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

Технології розроблення програмного забезпечення

«*ШАБЛОНИ «COMPOSITE», «FLYWEIGHT», «INTERPRETER», «VISITOR»* »

Варіант 29

Виконав:
студент групи ІА-13
Хілько І.А.

Перевірив:
Мягкий М. Ю.

Київ 2023

Тема: Шаблони Composite, Flyweight, Interpreter, Visitor.

Варіант:
Система для колективних покупок. (State, Chain of responsibility, Abstract factory, Mediator, **Composite**, Client-server).

Система дозволяє створити список групи для колективної покупки, список що потрібно купити з орієнтовною вартістю кожної позиції та орієнтовною загальною вартістю, запланувати хто що буде купляти. Щоб користувач міг відмітити що він купив, за яку суму, з можливістю прикріпити чек. Система дозволяє користувачу вести списки бажаних для нього покупок, з можливістю позначати списки, які будуть доступні для друзів (як списки, що можна подарувати користувачеві). Система дозволяє добавляти інших користувачів в друзі.

Хід роботи:
1. Реалізувати не менше 3-х класів та шаблон Composite.

Composite – це структурний патерн проектування, що дає змогу згрупувати декілька об’єктів у деревоподібну структуру, а потім працювати з нею так, ніби це одиничний об’єкт.

Паттерн Composite використовується для створення складного об'єкта CompositeDatabaseFactory, який є комбінацією різних фабрик баз даних (AbstractDatabaseFactory). Цей паттерн дозволяє об'єднати об'єкти в деревоподібну структуру і використовувати їх як єдиний об'єкт.

Component (AbstractDatabaseFactory):
Абстрактний клас, який визначає інтерфейс для всіх конкретних об'єктів та їхніх складних об'єктів.
Містить абстрактні методи create_connection та create_cursor, які будуть реалізовані в конкретних класах.

```
class AbstractDatabaseFactory(abc.ABC):
    @abc.abstractmethod
    def create_connection(self):
        pass
    @abc.abstractmethod
    def create_cursor(self, connection):
        pass
```

Leaf (SQLiteDatabaseFactory, PostgreSQLDatabaseFactory): Конкретні класи, які реалізовують методи create_connection та create_cursor.

```
class SQLiteDatabaseFactory(AbstractDatabaseFactory):
    def create_connection(self, database_name='database.db'):
        return sqlite3.connect(database_name)
    def create_cursor(self, connection):
        return connection.cursor()
```

```
class PostgreSQLDatabaseFactory(AbstractDatabaseFactory):
    def create_connection(self, database_name='database', user='user',
password='password', host='localhost', port=5432,
                          psycopg2=None):
        return psycopg2.connect(database=database_name, user=user, password=password,
host=host, port=port)
    def create_cursor(self, connection):
        return connection.cursor()
```

Composite (CompositeDatabaseFactory): Складний клас, який містить список інших об'єктів (фабрик баз даних) та реалізує ті ж самі методи, що і Component.

```
class CompositeDatabaseFactory(AbstractDatabaseFactory):
    def __init__(self):
        self.factories = []
    def add_factory(self, factory):
        self.factories.append(factory)
    def create_connection(self, *args, **kwargs):
        connections = [factory.create_connection(*args, **kwargs) for factory in
self.factories]
        return connections
    def create_cursor(self, connections):
        cursors = [factory.create_cursor(connection) for factory, connection in
zip(self.factories, connections)]
        return cursors
```

Методи create_connection та create_cursor об'єднують результати викликів відповідних методів у всіх компонентах.

Висновок: Отже, під час виконання лабораторної роботи, я реалізував не менше 3-х класів та шаблон Composite.