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 №6

ШАБЛОНИ «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»

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

Знімок — це поведінковий патерн проектування, що дає змогу зберігати та відновлювати минулий стан об'єктів, не розкриваючи подробиць їхньої реалізації.

У нашому випадку використовується для збереження стану плейліста в музичному плеєрі.

PlaylistMemento: Цей клас представляє момент (або стан) плейлиста, який може бути збережений і відновлений.

SaveMementoCommand: Цей клас є командою, яка викликається для збереження моменту (стану) плейлиста. Він створює об'єкт PlaylistMemento і додає його до

стеку memento_stack в music_player. Також відправляє повідомлення клієнту про збереження.

RestoreMementoCommand: Цей клас є командою для відновлення попереднього стану плейлиста. Він витягує останній збережений і викликає restore_playlist_from_memento у music_player для відновлення стану. Якщо стек порожній, відправляється повідомлення про відсутність доступних моментів для відновлення

```python
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 (2 dynamic)
    def create_playlist_memento(self):
        if self.current_playlist_id is not None:
            tracks = self.db_manager.get_tracks_for_playlist(self.current_playlist_id)
            return PlaylistMemento(self.current_playlist_id, tracks)
        return None

    2 usages (2 dynamic)
    def restore_playlist_from_memento(self, playlist_memento):
        client_socket = None
        if playlist_memento:
            self.current_playlist_id = playlist_memento.playlist_id
            self.db_manager.remove_track_from_playlist(self.current_playlist_id, track_title: "", client_socket, value = 1)
            for position, (title, path) in enumerate(playlist_memento.tracks):
                self.db_manager.add_track_to_playlist(self.current_playlist_id, title, path, client_socket, value = 1)
```