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 №9

РІЗНІ ВИДИ ВЗАЄМОДІЇ ДОДАТКІВ:
CLIENT-SERVER, PEER-TO-PEER,
SERVICE-ORIENTED ARCHITECTURE

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. Реалізувати взаємодію програми в одній з архітектур відповідно до обраної теми.

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

Кліент-серверна архітектура використовується для забезпечення взаємодії між клієнтом та сервером за допомогою сокетів. Основна ідея полягає в тому, щоб клієнт і сервер могли обмінюватися повідомленнями через мережу.

Сервер:
Клас MusicServer:

Ініціалізує сервер із зазначеними хостом та портом, створює сокет, прив'язує його до вказаної адреси та починає прослуховування вхідних з'єднань.

Ініціалізує екземпляр MusicPlayer.

Обробка клієнта:

Метод handle_client працює в циклі, безперервно отримуючи команди від підключеного клієнта і викликаючи метод handle_command для їх обробки.

Обробка команд:

Метод handle_command розбирає отриману команду, визначає ім'я команди та її аргументи, і виконує відповідний екземпляр команди.

Команди зберігаються у словнику (commands атрибут класу MusicServer), де ключ - це ім'я команди, а значення - екземпляр відповідного класу команди.

Реєстрація команд:

Метод register_commands ініціалізує екземпляри різних класів команд і зберігає їх у словнику commands.

Запуск сервера:

Метод start запускає сервер, прослуховуючи вхідні з'єднання і створюючи новий потік (handle_client) для кожного підключеного клієнта.

Клієнт:

Встановлюється з'єднання з сервером за допомогою сокету і потім входить в цикл, де користувач може вводити команди.

Введена команда надсилається серверу, і клієнт отримує та друкує відповідь сервера.

Загальний порядок дій:

Сервер працює безперервно, чекаючи на вхідні з'єднання.

Кожен підключений клієнт обробляється в окремому потоці (handle_client).

Клієнт взаємодіє з сервером, відправляючи команди, і сервер обробляє та відповідає на ці команди.

server.py

```python
1 usage
class MusicServer:
    def __init__(self, host='127.0.0.1', port=12345):
        self.host = host
        self.port = port
        self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server_socket.bind((self.host, self.port))
        self.server_socket.listen(5)
        self.memento_stack = []
        self.music_player = MusicPlayer()


    1 usage
    def handle_client(self, client_socket):
        while True:
            try:
                data = client_socket.recv(1024).decode('utf-8')
                if not data:
                    break
                self.handle_command(data, client_socket)

            except Exception as e:
                print(f"Error handling client: {e}")
                break

        client_socket.close()
```

```python
1 usage
def send_help(self, client_socket):
    help_message = """
    Available commands:
    - play: Play the current playlist.
    - pause: Pause the playback.
    - add_playlist [name]: Create a new playlist.
    - add_track_to_playlist [playlist_name] [track_title] [track_path]: Add a track to a playlist.
    - remove_track_from_playlist [playlist_name] [track_title]: Remove a track from a playlist.
    - shuffle_playlist [playlist_name]: Shuffle the tracks in a playlist.
    - show_playlists: Show all playlists.
    - show_tracks_for_playlist [playlist_name]: Show tracks for a specific playlist.
    - stop: Stop the music playback.
    - show_tracks_with_order [playlist_name]: Show tracks for a playlist with their current order.
    - select_playlist [playlist_id]: Select a playlist by ID.
    - play_track [playlist_name] [track_title]: Play a specific track from a playlist.
    - play_track_loop [playlist_name] [track_title]: Loop a specific track from a playlist.
    - remove_playlist [playlist_name]: Remove a playlist.
    - unpause: Resume the playback.
    - set_equalizer [level]: Set the equalizer level.
    - save_memento: save memento
    - restore_memento: restore memento
    """
    client_socket.sendall(help_message.encode('utf-8'))
```

```
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'))



1 usage
def start(self):
    print(f"Server listening on {self.host}:{self.port}")
    while True:
        client_socket, addr = self.server_socket.accept()
        client_handler = threading.Thread(target=self.handle_client, args=(client_socket,))
        client_handler.start()
```

```
1 usage
def register_commands(self):
    self.commands = {
        'play': PlayCommand(),
        'pause': PauseCommand(),
        'add_playlist': AddPlaylistCommand(),
        'add_track_to_playlist': AddTrackToPlaylistCommand(),
        'remove_track_from_playlist': RemoveTrackFromPlaylistCommand(),
        'shuffle_playlist': ShufflePlaylistCommand(),
        'show_playlists': ShowPlaylistsCommand(),
        'show_tracks_for_playlist': ShowTracksForPlaylistCommand(),
        'stop': StopCommand(),
        'show_tracks_with_order': ShowTracksWithOrderCommand(),
        'select_playlist': SelectPlaylistCommand(),
        'play_track': PlayTrackCommand(),
        'play_playlist_loop': PlayPlaylistLoopCommand(),
        'play_track_loop': PlayTrackLoopCommand(),
        'remove_playlist': RemovePlaylistCommand(),
        'unpause': UnpauseCommand(),
        'set_equalizer': SetEqualizerCommand(),
        'save_memento': SaveMementoCommand(),
        'restore_memento': RestoreMementoCommand(),
    }



2 usages
def main():
    music_server = MusicServer()
    music_server.register_commands()
    music_server.start()
```

client.py

```
1 usage
def main():
    print("Welcome to the Music Player CLI. Type 'help' to see available commands.")
    host = "localhost"
    port = 12345

    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
        client_socket.connect((host, port))

        while True:
            command = input(">> ")
            client_socket.sendall(command.encode())
            response = client_socket.recv(4096).decode()
            print(response)

if __name__ == '__main__':
    main()
```