



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 assert, 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.

```
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)
        logging_visitor = LoggingVisitor()
        command_instance.accept(logging_visitor)
    else:
        client_socket.sendall(f'unsupported command {command}'.encode('utf-8'))
```

```

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(),
    }
}

```

```

19 usages
4 class Command(ABC):
5     @abstractmethod
6     def execute(self, *args):
7         pass
8     def accept(self, *args): pass
9
10 2 usages
11 class SaveMementoCommand(Command):
12     def execute(self, music_player, client_socket, *args):
13         memento = music_player.create_playlist_memento()
14         music_player.memento_stack.append(memento)
15         client_socket.sendall(f'Memento saved for playlist: {music_player.current_playlist_id}'.encode('utf-8'))
16
17     def accept(self, visitor):
18         visitor.visit_save_memento(self)
19
20 2 usages
21 class RestoreMementoCommand(Command):
22     def execute(self, music_player, client_socket, *args):
23         if music_player.memento_stack:
24             memento = music_player.memento_stack.pop()
25             music_player.restore_playlist_from_memento(memento)
26             client_socket.sendall(f'Playlist restored from memento: {music_player.current_playlist_id}'.encode('utf-8'))
27         else:
28             client_socket.sendall('No mementos available for restoration.'.encode('utf-8'))
29
30     def accept(self, visitor):
31         visitor.visit_restore_memento(self)

```

```

29 class PlayCommand(Command):
30     1 usage
31     def execute(self, music_player, client_socket, *args):
32         music_player.play(client_socket)
33
34 class PauseCommand(Command):
35     1 usage
36     def execute(self, music_player, client_socket, *args):
37         music_player.pause(client_socket)
38
39 class AddPlaylistCommand(Command):
40     1 usage
41     def execute(self, music_player, client_socket, *args):
42         if args:
43             playlist_name = ' '.join(args)
44             music_player.add_playlist(playlist_name, client_socket)
45
46 class AddTrackToPlaylistCommand(Command):
47     1 usage
48     def execute(self, music_player, client_socket, *args):
49         if len(args) == 3:
50             playlist_name, track_title, track_path = args
51             music_player.add_track_to_playlist(playlist_name, track_title, track_path, client_socket)
52
53 class RemoveTrackFromPlaylistCommand(Command):
54     1 usage
55     def execute(self, music_player, client_socket, *args):
56         if len(args) == 2:
57             playlist_name, track_title = args
58             music_player.remove_track_from_playlist(playlist_name, track_title, client_socket)

```

```

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)

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

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)

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

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
82 class SelectPlaylistCommand(Command):
83     def execute(self, music_player, client_socket, *args):
84         if args:
85             playlist_id = int(args[0])
86             music_player.select_playlist(playlist_id, client_socket)
87
2 usages
88 class PlayTrackCommand(Command):
89     def execute(self, music_player, client_socket, *args):
90         if len(args) == 2:
91             playlist_name, track_title = args
92             music_player.play_track(playlist_name, track_title, client_socket)
93
2 usages
94 class PlayPlaylistLoopCommand(Command):
95     def execute(self, music_player, client_socket, *args):
96         music_player.play_playlist_loop(client_socket)
97
2 usages
98 class PlayTrackLoopCommand(Command):
99     def execute(self, music_player, client_socket, *args):
100         if len(args) == 2:
101             playlist_name, track_title = args
102             music_player.play_track_loop(playlist_name, track_title, client_socket)
103
2 usages
104 class RemovePlaylistCommand(Command):
105     def execute(self, music_player, client_socket, *args):
106         if args:
107             playlist_name = args[0]
108             music_player.remove_playlist(playlist_name, client_socket)
109

```

```

2 usages
110 class UnpauseCommand(Command):
111     def execute(self, music_player, client_socket, *args):
112         music_player.unpause(client_socket)
113
2 usages
114 class SetEqualizerCommand(Command):
115     def execute(self, music_player, client_socket, *args):
116         if args:
117             level = float(args[0])
118             music_player.set_equalizer(level, client_socket)
119

```