



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 №8  
Тема: **ШАБЛони «COMPOSITE», «FLYWEIGHT», «INTERPRETER»,  
«VISITOR»**  
Варіант 14

Виконав  
студент групи ІА – 24:  
Любченко І.М

Перевірив:  
Мягкий М. Ю

**Київ 2024**

**Зміст**

МЕТА.....	3
Теоретичні відомості.....	3
Хід роботи.....	4
Реалізація шаблону Façade.....	4
ВИСНОВОК.....	13

**Мета:** метою виконання лабораторної роботи є вивчення та практичне застосування шаблонів проєктування, таких як «COMPOSITE», «FLYWEIGHT», «INTERPRETER», «VISITOR» для роботи з БД при розробці корпоративних додатків. Ця лабораторна робота спрямована на розвиток навичок у використанні різних патернів для вирішення задач у розробці програмного забезпечення.

## Теоретичні відомості

### Шаблон "Composite"

Шаблон "Composite" дозволяє створювати деревоподібні структури об'єктів, де окремі об'єкти та їх колекції можуть оброблятися однаково. Це особливо корисно для роботи зі складними структурами даних, такими як ієрархії. Основні компоненти шаблону:

- **Component:** Абстрактний клас або інтерфейс, який визначає поведінку, яку мають реалізувати як окремі елементи, так і колекції елементів.
- **Leaf:** Клас, що представляє окремі елементи без підлеглих елементів.
- **Composite:** Клас, що представляє колекції елементів і зберігає підлегли елементи.

### Шаблон "Flyweight"

Шаблон "Flyweight" спрямований на оптимізацію використання пам'яті за рахунок повторного використання вже створених об'єктів. Він дозволяє зменшити кількість об'єктів, що створюються, шляхом розділення стану об'єктів на внутрішній і зовнішній:

- **Внутрішній стан:** Постійний і загальний для всіх об'єктів.
- **Зовнішній стан:** Відмінний і визначається контекстом використання.

### Шаблон "Interpreter"

Шаблон "Interpreter" надає спосіб визначення граматики для мови та інтерпретації її речень. Він особливо корисний для створення мов, які використовуються для специфічних задач:

- **AbstractExpression:** Абстрактний клас, що визначає інтерфейс для всіх виразів.
- **TerminalExpression:** Клас для термінальних виразів у граматиці.

- **NonterminalExpression**: Клас для нетермінальних виразів, що представляють правила граматики.

## **Шаблон "Visitor"**

Шаблон "Visitor" дозволяє додавати нові операції до класів, не змінюючи їх. Він особливо корисний, коли ви хочете додати нові функціональні можливості до об'єктної структури, не змінюючи її:

- **Visitor**: Абстрактний клас або інтерфейс, що визначає операції, які можуть бути виконані над елементами об'єктної структури.
- **ConcreteVisitor**: Конкретні реалізації відвідувача, що визначають поведінку операцій.
- **Element**: Інтерфейс або абстрактний клас для елементів, які можуть бути прийняті відвідувачем.

## **Хід роботи**

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

## **Тема: Архіватор**

### **Реалізація Шаблону Composite в Лабораторній Роботі**

Я використовував шаблон «Composite», оскільки він дозволяє працювати з окремими файлами та колекціями файлів однаково. Це ідеально підходить для архіватора, дозволяючи ефективно додавати файли до архівів, створювати архіви і розпаковувати їх, зберігаючи при цьому чисту структуру і організацію коду.

```

src > component.py > ...
1  from abc import ABC, abstractmethod
2
3  class Component(ABC):
4      @abstractmethod
5      def add(self, component):
6          pass
7
8      @abstractmethod
9      def remove(self, component):
10         pass
11
12     @abstractmethod
13     def display(self, depth):
14         pass
15
16     @abstractmethod
17     def archive(self, archiver):
18         pass

```

Рис 1. - Абстрактний клас Component визначає методи add, remove, display, та archive, які мають реалізувати класи Leaf та Composite.

```

c > leaf.py > ...
1  from component import Component
2
3  class Leaf(Component):
4      def __init__(self, name, path):
5          self.name = name
6          self.path = path
7
8      def add(self, component):
9          raise Exception("Cannot add to a leaf")
10
11     def remove(self, component):
12         raise Exception("Cannot remove from a leaf")
13
14     def display(self, depth):
15         print("-" * depth + self.name)
16
17     def archive(self, archiver, db_manager, archive_id):
18         archiver.add_file(self.path)
19         db_manager.add_file(archive_id, self.path)

```

Рис 2. - Клас Leaf представляє окремі файли. Він реалізує методи Component, але методи add та remove не мають сенсу для окремих файлів, тому генерують виключення. Метод archive додає файл до архіву та записує інформацію до бази даних.

```
rc > composite.py > ...
1  from component import Component
2
3  class Composite(Component):
4      def __init__(self, name):
5          self.name = name
6          self.children = []
7
8      def add(self, component):
9          self.children.append(component)
10
11      def remove(self, component):
12          self.children.remove(component)
13
14      def display(self, depth):
15          print("-" * depth + self.name)
16          for child in self.children:
17              child.display(depth + 2)
18
19      def archive(self, archiver, db_manager, archive_id):
20          for child in self.children:
21              child.archive(archiver, db_manager, archive_id)
```

Рис 3. - Клас Composite представляє колекції файлів і забезпечує можливість додавання, видалення та отримання підлеглих елементів. Метод archive делегує роботу підлеглим елементам і записує інформацію до бази даних.

src &gt; database\_manager.py &gt; ...

```

1  import sqlite3
2
3  class DatabaseManager:
4      def __init__(self, db_name="archiver.db"):
5          self.connection = sqlite3.connect(db_name)
6          self.cursor = self.connection.cursor()
7          self.create_tables()
8
9      def create_tables(self):
10         self.cursor.execute("""
11             CREATE TABLE IF NOT EXISTS archives (
12                 id INTEGER PRIMARY KEY AUTOINCREMENT,
13                 name TEXT NOT NULL
14             )
15         """)
16         self.cursor.execute("""
17             CREATE TABLE IF NOT EXISTS files (
18                 id INTEGER PRIMARY KEY AUTOINCREMENT,
19                 archive_id INTEGER,
20                 path TEXT NOT NULL,
21                 FOREIGN KEY (archive_id) REFERENCES archives (id)
22             )
23         """)
24         self.cursor.execute("""
25             CREATE TABLE IF NOT EXISTS notifications (
26                 id INTEGER PRIMARY KEY AUTOINCREMENT,
27                 message TEXT NOT NULL,
28                 timestamp DATETIME DEFAULT CURRENT_TIMESTAMP
29             )
30         """)
31         self.connection.commit()
32
33     def add_archive(self, name):
34         self.cursor.execute("INSERT INTO archives (name) VALUES (?)", (name,))
35         self.connection.commit()
36         return self.cursor.lastrowid
37
38     def add_file(self, archive_id, file_path):
39         self.cursor.execute("INSERT INTO files (archive_id, path) VALUES (?, ?)", (archive_id, file_path))
40         self.connection.commit()
41
42     def add_notification(self, message):
43         self.cursor.execute("INSERT INTO notifications (message) VALUES (?)", (message,))
44         self.connection.commit()
45
46     def get_archives(self):
47         self.cursor.execute("SELECT * FROM archives")
48         return self.cursor.fetchall()
49
50     def get_files_in_archive(self, archive_id):
51         self.cursor.execute("SELECT path FROM files WHERE archive_id = ?", (archive_id,))
52         return self.cursor.fetchall()
53
54     def close(self):
55         self.connection.close()

```

Рис 4-5. - Клас DatabaseManager забезпечує роботу з базою даних SQLite, включаючи створення таблиць, додавання архівів, файлів і сповіщень, а також отримання даних з бази даних.



```

src > archiver.py > Archiver > add_file
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")
51
52     def archive_component(self, component, db_manager, archive_id):
53         component.archive(self, db_manager, archive_id)
54         self.close()
55         self.notify_observers("Archiving components completed")

```

Рис 6-7. – Додаю метод `archive_component`, який використовує шаблон «Composite» для архівування компонентів і сповіщає спостерігачів.



```
rc > archive_facade.py > ...
1  from archiver import Archiver
2  from database_manager import DatabaseManager
3
4  class ArchiveFacade:
5      def __init__(self, archive_type, db_manager):
6          self.archiver = Archiver(archive_type)
7          self.db_manager = db_manager
8
9      def create_archive(self, file_name, component):
10         archive_id = self.db_manager.add_archive(file_name)
11         self.archiver.create_archive(file_name)
12         self.archiver.archive_component(component, self.db_manager, archive_id)
13
14     def extract_archive(self, archive_path, destination_folder):
15         self.archiver.extract_archive(archive_path, destination_folder)
```

Рис 8. - Додаю параметри db\_manager та archive\_id у метод create\_archive для збереження дій у базу даних.

```
irc > gui.py > ...
```

```

1  import tkinter as tk
2  from tkinter import filedialog, messagebox
3  from tkinter import ttk
4  from archive_facade import ArchiveFacade
5  from archive_process_observer import ArchiveProcessObserver
6  from composite import Composite
7  from leaf import Leaf
8  import tkinter.simpledialog
9  from database_manager import DatabaseManager
10
11 class ArchiverApp:
12     def __init__(self, root):
13         self.root = root
14         self.root.title("Архіватор")
15         self.root.geometry("600x400")
16         self.create_buttons()
17         self.facade = None
18         self.observers = []
19         self.db_manager = DatabaseManager("archiver.db")
20
21         # Ініціалізація компонента дерева
22         self.root_component = Composite("root")
23
24     def create_buttons(self):
25         self.create_button = ttk.Button(self.root, text="Створити архів", command=self.create_archive)
26         self.create_button.pack(pady=10)
27
28         self.extract_button = ttk.Button(self.root, text="Розпакувати архів", command=self.extract_archive)
29         self.extract_button.pack(pady=10)
30
31         self.add_button = ttk.Button(self.root, text="Додати файли до архіву", command=self.add_files_to_archive)
32         self.add_button.pack(pady=10)
33
34         self.show_db_button = ttk.Button(self.root, text="Показати базу даних", command=self.show_database)
35         self.show_db_button.pack(pady=10)
36
37     def add_observer_to_archiver(self, observer):
38         self.observers.append(observer)
39         if self.facade is not None:
40             self.facade.archiver.add_observer(observer)
41
42     def notify_observers(self, message):
43         for observer in self.observers:
44             observer.update(message)
45         self.db_manager.add_notification(message)
46
47     def create_archive(self):
48         archive_type = self.get_archive_type()
49         if archive_type:

```

```

50     archive_name = filedialog.asksaveasfilename(defaultextension=f".{archive_type}",
51                                                  filetypes=[(f"{archive_type.upper()} файли", f"*.{archive_type}")])
52     if archive_name:
53         try:
54             self.facade = ArchiveFacade(archive_type, self.db_manager)
55             self.facade.create_archive(archive_name, self.root_component)
56             messagebox.showinfo("Успіх", f"Архів {archive_name} успішно створений.")
57             self.notify_observers(f"Архів {archive_name} успішно створений.")
58             self.show_database() # Показати базу даних після створення архіву
59         except Exception as e:
60             self.notify_observers(f"Error: {str(e)}")
61             messagebox.showerror("Помилка", str(e))
62
63     def extract_archive(self):
64         file_path = filedialog.askopenfilename(filetypes=[("Бці архіви", "*.zip;*.tar.gz;*.rar")])
65         if file_path:
66             destination_folder = filedialog.askdirectory()
67             if destination_folder:
68                 try:
69                     self.facade = ArchiveFacade(file_path.split('.')[-1], self.db_manager)
70                     self.facade.extract_archive(file_path, destination_folder)
71                     messagebox.showinfo("Успіх", f"Архів успішно розпаковано до {destination_folder}.")
72                     self.notify_observers(f"Архів успішно розпаковано до {destination_folder}.")
73                     self.show_database() # Показати базу даних після розпакування архіву
74                 except Exception as e:
75                     self.notify_observers(f"Error: {str(e)}")
76                     messagebox.showerror("Помилка", str(e))
77
78     def add_files_to_archive(self):
79         files_to_add = filedialog.askopenfilenames(title="Виберіть файли для додавання")
80         if files_to_add:
81             try:
82                 for file_path in files_to_add:
83                     leaf = Leaf(file_path.split("/")[-1], file_path)
84                     self.root_component.add(leaf)
85                     self.notify_observers("Файли успішно додано до архіву.")
86                     messagebox.showinfo("Успіх", "Файли успішно додано до архіву.")
87             except Exception as e:
88                 self.notify_observers(f"Error: {str(e)}")
89                 messagebox.showerror("Помилка", str(e))
90
91     def get_archive_type(self):
92         archive_type = tkinter.simpledialog.askstring("Тип архіву", "Введіть тип архіву (zip, rar, tar.gz):")
93         if archive_type in ['zip', 'rar', 'tar.gz']:
94             return archive_type
95         else:
96             messagebox.showerror("Помилка", "Невірний тип архіву.")
97
98     return None
99
100     def show_database(self):
101         archives = self.db_manager.get_archives()
102         db_info = ""
103         for archive in archives:
104             archive_id, archive_name = archive
105             db_info += f"Архів: {archive_name}\n"
106             files = self.db_manager.get_files_in_archive(archive_id)
107             for file in files:
108                 db_info += f"    Файл: {file[0]}\n"
109         messagebox.showinfo("База даних", db_info)

```

Рис 9-11. - Додаю кнопку для відображення вмісту бази даних і метод show\_database, який отримує дані з бази даних і відображає їх у повідомленні.

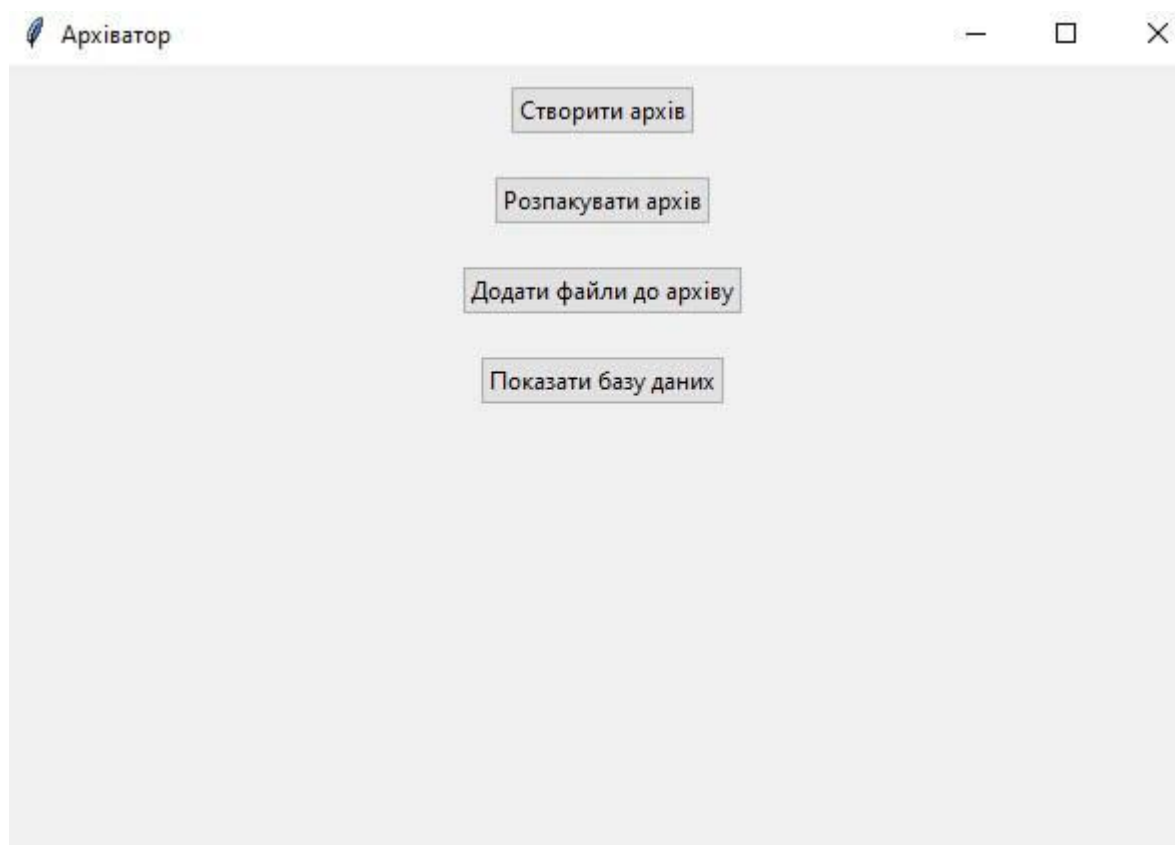


Рис 12. – Панель функціонала

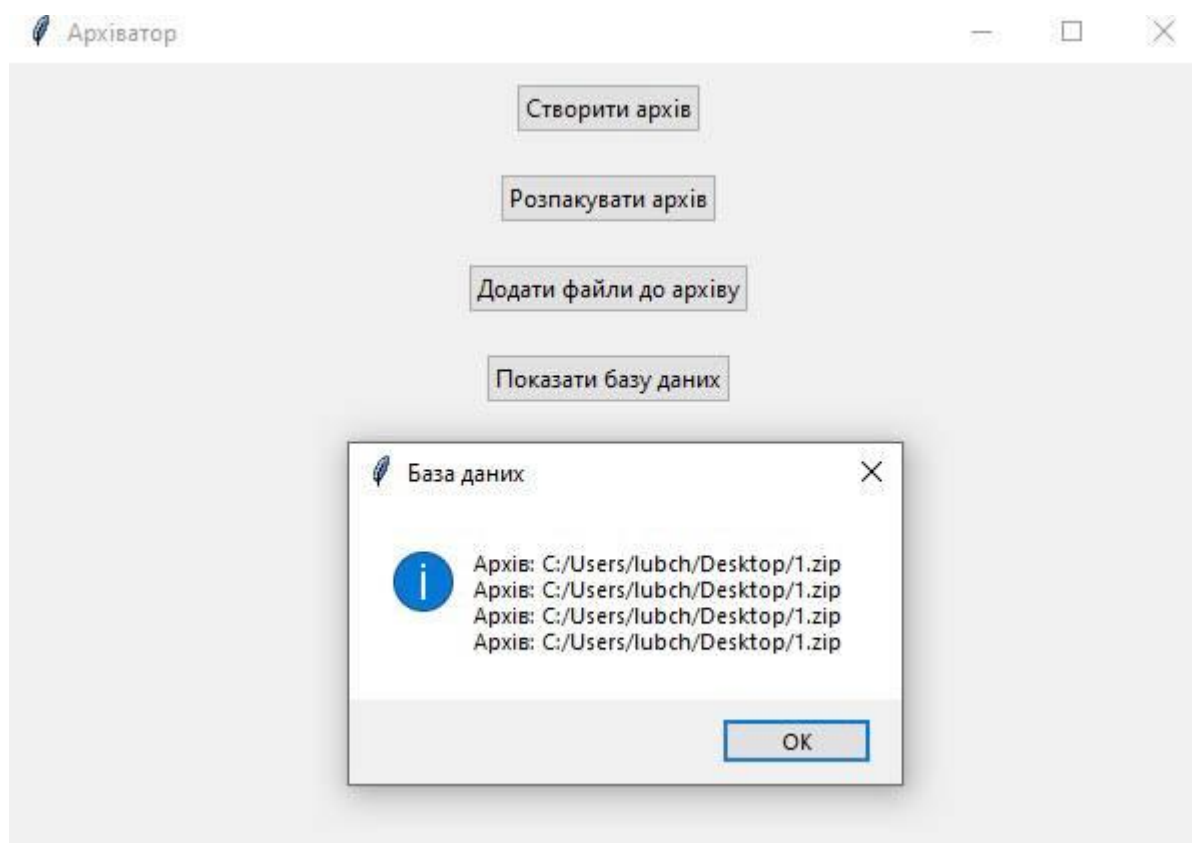


Рис 13. – Результати виклику функції: “Показати базу даних”

**Висновок:** У цій лабораторній роботі я зосередився на використанні шаблону «Composite» для створення ефективної структури для архівування файлів. Завдяки цьому шаблону я організував файли в деревоподібну структуру, де кожен файл або набір файлів може оброблятися однаково, що значно спрощує роботу з архівами.

Я також інтегрували роботу з базою даних для збереження інформації про створені архіви та файли в них. Це дозволило мені легко відстежувати дії користувача і забезпечує прозорість у роботі з архівами.

Загалом, використання шаблону «Composite» забезпечило мені гнучкість і зручність у роботі з файлами та архівами, полегшило підтримку і розширення системи. Інтеграція з базою даних додала додатковий рівень функціональності, що підвищило надійність і ефективність мого архіватора. Це дозволило створити модульну і добре організовану систему, яка може легко масштабуватися і адаптуватися до нових вимог