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 №5

ШАБЛОНИ «ADAPTER», «BUILDER», «COMMAND», «CHAIN OF RESPONSIBILITY», «PROTOTYPE»

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 та 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

Команда — це поведінковий патерн проектування, який перетворює запити на об'єкти, дозволяючи передавати їх як аргументи під час виклику методів, ставити запити в чергу, логувати їх, а також підтримувати скасування операцій.

Паттерн Command використовується тут для інкапсуляції логіки виконання команд та розділення її від клієнтського коду, що дозволяє легко розширювати функціонал системи та підтримувати нові команди без зміни основного коду.

Клас Command:

Це абстрактний базовий клас для всіх конкретних команд.

Має два методи: execute, який визначає, як команда повинна виконувати певну дію, та accept, який приймає відвідувача.

Конкретні реалізації команд:

Кожен клас-команда (наприклад, SaveMementoCommand, PlayCommand, і т. д.) реалізує метод execute, який викликає відповідний метод об'єкта MusicPlayer.

Метод register_commands:

Реєструє всі конкретні команди в об'єкті MusicServer, що дозволяє динамічно визначати, яку команду викликати.

Метод handle_command:

Розділяє команду на частини та викликає відповідну команду, якщо вона зареєстрована.

Використання паттерну у методі handle_command:

Команда обробляється залежно від її імені, і викликається відповідний метод конкретного об'єкта команди.

```python
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(),
    }
```

```python
19 usages
class Command(ABC):
    @abstractmethod
    def execute(self, *args):
        pass
    1 usage
    def accept(self, *args): pass

2 usages
class SaveMementoCommand(Command):
    def execute(self, music_player, client_socket, *args):
        memento = music_player.create_playlist_memento()
        music_player.memento_stack.append(memento)
        client_socket.sendall(f'Memento saved for playlist: {music_player.current_playlist_id}'.encode('utf-8'))

    def accept(self, visitor):
        visitor.visit_save_memento(self)
2 usages
class RestoreMementoCommand(Command):
    def execute(self, music_player, client_socket, *args):
        if music_player.memento_stack:
            memento = music_player.memento_stack.pop()
            music_player.restore_playlist_from_memento(memento)
            client_socket.sendall(f'Playlist restored from memento: {music_player.current_playlist_id}'.encode('utf-8'))
        else:
            client_socket.sendall('No mementos available for restoration.'.encode('utf-8'))

    def accept(self, visitor):
        visitor.visit_restore_memento(self)
```

```python
2 usages
class PlayCommand(Command):
    1 usage
    def execute(self, music_player, client_socket, *args):
        music_player.play(client_socket)

2 usages
class PauseCommand(Command):
    1 usage
    def execute(self, music_player, client_socket, *args):
        music_player.pause(client_socket)

2 usages
class AddPlaylistCommand(Command):
    1 usage
    def execute(self, music_player, client_socket, *args):
        if args:
            playlist_name = ' '.join(args)
            music_player.add_playlist(playlist_name, client_socket)

2 usages
class AddTrackToPlaylistCommand(Command):
    1 usage
    def execute(self, music_player, client_socket, *args):
        if len(args) == 3:
            playlist_name, track_title, track_path = args

            music_player.add_track_to_playlist(playlist_name, track_title, track_path, client_socket)

2 usages
class RemoveTrackFromPlaylistCommand(Command):
    1 usage
    def execute(self, music_player, client_socket, *args):
        if len(args) == 2:
            playlist_name, track_title = args
            music_player.remove_track_from_playlist(playlist_name, track_title, client_socket)

```

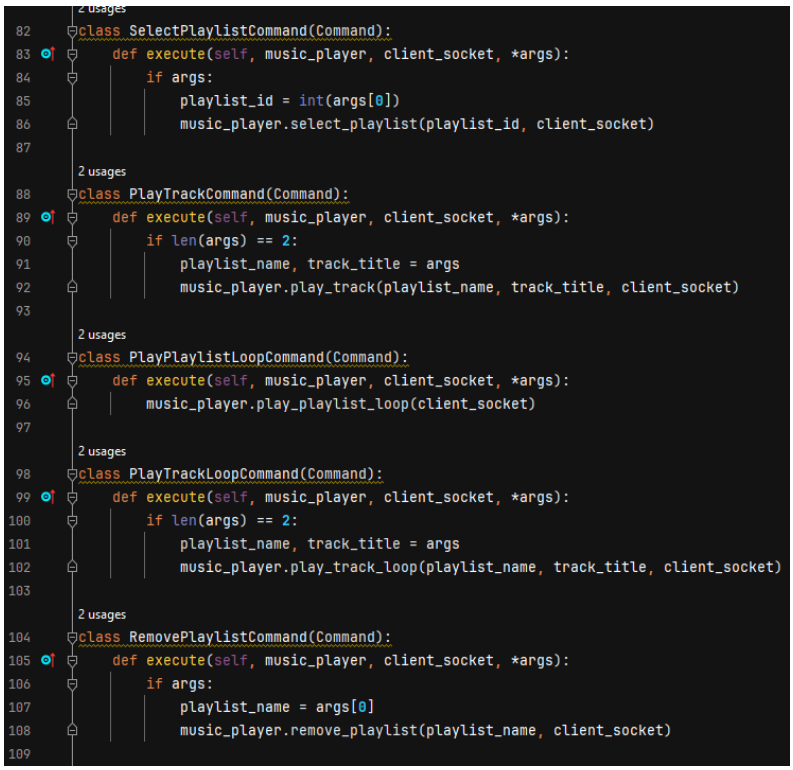
```python
class ShufflePlaylistCommand(Command):
    1 usage
    def execute(self, music_player, client_socket, *args):
        if args:
            playlist_name = args[0]
            music_player.shuffle_playlist(playlist_name, client_socket)

2 usages
class ShowPlaylistsCommand(Command):
    1 usage
    def execute(self, music_player, client_socket, *args):
        music_player.show_playlists(client_socket)

2 usages
class ShowTracksForPlaylistCommand(Command):
    1 usage
    def execute(self, music_player, client_socket, *args):
        if args:
            playlist_name = args[0]
            music_player.show_tracks_for_playlist(playlist_name, client_socket)

2 usages
class StopCommand(Command):
    1 usage
    def execute(self, music_player, client_socket, *args):
        music_player.stop(client_socket)

2 usages
class ShowTracksWithOrderCommand(Command):
    1 usage
    def execute(self, music_player, client_socket, *args):
        if args:
            playlist_name = args[0]
            music_player.show_tracks_with_order(playlist_name, client_socket)
```



```
2 usages
class SelectPlaylistCommand(Command):
    def execute(self, music_player, client_socket, *args):
        if args:
            playlist_id = int(args[0])
            music_player.select_playlist(playlist_id, client_socket)

2 usages
class PlayTrackCommand(Command):
    def execute(self, music_player, client_socket, *args):
        if len(args) == 2:
            playlist_name, track_title = args
            music_player.play_track(playlist_name, track_title, client_socket)

2 usages
class PlayPlaylistLoopCommand(Command):
    def execute(self, music_player, client_socket, *args):
        music_player.play_playlist_loop(client_socket)

2 usages
class PlayTrackLoopCommand(Command):
    def execute(self, music_player, client_socket, *args):
        if len(args) == 2:
            playlist_name, track_title = args
            music_player.play_track_loop(playlist_name, track_title, client_socket)

2 usages
class RemovePlaylistCommand(Command):
    def execute(self, music_player, client_socket, *args):
        if args:
            playlist_name = args[0]
            music_player.remove_playlist(playlist_name, client_socket)
```

```
2 usages
class UnpauseCommand(Command):
    def execute(self, music_player, client_socket, *args):
        music_player.unpause(client_socket)

2 usages
class SetEqualizerCommand(Command):
    def execute(self, music_player, client_socket, *args):
        if args:
            level = float(args[0])
            music_player.set_equalizer(level, client_socket)
```