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

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

Варіант 1

Виконав
студент групи ІА – 13:
Вознюк Максим

Перевірив:
Мягкий М. Ю

Завдання:

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.

Варіант:

..1 Музичний програвач (iterator, command, memento, facade, visitor, client-server)

Музичний програвач становить собою програму для програвання музичних файлів або відтворення потокової музики з можливістю створення, запам'ятовування і редагування списків програвання, перемішування/повторення (shuffle/repeat), розпізнавання різних аудіо-форматів, еквалайзер.

Хід роботи

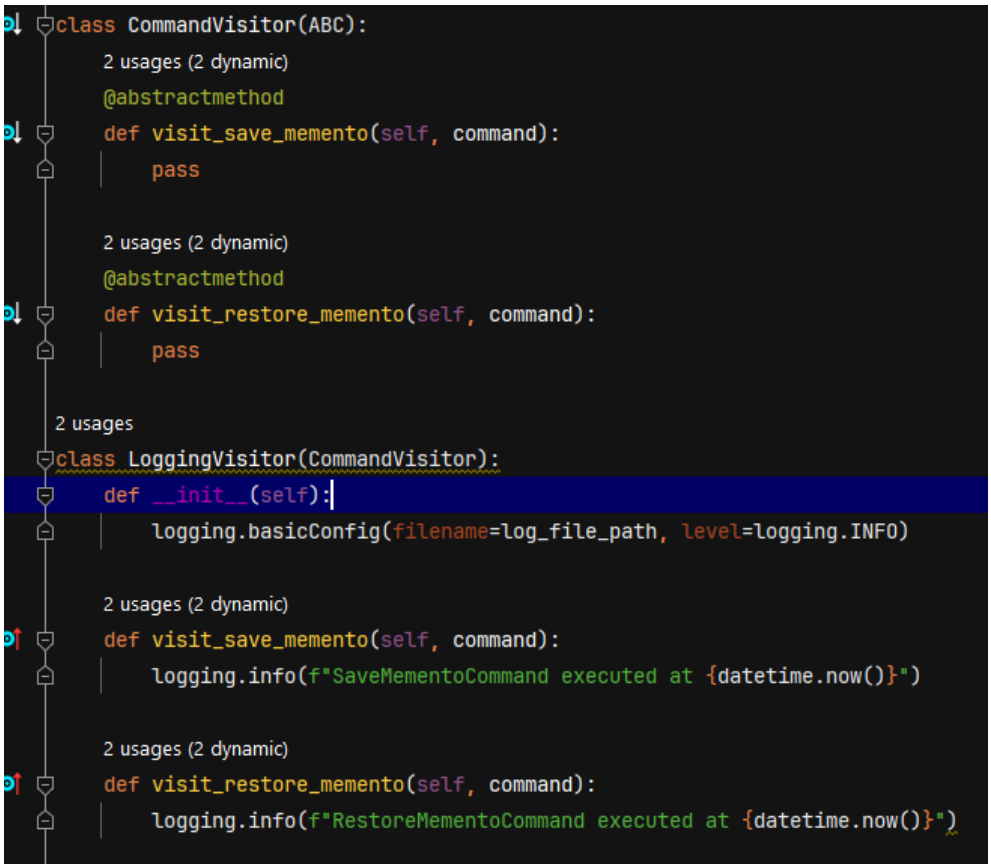
Паттерн Відвідувач(Visitor)

Відвідувач — це поведінковий патерн проектування, що дає змогу додавати до програми нові операції, не змінюючи класи об'єктів, над якими ці операції можуть виконуватися.

Клас CommandVisitor є абстрактним класом, який визначає методи для відвідування різних команд (visit_save_memento і visit_restore_memento). Підклас LoggingVisitor реалізує ці методи для логування виконаних команд.

LoggingVisitor використовується для логування виконаних команд SaveMementoCommand і RestoreMementoCommand. Коли команда виконується в методі handle_command, спочатку вона виконується за звичайним шляхом, а потім до неї застосовується метод accept з об'єктом LoggingVisitor, який викликає відповідний метод відвідувача для логування.

Це дозволяє додавати нові види операцій над командами, не змінюючи самі класи команд.



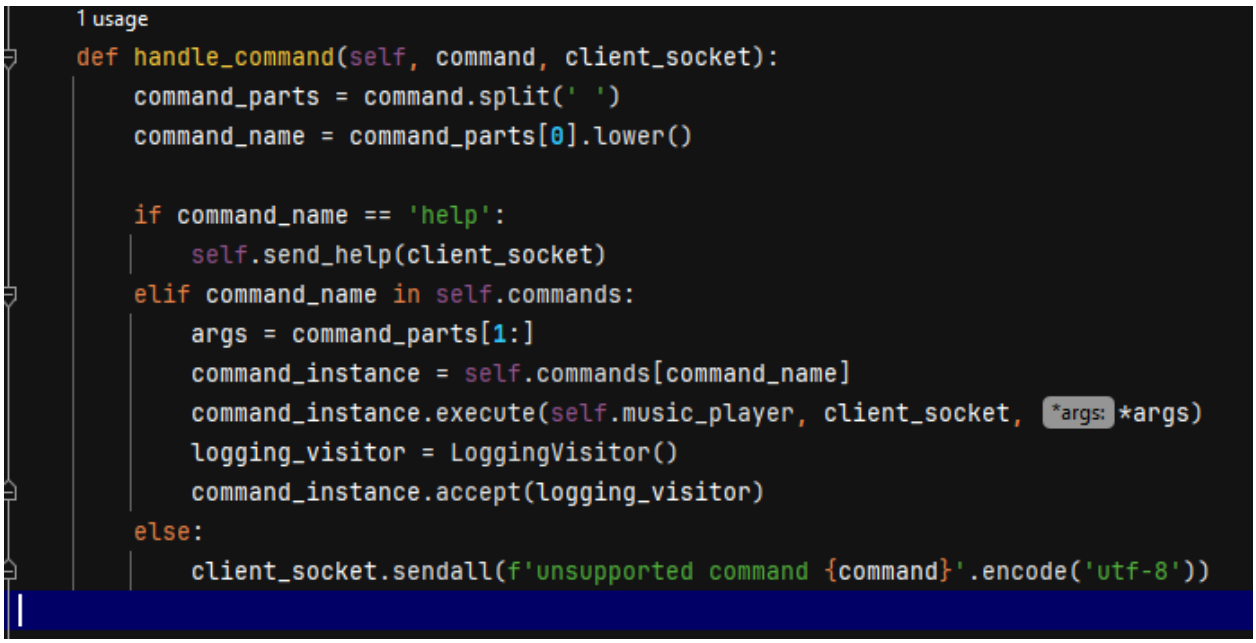
```
class CommandVisitor(ABC):
    2 usages (2 dynamic)
    @abstractmethod
    def visit_save_memento(self, command):
        pass

    2 usages (2 dynamic)
    @abstractmethod
    def visit_restore_memento(self, command):
        pass

2 usages
class LoggingVisitor(CommandVisitor):
    def __init__(self):
        logging.basicConfig(filename=log_file_path, level=logging.INFO)

    2 usages (2 dynamic)
    def visit_save_memento(self, command):
        logging.info(f"SaveMementoCommand executed at {datetime.now()}")

    2 usages (2 dynamic)
    def visit_restore_memento(self, command):
        logging.info(f"RestoreMementoCommand executed at {datetime.now()}")
```



```
    1 usage
    def handle_command(self, command, client_socket):
        command_parts = command.split(' ')
        command_name = command_parts[0].lower()

        if command_name == 'help':
            self.send_help(client_socket)
        elif command_name in self.commands:
            args = command_parts[1:]
            command_instance = self.commands[command_name]
            command_instance.execute(self.music_player, client_socket, *args: *args)
            logging_visitor = LoggingVisitor()
            command_instance.accept(logging_visitor)
        else:
            client_socket.sendall(f'unsupported command {command}'.encode('utf-8'))
```