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

Технології розроблення програмного забезпечення

«*ШАБЛОНИ «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD»*»

Варіант 29

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

Перевірив:
Мягкий М. Ю.

Київ 2023

Тема: Шаблони Mediator, Facade, Bridge, Template method.

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 Mediator.

Mediator – це поведінковий патерн проектування, що дає змогу зменшити зв’язаність великої кількості класів між собою, завдяки переміщенню цих зв’язків до одного класу-посередника.

DatabaseManager використовується як посередник між взаємодією серверу та базою даних.

Метод __init__ приймає параметр database_factory, який є фабрикою бази даних. Фабрика відповідає за створення об'єктів, пов'язаних із з'єднанням і курсором.

self.database_factory = database_factory - зберігає об'єкт фабрики бази даних для подальшого використання.

self.conn = None та self.cursor = None - ініціалізує змінні conn та cursor як None, оскільки на момент створення об'єкта з'єднання та курсора ще не встановлено.

__enter__ - цей метод викликається при вході в блок with DatabaseManager(...) as cursor. Внутрішній код цього методу встановлює нове з'єднання з базою даних та повертає об'єкт курсора для використання у блоку with.

self.conn = self.database_factory.create_connection() - викликає фабричний метод для створення нового з'єднання з базою даних та зберігає його у змінній conn.

self.cursor = self.database_factory.create_cursor(self.conn) - викликає фабричний метод для створення нового курсора, пов'язаного зі з'єднанням, та зберігає його у змінній cursor.
return self.cursor - повертає створений курсор для використання в блоку with.

__exit__ - цей метод викликається при виході з блоку with, навіть якщо сталася помилка. Внутрішній код цього методу виконує закриття з'єднання з базою даних, а також фіксує зміни, якщо вони були.

if self.conn - перевіряє, чи з'єднання було успішно створено.

self.conn.commit() - фіксує всі зміни, які були зроблені в базі даних.

self.conn.close() - закриває з'єднання з базою даних.

```
class DatabaseManager:
 def __init__(self, database_factory):
     self.database_factory = database_factory
     self.conn = None
     self.cursor = None

 def __enter__(self):
     self.conn = self.database_factory.create_connection()
     self.cursor = self.database_factory.create_cursor(self.conn)
     return self.cursor

 def __exit__(self, exc_type, exc_value, traceback):
     if self.conn:
         self.conn.commit()
         self.conn.close()
```

Цей клас ми використовуємо майже у всіх методах серверу, наприклад:

```
def list_wishlists(self, client_socket, user, args):
      try:
          with DatabaseManager(SQLiteDatabaseFactory()) as cursor:
```

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