо користувач вирішить скасувати дію, система повертає попередній стан. Цей підхід дозволяє легко управляти змінами без змішування логіки управління станом із бізнес-логікою.

Observer

Observer, або "Спостерігач", дозволяє об'єкту-спостерігачу автоматично отримувати повідомлення про зміну стану іншого об'єкта. Це забезпечує динамічну залежність між об'єктами, дозволяючи їм взаємодіяти без жорсткого зв'язування. Наприклад, цей шаблон використовується у додатках новин, де підписники автоматично отримують оновлення, коли з'являється новина. Суб'єкт, у даному випадку сервер новин, повідомляє всіх підписників про зміни. Такий підхід значно спрощує процес синхронізації та забезпечує 5 легке додавання або видалення спостерігачів.

Decorator

Decorator дозволяє динамічно додавати нову поведінку об'єктам, зберігаючи їхню структуру незмінною. Це досягається шляхом обгортання об'єкта в інший клас, який реалізує додаткову функціональність. Наприклад, у графічних редакторах базовий об'єкт зображення може бути обгорнутий декоратором, який додає ефект тіні, рамки чи іншого візуального елементу. Decorator особливо корисний, коли потрібно розширити функціональність, уникаючи створення великої кількості підкласів для кожного можливого поєднання функцій.

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

Шаблон **Observer** був вибраний мною для проекту, оскільки він дозволяє ефективно керувати сповіщеннями про різні події, такі як створення та розпакування архівів. Це робить систему більш гнучкою, масштабованою та зрозумілою для подальшої підтримки і розвитку.

Реалізація

1. **Клас Observable**: Відповідає за додавання, видалення та сповіщення спостерігачів.
2. **Клас Observer**: Інтерфейс для спостерігачів, який визначає метод `update`.
3. **Клас ArchiveProcessObserver**: Реалізація **Observer**, яка отримує сповіщення та виводить їх у термінал.
4. **Клас Archiver**: Реалізує функціонал архіватора з використанням шаблону **Observer** для сповіщень.

```
src > observable.py > ...
1  class Observable:
2      def __init__(self):
3          self.observers = []
4
5      def add_observer(self, observer):
6          if observer not in self.observers:
7              self.observers.append(observer)
8
9      def remove_observer(self, observer):
10         if observer in self.observers:
11             self.observers.remove(observer)
12
13     def notify_observers(self, message):
14         for observer in self.observers:
15             observer.update(message)
```

Рис 1. - Створення класу **Observable**, який дозволяє додавати, видаляти та сповіщати наглядачів про зміни

```
src > observer.py > ...
1 class Observer:
2     def update(self, message):
3         pass
4
```

Рис 2. - Створення базового класу Observer, який визначає інтерфейс для отримання сповіщень

```
src > archive_process_observer.py > ...
1 from observer import Observer
2
3 class ArchiveProcessObserver(Observer):
4     def update(self, message):
5         print(f"Notification received: {message}")
6
```

Рис 3. - Реалізація класу ArchiveProcessObserver, який отримує сповіщення від архіватора і виводить їх у термінал

```
src > archiver.py > ...
1 from archive_factory import ArchiveFactory
2 from observable import Observable
3 import os
4
5 class Archiver(Observable):
6     def __init__(self, archive_type):
7         super().__init__()
8         self.adapter = ArchiveFactory.create_archiver(archive_type)
9         self.files_added = set()
10        self.archive_initialized = False
11
12    def create_archive(self, file_name):
13        if self.adapter:
14            self.adapter.create_archive(file_name)
15            self.archive_initialized = True
16            self.notify_observers(f"Archive {file_name} created")
17        else:
18            self.notify_observers("Error: Adapter not initialized")
19
20    def add_file(self, file_path):
21        if not self.archive_initialized:
22            self.notify_observers("Error: Archive not initialized")
23            return
24
25        if os.path.exists(file_path):
26            if file_path not in self.files_added:
27                self.adapter.add_file(file_path)
28                self.files_added.add(file_path)
29                self.notify_observers(f"File {file_path} added to archive")
30            else:
31                self.notify_observers(f"File {file_path} already added to archive")
32        else:
33            self.notify_observers(f"Error: File {file_path} not found")
34
35    def close(self):
36        if not self.archive_initialized:
37            self.notify_observers("Error: Archive not initialized")
38            return
39
40        if hasattr(self.adapter, 'close'):
41            self.adapter.close()
42            self.archive_initialized = False
43            self.notify_observers("Archive process completed")
44
45    def extract_archive(self, archive_path, destination_folder):
46        if self.adapter:
47            self.adapter.extract_archive(archive_path, destination_folder)
48            self.notify_observers(f"Archive {archive_path} successfully extracted to {destination_folder}")
49        else:
50            self.notify_observers("Error: Adapter not initialized")
```

Рис 4. - Реалізація класу Archiver, який успадковує Observable і використовує фабрику для створення адаптерів архівів, а також сповіщає наглядців про створення та розпакування архівів


```

rc > guipy > ...
7 class ArchiverApp:
35     def create_archive(self):
52         messagebox.showinfo("Успіх", f"Архів {archive_name} успішно створений.")
53     except Exception as e:
54         self.notify_observers(f"Error: {str(e)}")
55         messagebox.showerror("Помилка", str(e))
56
57     def extract_archive(self):
58         file_path = filedialog.askopenfilename(filetypes=[("Біт архів", "*.zip;*.tar.gz;*.rar")])
59         if file_path:
60             destination_folder = filedialog.askdirectory()
61             if destination_folder:
62                 try:
63                     self.archiver = Archiver(file_path.split('.')[-1])
64                     for observer in self.observers:
65                         self.archiver.add_observer(observer)
66                     self.notify_observers(f"Extracting archive {file_path}")
67                     self.archiver.extract_archive(file_path, destination_folder)
68                     self.notify_observers(f"Archive {file_path} extraction completed successfully")
69                     messagebox.showinfo("Успіх", f"Архів успішно розпаковано до {destination_folder}.")
70                 except Exception as e:
71                     self.notify_observers(f"Error: {str(e)}")
72                     messagebox.showerror("Помилка", str(e))
73
74     def add_files_to_archive(self):
75         archive_path = filedialog.askopenfilename(title="Виберіть архів", filetypes=[("Біт архів", "*.zip;*.tar.gz;*.rar")])
76         if archive_path:
77             files_to_add = filedialog.askopenfilenames(title="Виберіть файли для додавання")
78             if files_to_add:
79                 try:
80                     self.archiver = Archiver(archive_path.split('.')[-1])
81                     for observer in self.observers:
82                         self.archiver.add_observer(observer)
83                     self.notify_observers(f"Adding files to archive {archive_path}")
84                     for file_path in files_to_add:
85                         self.archiver.add_file(file_path)
86                     self.archiver.close()
87                     self.notify_observers("Files added to archive successfully")
88                     messagebox.showinfo("Успіх", f"Файли успішно додано до архіву {archive_path}.")
89                 except Exception as e:
90                     self.notify_observers(f"Error: {str(e)}")
91                     messagebox.showerror("Помилка", str(e))
92
93     def get_archive_type(self):
94         archive_type = tkinter.simpledialog.askstring("Тип архіву", "Введіть тип архіву (zip, rar, tar.gz):")
95         if archive_type in ['zip', 'rar', 'tar.gz']:
96             return archive_type
97         else:
98             messagebox.showerror("Помилка", "Невірний тип архіву.")
99         return None

```

Рис 5-6. - Графічний інтерфейс для архіватора, який дозволяє користувачам створювати та розпаковувати архіви, а також сповіщати наглядачів про ці події


```
(venv) PS C:\Users\lubch\Desktop\TRPZ_labs_Liubchenko_IA24\lab4trpz> python src\main.py
Creating ZIP archive...
Notification received: Archive C:/Users/lubch/Desktop/1.zip created
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
[]
```

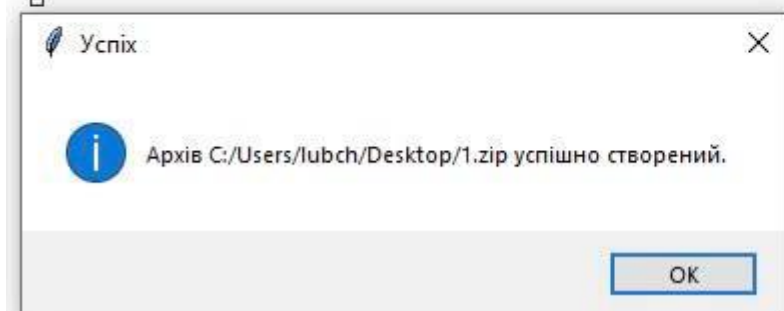


Рис 7. – Сповіщення про успішне створення архіву

```
(venv) PS C:\Users\lubch\Desktop\TRPZ_labs_Liubchenko_IA24\lab4trpz> python src\main.py
User user123 logged in with cookie user123
Notification received: Extracting archive C:/Users/lubch/Desktop/1.zip
Notification received: Archive C:/Users/lubch/Desktop/1.zip successfully extracted to C:/Users/lubch/Desktop
Notification received: Archive C:/Users/lubch/Desktop/1.zip extraction completed successfully
[]
```

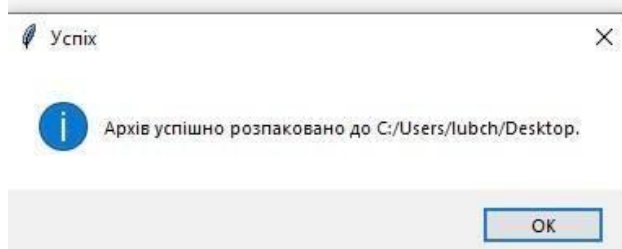


Рис 8. – Сповіщення про успішне розпакування архіву

Висновок: Виконання цієї лабораторної роботи з імплементацією шаблону Observer допомогло мені краще зрозуміти, як реалізовувати розподілену архітектуру сповіщень між об'єктами у складних системах. У результаті я зміг використати цей шаблон для створення функціоналу сповіщень у своєму проєкті архіватора.

Особливо цікаво було розібратися в ролі кожного елемента — Observable, Observer та конкретні реалізації наглядців. Кожен із них чітко виконує свою задачу, і це дозволяє зробити систему не тільки ефективною, але й зрозумілою для підтримки та розвитку.

Також робота змусила мене замислитися над тим, як правильно організувати взаємодію між об'єктами, які спостерігають за змінами, і об'єктами, що їх ініціюють. Це показало важливість балансу між гнучкістю і ефективністю комунікації. У реальних проєктах, де необхідна швидка і надійна реакція на події, шаблон Observer може вимагати додаткової оптимізації чи засобів синхронізації.

У процесі я також переконався, наскільки важливо використовувати готові патерни проєктування. Вони допомагають уникнути повторення коду і спрощують роботу з проєктом у довгостроковій перспективі. Загалом, робота не тільки розширила мої знання, а й додала впевненості у застосуванні шаблонів для розв'язання реальних задач.