Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии

ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №10
дисциплины «Объектно-ориентированное программирование»
Вариант 7

Выполнил:
Мотовилов Вадим Борисович
3 курс, группа ИВТ-б-о-23-2,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной техники и
автоматизированных систем», очная
форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р.А. ктн доцент
департамента перспективной
инженерии

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2026 г.

Тема: разработка приложений с интерфейсом командной строки в языке Python.

Цель работы: приобретение построения приложений с интерфейсом командной строки с помощью языка программирования Python версии 3.x.

Порядок выполнения работы:

Задание 1.

> Датакласс: название трека, исполнитель, длительность. CLI: добавить, вывод, выбор треков короче заданного времени, сортировка по длительности, сохранение в JSON.

Код программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import argparse
import json
import sys
from dataclasses import dataclass, field, asdict
from datetime import timedelta
from typing import List, Optional
from enum import Enum

class MusicGenre(Enum):
    """Музыкальные жанры"""

    POP = "Поп"
    ROCK = "Рок"
    JAZZ = "Джаз"
    HIP_HOP = "Хип-хоп"
    ELECTRONIC = "Электронная"
    CLASSICAL = "Классическая"
    COUNTRY = "Кантри"
    RNB = "R&B"
    METAL = "Метал"
    INDIE = "Инди"
    OTHER = "Другое"

@dataclass
class Track:
    """Датакласс для представления музыкального трека"""

    title: str
    artist: str
    duration: timedelta  # Длительность в формате timedelta
    genre: MusicGenre = MusicGenre.OTHER
    year: Optional[int] = None

    def __post_init__(self):
```

```
    """Валидация данных после инициализации"""
    if not self.title.strip():
        raise ValueError("Название трека не может быть пустым")
    if not self.artist.strip():
        raise ValueError("Имя исполнителя не может быть пустым")
    if self.duration.total_seconds() <= 0:
        raise ValueError("Длительность должна быть положительной")
    if self.year is not None and (self.year < 1900 or self.year > 2100):
        raise ValueError("Некорректный год")

@property
def duration_str(self) -> str:
    """Возвращает длительность в формате MM:SS"""
    total_seconds = int(self.duration.total_seconds())
    minutes = total_seconds // 60
    seconds = total_seconds % 60
    return f"{minutes:02d}:{seconds:02d}"

@property
def duration_seconds(self) -> int:
    """Возвращает длительность в секундах"""
    return int(self.duration.total_seconds())

def to_dict(self) -> dict:
    """Сериализация в словарь для JSON"""
    return {
        "title": self.title,
        "artist": self.artist,
        "duration": self.duration_str,
        "genre": self.genre.value,
        "year": self.year,
    }

@classmethod
def from_dict(cls, data: dict) -> "Track":
    """Десериализация из словаря"""
    # Преобразуем строку длительности в timedelta
    duration_parts = data["duration"].split(":")
    if len(duration_parts) == 2:
        minutes, seconds = map(int, duration_parts)
        duration = timedelta(minutes=minutes, seconds=seconds)
    elif len(duration_parts) == 3:
        hours, minutes, seconds = map(int, duration_parts)
        duration = timedelta(hours=hours, minutes=minutes, seconds=seconds)
    else:
        raise ValueError(f"Неверный формат длительности: {data['duration']}")

    # Находим жанр по значению
    genre_value = data.get("genre", "Другое")
    genre = MusicGenre.OTHER
    for g in MusicGenre:
        if g.value == genre_value:
```

```
                genre = g
                break

        return cls(
            title=data["title"],
            artist=data["artist"],
            duration=duration,
            genre=genre,
            year=data.get("year"),
        )

    def __str__(self) -> str:
        """Строковое представление трека"""
        year_str = f" ({self.year})" if self.year else ""
        return (
            f"  '{self.title}' - {self.artist}{year_str} "
            f"[{self.genre.value}]  {self.duration_str}"
        )

@dataclass
class Playlist:
    """Контейнер для хранения и управления коллекцией треков"""

    name: str
    tracks: List[Track] = field(default_factory=list)

    def add_track(self, track: Track) -> None:
        """Добавить трек в плейлист"""
        self.tracks.append(track)

    def add_tracks(self, *tracks: Track) -> None:
        """Добавить несколько треков"""
        for track in tracks:
            self.add_track(track)

    def remove_track(self, index: int) -> bool:
        """Удалить трек по индексу"""
        if 0 <= index < len(self.tracks):
            del self.tracks[index]
            return True
        return False

    def get_tracks_by_artist(self, artist: str) -> List[Track]:
        """Получить треки указанного исполнителя"""
        artist_lower = artist.strip().lower()
        return [track for track in self.tracks if track.artist.lower() == artist_lower]

    def get_tracks_by_genre(self, genre: MusicGenre) -> List[Track]:
        """Получить треки указанного жанра"""
        return [track for track in self.tracks if track.genre == genre]

    def get_tracks_by_year(self, year: int) -> List[Track]:
```

```
    """Получить треки указанного года"""
    return [track for track in self.tracks if track.year == year]

def get_tracks_in_duration_range(self, min_sec: int, max_sec: int) -> List[Track]:
    """Получить треки, длительность которых в указанном диапазоне (в секундах)"""
    return [
        track
        for track in self.tracks
        if min_sec <= track.duration_seconds <= max_sec
    ]

def sort_by_duration(self, descending: bool = False) -> None:
    """Отсортировать треки по длительности"""
    self.tracks.sort(key=lambda t: t.duration_seconds, reverse=descending)

def sort_by_title(self, descending: bool = False) -> None:
    """Отсортировать треки по названию"""
    self.tracks.sort(key=lambda t: t.title.lower(), reverse=descending)

def sort_by_artist(self, descending: bool = False) -> None:
    """Отсортировать треки по исполнителю"""
    self.tracks.sort(key=lambda t: t.artist.lower(), reverse=descending)

def get_total_duration(self) -> timedelta:
    """Получить общую длительность плейлиста"""
    total_seconds = sum(track.duration_seconds for track in self.tracks)
    return timedelta(seconds=total_seconds)

def get_statistics(self) -> dict:
    """Получить статистику по плейлисту"""
    stats = {
        "total_tracks": len(self.tracks),
        "total_duration": str(self.get_total_duration()),
        "artists": len(set(track.artist.lower() for track in self.tracks)),
        "genres": {},
        "years": {},
    }

    # Статистика по жанрам
    for genre in MusicGenre:
        count = len(self.get_tracks_by_genre(genre))
        if count > 0:
            stats["genres"][genre.value] = count

    # Статистика по годам
    for track in self.tracks:
        if track.year:
            stats["years"][track.year] = stats["years"].get(track.year, 0) + 1

    return stats

def display_tracks(self, tracks: Optional[List[Track]] = None) -> None:
```

```
    """Вывести список треков"""
    if tracks is None:
        tracks = self.tracks

    if not tracks:
        print("Нет треков для отображения")
        return

    print(f"\nПлейлист '{self.name}' ({len(tracks)} треков):")
    print("=" * 80)
    for i, track in enumerate(tracks, 1):
        print(f"{i:3}. {track}")
    print("=" * 80)

def save_to_json(self, filename: str) -> bool:
    """Сохранить плейлист в JSON файл"""
    try:
        data = {
            "playlist_name": self.name,
            "tracks": [track.to_dict() for track in self.tracks],
        }

        with open(filename, "w", encoding="utf-8") as f:
            json.dump(data, f, ensure_ascii=False, indent=2)

        print(f"Плейлист сохранен в файл: {filename}")
        return True
    except Exception as e:
        print(f"Ошибка при сохранении в файл {filename}: {e}")
        return False

def load_from_json(self, filename: str) -> bool:
    """Загрузить плейлист из JSON файла"""
    try:
        with open(filename, "r", encoding="utf-8") as f:
            data = json.load(f)

        self.name = data.get("playlist_name", self.name)
        self.tracks.clear()

        for track_data in data.get("tracks", []):
            try:
                track = Track.from_dict(track_data)
                self.add_track(track)
            except Exception as e:
                print(f"Ошибка при загрузке трека: {e}")

        print(f"Плейлист загружен из файла: {filename}")
        print(f"   Загружено треков: {len(self.tracks)}")
        return True
    except FileNotFoundError:
        print(f"Файл не найден: {filename}")
```

```
            return False
        except json.JSONDecodeError:
            print(f"Ошибка формата JSON в файле: {filename}")
            return False
        except Exception as e:
            print(f"Ошибка при загрузке из файла {filename}: {e}")
            return False

    def __len__(self) -> int:
        """Количество треков в плейлисте"""
        return len(self.tracks)

class MusicManagerCLI:
    """Класс для управления CLI интерфейсом"""

    def __init__(self):
        self.playlist = Playlist("Мой плейлист")
        self.setup_parser()

    def setup_parser(self):
        """Настройка парсера аргументов командной строки"""
        self.parser = argparse.ArgumentParser(
            description="Менеджер музыкальной коллекции",
            formatter_class=argparse.RawDescriptionHelpFormatter,
            epilog="""
Примеры использования:
  %(prog)s add --title "Bohemian Rhapsody" --artist "Queen" --duration "5:55" --genre ROCK --
year 1975
  %(prog)s show --sort duration --reverse
  %(prog)s filter --artist "Queen"
  %(prog)s filter --min-duration 180 --max-duration 300
  %(prog)s save --file my_playlist.json
  %(prog)s load --file my_playlist.json
            """,
        )

        subparsers = self.parser.add_subparsers(dest="command", help="Команды")

        # Команда add: добавить трек
        parser_add = subparsers.add_parser("add", help="Добавить новый трек")
        parser_add.add_argument("--title", required=True, help="Название трека")
        parser_add.add_argument("--artist", required=True, help="Исполнитель")
        parser_add.add_argument(
            "--duration", required=True, help="Длительность (формат MM:SS или HH:MM:SS)"
        )
        parser_add.add_argument(
            "--genre",
            choices=[g.name for g in MusicGenre],
            default="OTHER",
            help="Музыкальный жанр",
        )
        parser_add.add_argument("--year", type=int, help="Год выпуска")
```

```
# Команда show: показать треки
parser_show = subparsers.add_parser("show", help="Показать все треки")
parser_show.add_argument(
    "--sort",
    choices=["duration", "title", "artist"],
    help="Критерий сортировки",
)
parser_show.add_argument(
    "--reverse", action="store_true", help="Сортировка по убыванию"
)

# Команда filter: фильтрация треков
parser_filter = subparsers.add_parser("filter", help="Фильтрация треков")
filter_group = parser_filter.add_mutually_exclusive_group(required=True)
filter_group.add_argument("--artist", help="Фильтр по исполнителю")
filter_group.add_argument(
    "--genre", choices=[g.name for g in MusicGenre], help="Фильтр по жанру"
)
filter_group.add_argument("--year", type=int, help="Фильтр по году")
filter_group.add_argument(
    "--min-duration", type=int, help="Минимальная длительность (секунды)"
)
filter_group.add_argument(
    "--max-duration", type=int, help="Максимальная длительность (секунды)"
)
parser_filter.add_argument(
    "--sort",
    choices=["duration", "title", "artist"],
    help="Критерий сортировки",
)
parser_filter.add_argument(
    "--reverse", action="store_true", help="Сортировка по убыванию"
)

# Команда stats: статистика
parser_stats = subparsers.add_parser("stats", help="Статистика плейлиста")

# Команда save: сохранить в JSON
parser_save = subparsers.add_parser("save", help="Сохранить в JSON файл")
parser_save.add_argument(
    "--file",
    default="playlist.json",
    help="Имя файла (по умолчанию: playlist.json)",
)

# Команда load: загрузить из JSON
parser_load = subparsers.add_parser("load", help="Загрузить из JSON файла")
parser_load.add_argument("--file", required=True, help="Имя файла")

# Команда demo: демонстрационные данные
subparsers.add_parser("demo", help="Добавить демонстрационные данные")
```

```
def parse_duration(self, duration_str: str) -> timedelta:
    """Парсинг строки длительности в timedelta"""
    try:
        parts = duration_str.split(":")
        if len(parts) == 2:
            minutes, seconds = map(int, parts)
            return timedelta(minutes=minutes, seconds=seconds)
        elif len(parts) == 3:
            hours, minutes, seconds = map(int, parts)
            return timedelta(hours=hours, minutes=minutes, seconds=seconds)
        else:
            # Попробуем интерпретировать как секунды
            seconds = int(duration_str)
            return timedelta(seconds=seconds)
    except ValueError:
        raise argparse.ArgumentTypeError(
            f"Неверный формат длительности: '{duration_str}'. "
            f"Используйте MM:SS или HH:MM:SS"
        )

def handle_add(self, args):
    """Обработка команды add"""
    try:
        duration = self.parse_duration(args.duration)
        genre = MusicGenre[args.genre]

        track = Track(
            title=args.title,
            artist=args.artist,
            duration=duration,
            genre=genre,
            year=args.year,
        )

        self.playlist.add_track(track)
        print(f"Трек добавлен: {track}")

    except ValueError as e:
        print(f"Ошибка при добавлении трека: {e}")
    except Exception as e:
        print(f"Неожиданная ошибка: {e}")

def handle_show(self, args):
    """Обработка команды show"""
    if not self.playlist.tracks:
        print("Плейлист пуст. Используйте команду 'add' для добавления треков.")
        return

    # Применяем сортировку если указана
    if args.sort:
        if args.sort == "duration":
```

```
            self.playlist.sort_by_duration(args.reverse)
        elif args.sort == "title":
            self.playlist.sort_by_title(args.reverse)
        elif args.sort == "artist":
            self.playlist.sort_by_artist(args.reverse)

    self.playlist.display_tracks()

def handle_filter(self, args):
    """Обработка команды filter"""
    if not self.playlist.tracks:
        print("Плейлист пуст. Нет данных для фильтрации.")
        return

    filtered_tracks = []

    if args.artist:
        filtered_tracks = self.playlist.get_tracks_by_artist(args.artist)
        print(f"Треки исполнителя '{args.artist}':")

    elif args.genre:
        genre = MusicGenre[args.genre]
        filtered_tracks = self.playlist.get_tracks_by_genre(genre)
        print(f"Треки жанра '{genre.value}':")

    elif args.year:
        filtered_tracks = self.playlist.get_tracks_by_year(args.year)
        print(f"Треки {args.year} года:")

    elif args.min_duration or args.max_duration:
        min_sec = args.min_duration or 0
        max_sec = args.max_duration or 86400  # 24 часа

        if min_sec > max_sec:
            print("Минимальная длительность не может быть больше максимальной")
            return

        filtered_tracks = self.playlist.get_tracks_in_duration_range(
            min_sec, max_sec
        )
        print(f"Треки длительностью от {min_sec} до {max_sec} секунд:")

    if not filtered_tracks:
        print("Нет треков, соответствующих критериям фильтрации")
        return

    # Применяем сортировку если указана
    if args.sort:
        if args.sort == "duration":
            filtered_tracks.sort(
                key=lambda t: t.duration_seconds, reverse=args.reverse
            )
```

```
        elif args.sort == "title":
            filtered_tracks.sort(
                key=lambda t: t.title.lower(), reverse=args.reverse
            )
        elif args.sort == "artist":
            filtered_tracks.sort(
                key=lambda t: t.artist.lower(), reverse=args.reverse
            )

    self.playlist.display_tracks(filtered_tracks)

def handle_stats(self, args):
    """Обработка команды stats"""
    if not self.playlist.tracks:
        print("Плейлист пуст. Нет данных для статистики.")
        return

    stats = self.playlist.get_statistics()

    print(f"\nСтатистика плейлиста '{self.playlist.name}':")
    print("=" * 50)
    print(f"Всего треков: {stats['total_tracks']}")
    print(f"Общая длительность: {stats['total_duration']}")
    print(f"Уникальных исполнителей: {stats['artists']}")

    if stats["genres"]:
        print("\nРаспределение по жанрам:")
        for genre, count in stats["genres"].items():
            print(f"  {genre}: {count}")

    if stats["years"]:
        print("\nРаспределение по годам:")
        for year, count in sorted(stats["years"].items()):
            print(f"  {year}: {count}")

    print("=" * 50)

def handle_save(self, args):
    """Обработка команды save"""
    if not self.playlist.tracks:
        print("Плейлист пуст. Сохранение пустого плейлиста.")

    self.playlist.save_to_json(args.file)

def handle_load(self, args):
    """Обработка команды load"""
    self.playlist.load_from_json(args.file)

def handle_demo(self, args):
    """Обработка команды demo - добавление демонстрационных данных"""
    demo_tracks = [
        Track(
```

```
    "Bohemian Rhapsody",
    "Queen",
    timedelta(minutes=5, seconds=55),
    MusicGenre.ROCK,
    1975,
),
Track(
    "Smells Like Teen Spirit",
    "Nirvana",
    timedelta(minutes=5, seconds=1),
    MusicGenre.ROCK,
    1991,
),
Track(
    "Billie Jean",
    "Michael Jackson",
    timedelta(minutes=4, seconds=54),
    MusicGenre.POP,
    1982,
),
Track(
    "Shape of You",
    "Ed Sheeran",
    timedelta(minutes=3, seconds=53),
    MusicGenre.POP,
    2017,
),
Track(
    "Take Five",
    "Dave Brubeck",
    timedelta(minutes=5, seconds=24),
    MusicGenre.JAZZ,
    1959,
),
Track(
    "Hotel California",
    "Eagles",
    timedelta(minutes=6, seconds=30),
    MusicGenre.ROCK,
    1976,
),
Track(
    "Moonlight Sonata",
    "Beethoven",
    timedelta(minutes=15, seconds=0),
    MusicGenre.CLASSICAL,
    1801,
),
Track(
    "Lose Yourself",
    "Eminem",
    timedelta(minutes=5, seconds=26),
```

```
                MusicGenre.HIP_HOP,
                2002,
            ),
            Track(
                "Stairway to Heaven",
                "Led Zeppelin",
                timedelta(minutes=8, seconds=2),
                MusicGenre.ROCK,
                1971,
            ),
            Track(
                "Imagine",
                "John Lennon",
                timedelta(minutes=3, seconds=3),
                MusicGenre.POP,
                1971,
            ),
        ]

        self.playlist.add_tracks(*demo_tracks)
        print(f"Добавлено {len(demo_tracks)} демонстрационных треков")
        print(f"Теперь в плейлисте: {len(self.playlist)} треков")

    def run(self):
        """Запуск CLI интерфейса"""
        if len(sys.argv) == 1:
            self.parser.print_help()
            return

        args = self.parser.parse_args()

        if args.command == "add":
            self.handle_add(args)
        elif args.command == "show":
            self.handle_show(args)
        elif args.command == "filter":
            self.handle_filter(args)
        elif args.command == "stats":
            self.handle_stats(args)
        elif args.command == "save":
            self.handle_save(args)
        elif args.command == "load":
            self.handle_load(args)
        elif args.command == "demo":
            self.handle_demo(args)
        else:
            self.parser.print_help()

def main():
    """Главная функция программы"""
    cli = MusicManagerCLI()
    cli.run()
```

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

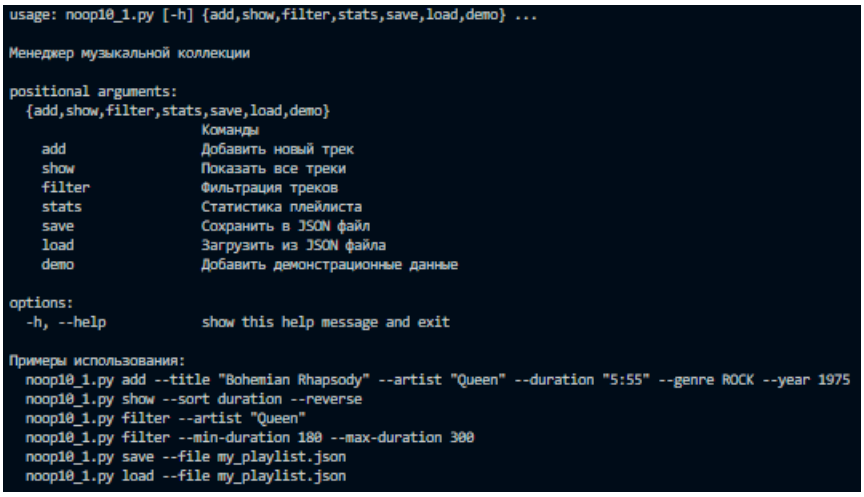

Рисунок 1. Пример работы кода

Задание 2.

Самостоятельно изучите работу с пакетом `click` для построения интерфейса командной строки (CLI). Для своего варианта задания 1 необходимо реализовать интерфейс командной строки с использованием пакета `click`.

Код программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import click
import json
import sys
from dataclasses import dataclass, field, asdict
from datetime import timedelta
from typing import List, Optional
from enum import Enum
import os

class MusicGenre(Enum):
    """Музыкальные жанры"""

    POP = "Поп"
    ROCK = "Рок"
    JAZZ = "Джаз"
    HIP_HOP = "Хип-хоп"
    ELECTRONIC = "Электронная"
    CLASSICAL = "Классическая"
    COUNTRY = "Кантри"
    RNB = "R&B"
    METAL = "Метал"
```

```
    INDIE = "Инди"
    OTHER = "Другое"

@dataclass
class Track:
    """Датакласс для представления музыкального трека"""

    title: str
    artist: str
    duration: timedelta
    genre: MusicGenre = MusicGenre.OTHER
    year: Optional[int] = None

    def __post_init__(self):
        """Валидация данных после инициализации"""
        if not self.title.strip():
            raise ValueError("Название трека не может быть пустым")
        if not self.artist.strip():
            raise ValueError("Имя исполнителя не может быть пустым")
        if self.duration.total_seconds() <= 0:
            raise ValueError("Длительность должна быть положительной")
        if self.year is not None and (self.year < 1900 or self.year > 2100):
            raise ValueError("Некорректный год")

    @property
    def duration_str(self) -> str:
        """Возвращает длительность в формате MM:SS"""
        total_seconds = int(self.duration.total_seconds())
        minutes = total_seconds // 60
        seconds = total_seconds % 60
        return f"{minutes:02d}:{seconds:02d}"

    @property
    def duration_seconds(self) -> int:
        """Возвращает длительность в секундах"""
        return int(self.duration.total_seconds())

    def to_dict(self) -> dict:
        """Сериализация в словарь для JSON"""
        return {
            "title": self.title,
            "artist": self.artist,
            "duration": self.duration_str,
            "genre": self.genre.value,
            "year": self.year,
        }

    @classmethod
    def from_dict(cls, data: dict) -> "Track":
        """Десериализация из словаря"""
        duration_parts = data["duration"].split(":")
        if len(duration_parts) == 2:
```

```
            minutes, seconds = map(int, duration_parts)
            duration = timedelta(minutes=minutes, seconds=seconds)
        elif len(duration_parts) == 3:
            hours, minutes, seconds = map(int, duration_parts)
            duration = timedelta(hours=hours, minutes=minutes, seconds=seconds)
        else:
            raise ValueError(f"Неверный формат длительности: {data['duration']}")

        genre_value = data.get("genre", "Другое")
        genre = MusicGenre.OTHER
        for g in MusicGenre:
            if g.value == genre_value:
                genre = g
                break

        return cls(
            title=data["title"],
            artist=data["artist"],
            duration=duration,
            genre=genre,
            year=data.get("year"),
        )

    def __str__(self) -> str:
        """Строковое представление трека"""
        year_str = f" ({self.year})" if self.year else ""
        return (
            f" '{self.title}' - {self.artist}{year_str} "
            f"[{self.genre.value}] ⏱ {self.duration_str}"
        )

@dataclass
class Playlist:
    """Контейнер для хранения и управления коллекцией треков"""

    name: str
    tracks: List[Track] = field(default_factory=list)

    def add_track(self, track: Track) -> None:
        """Добавить трек в плейлист"""
        self.tracks.append(track)

    def add_tracks(self, *tracks: Track) -> None:
        """Добавить несколько треков"""
        for track in tracks:
            self.add_track(track)

    def remove_track(self, index: int) -> bool:
        """Удалить трек по индексу"""
        if 0 <= index < len(self.tracks):
            del self.tracks[index]
            return True
```

```
        return False

    def get_tracks_by_artist(self, artist: str) -> List[Track]:
        """Получить треки указанного исполнителя"""
        artist_lower = artist.strip().lower()
        return [track for track in self.tracks if track.artist.lower() == artist_lower]

    def get_tracks_by_genre(self, genre: MusicGenre) -> List[Track]:
        """Получить треки указанного жанра"""
        return [track for track in self.tracks if track.genre == genre]

    def get_tracks_by_year(self, year: int) -> List[Track]:
        """Получить треки указанного года"""
        return [track for track in self.tracks if track.year == year]

    def get_tracks_in_duration_range(self, min_sec: int, max_sec: int) -> List[Track]:
        """Получить треки, длительность которых в указанном диапазоне (в секундах)"""
        return [
            track
            for track in self.tracks
            if min_sec <= track.duration_seconds <= max_sec
        ]

    def sort_by_duration(self, descending: bool = False) -> None:
        """Отсортировать треки по длительности"""
        self.tracks.sort(key=lambda t: t.duration_seconds, reverse=descending)

    def sort_by_title(self, descending: bool = False) -> None:
        """Отсортировать треки по названию"""
        self.tracks.sort(key=lambda t: t.title.lower(), reverse=descending)

    def sort_by_artist(self, descending: bool = False) -> None:
        """Отсортировать треки по исполнителю"""
        self.tracks.sort(key=lambda t: t.artist.lower(), reverse=descending)

    def get_total_duration(self) -> timedelta:
        """Получить общую длительность плейлиста"""
        total_seconds = sum(track.duration_seconds for track in self.tracks)
        return timedelta(seconds=total_seconds)

    def get_statistics(self) -> dict:
        """Получить статистику по плейлисту"""
        stats = {
            "total_tracks": len(self.tracks),
            "total_duration": str(self.get_total_duration()),
            "artists": len(set(track.artist.lower() for track in self.tracks)),
            "genres": {},
            "years": {},
        }

        for genre in MusicGenre:
            count = len(self.get_tracks_by_genre(genre))
```

```
            if count > 0:
                stats["genres"][genre.value] = count

        for track in self.tracks:
            if track.year:
                stats["years"][track.year] = stats["years"].get(track.year, 0) + 1

        return stats

    def display_tracks(self, tracks: Optional[List[Track]] = None) -> None:
        """Вывести список треков"""
        if tracks is None:
            tracks = self.tracks

        if not tracks:
            click.echo("Нет треков для отображения")
            return

        click.echo(f"\nПлейлист '{self.name}' ({len(tracks)} треков):")
        click.echo("=" * 80)
        for i, track in enumerate(tracks, 1):
            click.echo(f"{i:3}. {track}")
        click.echo("=" * 80)

    def save_to_json(self, filename: str) -> bool:
        """Сохранить плейлист в JSON файл"""
        try:
            data = {
                "playlist_name": self.name,
                "tracks": [track.to_dict() for track in self.tracks],
            }

            with open(filename, "w", encoding="utf-8") as f:
                json.dump(data, f, ensure_ascii=False, indent=2)

            click.echo(f"Плейлист сохранен в файл: {filename}")
            return True
        except Exception as e:
            click.echo(f"Ошибка при сохранении в файл {filename}: {e}", err=True)
            return False

    def load_from_json(self, filename: str) -> bool:
        """Загрузить плейлист из JSON файла"""
        try:
            with open(filename, "r", encoding="utf-8") as f:
                data = json.load(f)

            self.name = data.get("playlist_name", self.name)
            self.tracks.clear()

            for track_data in data.get("tracks", []):
                try:
```

```
                track = Track.from_dict(track_data)
                self.add_track(track)
            except Exception as e:
                click.echo(f"Ошибка при загрузке трека: {e}")

        click.echo(f"Плейлист загружен из файла: {filename}")
        click.echo(f"   Загружено треков: {len(self.tracks)}")
        return True
    except FileNotFoundError:
        click.echo(f"Файл не найден: {filename}", err=True)
        return False
    except json.JSONDecodeError:
        click.echo(f"Ошибка формата JSON в файле: {filename}", err=True)
        return False
    except Exception as e:
        click.echo(f"Ошибка при загрузке из файла {filename}: {e}", err=True)
        return False

def __len__(self) -> int:
    """Количество треков в плейлисте"""
    return len(self.tracks)

# Глобальный объект плейлиста
playlist = Playlist("Моя музыкальная коллекция")

def parse_duration(ctx, param, value):
    """Парсинг длительности из строки в timedelta"""
    if value is None:
        return None

    try:
        parts = value.split(":")
        if len(parts) == 2:
            minutes, seconds = map(int, parts)
            return timedelta(minutes=minutes, seconds=seconds)
        elif len(parts) == 3:
            hours, minutes, seconds = map(int, parts)
            return timedelta(hours=hours, minutes=minutes, seconds=seconds)
        else:
            # Попробуем интерпретировать как секунды
            seconds = int(value)
            return timedelta(seconds=seconds)
    except ValueError:
        raise click.BadParameter(
            f"Неверный формат длительности: '{value}'. Используйте MM:SS или HH:MM:SS"
        )

def get_genre_by_name(name: str) -> MusicGenre:
    """Получить жанр по имени"""
    try:
        return MusicGenre[name.upper()]
    except KeyError:
```

```
        # Попробуем найти по значению
        for genre in MusicGenre:
            if genre.value.lower() == name.lower():
                return genre
        raise click.BadParameter(
            f"Неизвестный жанр: '{name}'. Доступные: {', '.join(g.name for g in MusicGenre)}"
        )

def add_demo_tracks():
    """Добавить демонстрационные треки"""
    demo_tracks = [
        Track(
            "Bohemian Rhapsody",
            "Queen",
            timedelta(minutes=5, seconds=55),
            MusicGenre.ROCK,
            1975,
        ),
        Track(
            "Smells Like Teen Spirit",
            "Nirvana",
            timedelta(minutes=5, seconds=1),
            MusicGenre.ROCK,
            1991,
        ),
        Track(
            "Billie Jean",
            "Michael Jackson",
            timedelta(minutes=4, seconds=54),
            MusicGenre.POP,
            1982,
        ),
        Track(
            "Shape of You",
            "Ed Sheeran",
            timedelta(minutes=3, seconds=53),
            MusicGenre.POP,
            2017,
        ),
        Track(
            "Take Five",
            "Dave Brubeck",
            timedelta(minutes=5, seconds=24),
            MusicGenre.JAZZ,
            1959,
        ),
        Track(
            "Hotel California",
            "Eagles",
            timedelta(minutes=6, seconds=30),
            MusicGenre.ROCK,
            1976,
```

```
        ),
        Track(
            "Moonlight Sonata",
            "Beethoven",
            timedelta(minutes=15, seconds=0),
            MusicGenre.CLASSICAL,
            1801,
        ),
        Track(
            "Lose Yourself",
            "Eminem",
            timedelta(minutes=5, seconds=26),
            MusicGenre.HIP_HOP,
            2002,
        ),
        Track(
            "Stairway to Heaven",
            "Led Zeppelin",
            timedelta(minutes=8, seconds=2),
            MusicGenre.ROCK,
            1971,
        ),
        Track(
            "Imagine",
            "John Lennon",
            timedelta(minutes=3, seconds=3),
            MusicGenre.POP,
            1971,
        ),
    ]

    playlist.add_tracks(*demo_tracks)
    return demo_tracks

@click.group()
@click.version_option("1.0.0")
def cli():
    """Музыкальный менеджер - управление коллекцией треков"""
    pass

@cli.command()
@click.option("--title", "-t", required=True, help="Название трека")
@click.option("--artist", "-a", required=True, help="Исполнитель")
@click.option(
    "--duration",
    "-d",
    required=True,
    callback=parse_duration,
    help="Длительность (MM:SS или HH:MM:SS)",
)
@click.option(
    "--genre",
```

```
    "-g",
    type=click.Choice([g.name for g in MusicGenre]),
    default="OTHER",
    help="Жанр музыки",
)
@click.option("--year", "-y", type=int, help="Год выпуска")
def add(title, artist, duration, genre, year):
    """Добавить новый трек в коллекцию"""
    try:
        genre_enum = MusicGenre[genre]
        track = Track(
            title=title, artist=artist, duration=duration, genre=genre_enum, year=year
        )

        playlist.add_track(track)
        click.echo(f"Трек добавлен: {track}")

    except ValueError as e:
        click.echo(f"Ошибка при добавлении трека: {e}", err=True)
    except Exception as e:
        click.echo(f"Неожиданная ошибка: {e}", err=True)

@cli.command()
@click.option(
    "--sort-by",
    "-s",
    type=click.Choice(["duration", "title", "artist"]),
    help="Критерий сортировки",
)
@click.option("--reverse", "-r", is_flag=True, help="Сортировка по убыванию")
def show(sort_by, reverse):
    """Показать все треки в коллекции"""
    if not playlist.tracks:
        click.echo("Плейлист пуст. Используйте команду 'add' для добавления треков.")
        return

    # Применяем сортировку если указана
    if sort_by:
        if sort_by == "duration":
            playlist.sort_by_duration(reverse)
        elif sort_by == "title":
            playlist.sort_by_title(reverse)
        elif sort_by == "artist":
            playlist.sort_by_artist(reverse)

    playlist.display_tracks()

@cli.command()
@click.option("--artist", "-a", help="Фильтр по исполнителю")
@click.option(
    "--genre",
    "-g",
```

```
    type=click.Choice([g.name for g in MusicGenre]),
    help="Фильтр по жанру",
)
@click.option("--year", "-y", type=int, help="Фильтр по году")
@click.option("--min-duration", type=int, help="Минимальная длительность (секунды)")
@click.option("--max-duration", type=int, help="Максимальная длительность (секунды)")
@click.option(
    "--sort-by",
    "-s",
    type=click.Choice(["duration", "title", "artist"]),
    help="Критерий сортировки",
)
@click.option("--reverse", "-r", is_flag=True, help="Сортировка по убыванию")
def filter(artist, genre, year, min_duration, max_duration, sort_by, reverse):
    """Фильтрация треков по заданным критериям"""
    if not playlist.tracks:
        click.echo("Плейлист пуст. Нет данных для фильтрации.")
        return

    filtered_tracks = []

    if artist:
        filtered_tracks = playlist.get_tracks_by_artist(artist)
        click.echo(f"Треки исполнителя '{artist}':")

    elif genre:
        genre_enum = MusicGenre[genre]
        filtered_tracks = playlist.get_tracks_by_genre(genre_enum)
        click.echo(f"Треки жанра '{genre_enum.value}':")

    elif year:
        filtered_tracks = playlist.get_tracks_by_year(year)
        click.echo(f"Треки {year} года:")

    elif min_duration or max_duration:
        min_sec = min_duration or 0
        max_sec = max_duration or 86400  # 24 часа

        if min_sec > max_sec:
            click.echo(
                "Минимальная длительность не может быть больше максимальной",
                err=True,
            )
            return

        filtered_tracks = playlist.get_tracks_in_duration_range(min_sec, max_sec)
        click.echo(f"Треки длительностью от {min_sec} до {max_sec} секунд:")

    else:
        click.echo("Укажите хотя бы один критерий фильтрации")
        return
```

```
    if not filtered_tracks:
        click.echo("Нет треков, соответствующих критериям фильтрации")
        return

    # Применяем сортировку если указана
    if sort_by:
        if sort_by == "duration":
            filtered_tracks.sort(key=lambda t: t.duration_seconds, reverse=reverse)
        elif sort_by == "title":
            filtered_tracks.sort(key=lambda t: t.title.lower(), reverse=reverse)
        elif sort_by == "artist":
            filtered_tracks.sort(key=lambda t: t.artist.lower(), reverse=reverse)

    # Создаем временный плейлист для отображения
    temp_playlist = Playlist("Результаты фильтрации", filtered_tracks)
    temp_playlist.display_tracks()

@cli.command()
def stats():
    """Показать статистику коллекции"""
    if not playlist.tracks:
        click.echo("Плейлист пуст. Нет данных для статистики.")
        return

    stats_data = playlist.get_statistics()

    click.echo(f"\nСтатистика плейлиста '{playlist.name}':")
    click.echo("=" * 50)
    click.echo(f"Всего треков: {stats_data['total_tracks']}")
    click.echo(f"Общая длительность: {stats_data['total_duration']}")
    click.echo(f"Уникальных исполнителей: {stats_data['artists']}")

    if stats_data["genres"]:
        click.echo("\nРаспределение по жанрам:")
        for genre, count in stats_data["genres"].items():
            click.echo(f"  {genre}: {count}")

    if stats_data["years"]:
        click.echo("\nРаспределение по годам:")
        for year, count in sorted(stats_data["years"].items()):
            click.echo(f"  {year}: {count}")

    click.echo("=" * 50)

@cli.command()
@click.option(
    "--file",
    "-f",
    default="playlist.json",
    help="Имя файла (по умолчанию: playlist.json)",
)
def save(file):
```

```
    """Сохранить коллекцию в JSON файл"""
    if not playlist.tracks:
        click.echo("Плейлист пуст. Сохранение пустого плейлиста.")

    playlist.save_to_json(file)

@cli.command()
@click.option("--file", "-f", required=True, help="Имя файла для загрузки")
def load(file):
    """Загрузить коллекцию из JSON файла"""
    if not os.path.exists(file):
        click.echo(f"Файл не найден: {file}", err=True)
        return

    playlist.load_from_json(file)

@cli.command()
@click.option("--count", "-c", default=10, help="Количество демо-треков для добавления")
def demo(count):
    """Добавить демонстрационные треки в коллекцию"""
    all_demo_tracks = add_demo_tracks()

    if count < len(all_demo_tracks):
        # Ограничиваем количество если указано меньше
        playlist.tracks = playlist.tracks[: -len(all_demo_tracks)]  # Удаляем все демо
        for i in range(count):
            playlist.add_track(all_demo_tracks[i])

    click.echo(
        f"Добавлено {min(count, len(all_demo_tracks))} демонстрационных треков"
    )
    click.echo(f"Теперь в плейлисте: {len(playlist)} треков")

@cli.command()
@click.confirmation_option(prompt="Вы уверены, что хотите удалить все треки?")
def clear():
    """Очистить всю коллекцию треков"""
    playlist.tracks.clear()
    click.echo("Коллекция треков очищена")

@cli.command()
@click.argument("index", type=int)
def remove(index):
    """Удалить трек по индексу"""
    if not playlist.tracks:
        click.echo("Плейлист пуст. Нет треков для удаления.")
        return

    if index < 1 or index > len(playlist):
        click.echo(
            f"Неверный индекс. Допустимые значения: 1-{len(playlist)}", err=True
        )
```

```
        return

    removed_track = playlist.tracks[index - 1]
    if playlist.remove_track(index - 1):
        click.echo(f"Трек удален: {removed_track}")
        click.echo(f"Осталось треков: {len(playlist)}")
    else:
        click.echo(f"Не удалось удалить трек с индексом {index}", err=True)

@cli.command()
def genres():
    """Показать доступные музыкальные жанры"""
    click.echo("\nДоступные музыкальные жанры:")
    click.echo("=" * 30)
    for genre in MusicGenre:
        click.echo(f"  {genre.name:15} - {genre.value}")
    click.echo("=" * 30)

@cli.command()
@click.pass_context
def interactive(ctx):
    """Интерактивный режим работы"""
    click.echo("Музыкальный менеджер - Интерактивный режим")
    click.echo("=" * 50)
    click.echo("Доступные команды: add, show, filter, stats, save, load, demo, exit")

    while True:
        try:
            command = click.prompt("\nВведите команду", type=str).strip().lower()

            if command == "exit":
                click.echo("Выход из интерактивного режима")
                break
            elif command == "add":
                title = click.prompt("Название трека")
                artist = click.prompt("Исполнитель")
                duration = click.prompt("Длительность (MM:SS)")
                genre = click.prompt("Жанр", default="OTHER")
                year = click.prompt("Год (опционально)", default=None, type=int)

                ctx.invoke(
                    add,
                    title=title,
                    artist=artist,
                    duration=duration,
                    genre=genre,
                    year=year,
                )

            elif command == "show":
                ctx.invoke(show)
```

```
            elif command == "stats":
                ctx.invoke(stats)

            elif command == "demo":
                ctx.invoke(demo)

            else:
                click.echo(f"Неизвестная команда: {command}")

        except KeyboardInterrupt:
            click.echo("\nВыход из интерактивного режима")
            break
        except Exception as e:
            click.echo(f"Ошибка: {e}")

if __name__ == "__main__":
    cli()
```

Вывод: в ходе лабораторной работы получили навыки построения приложений с интерфейсом командной строки с помощью языка программирования Python версии 3.x.