



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 №4
**ШАБЛОНИ «SINGLETON»,
«ITERATOR», «PROXY», «STATE»,
«STRATEGY»**

Виконав
студент групи ІА – 13:
Рябушко Егор

Перевірів:

Завдання:

1. Ознайомитися з короткими теоретичними відомостями.
2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3. Застосування одного з розглянутих шаблонів при реалізації програми

Хід роботи

Ітератор — це поведінковий патерн проектування, що дає змогу послідовно обходити елементи складових об'єктів, не розкриваючи їхньої внутрішньої організації.

Використовується паттерн для ітерації по треках у плейлістах.

```
1 usage
class PlaylistIterator:
    def __init__(self, playlist_tracks):
        self.playlist_tracks = playlist_tracks
        self.current_track_index = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.current_track_index < len(self.playlist_tracks):
            current_track = self.playlist_tracks[self.current_track_index]
            self.current_track_index += 1
            return current_track
        else:
            raise StopIteration
```

`__init__(self, playlist_tracks)`: Ініціалізує ітератор зі списку треків плейлиста та встановлює індекс поточного треку на початок.

`__iter__(self)`: Повертає сам об'єкт ітератора. Це метод, який дозволяє об'єктам бути ітерованими.

`__next__(self)`: Повертає наступний трек у списку. Якщо кінець списку досягнутий, викликає `StopIteration`, що сигналізує про закінчення ітерації.

Використання ітератора:

```
class Playlist:
    def __init__(self, name):
        self.name = name

    2 usages (2 dynamic)
    def get_tracks(self, playlist_manager):
        playlist_tracks = playlist_manager.get_tracks_for_playlist(self.name)
        return PlaylistIterator(playlist_tracks)
```

У методі `get_tracks` класу `Playlist` створюється новий об'єкт `PlaylistIterator` на основі треків, отриманих з бази даних.

Використання ітератора у методі `play_playlist`:

```
1 usage
def play_playlist(self, client_socket, playlist_adapter):
    playlist_iterator = playlist_adapter.get_tracks(self.current_playlist)

    def send_audio():
        for track_path in playlist_iterator:
```

У методі `play_playlist` екземпляр `PlaylistIterator` використовується для ітерації по треках поточного плейлиста під час відтворення аудіо.

Висновок:

Я оволодів навичками проектування паттерну `Iterator`. Також реалізував цей паттерн у проекті.