SPŠE Ječná

Informační technologie - Programování a digitální technologie

Ječná 517, 120 00 Nové Město

KYTE MUSIC PLAYER

Text based Java app for managing and playing songs.

Tobiáš Bíreš, C2c
Informační a komunikační technologie
2025

Content

1 The goal of the work

The goal of this project is to develop a console-based music player application that allows users to manage and play audio files directly from the command line. The application aims to simulate a basic music player with user interaction via typed commands.

The main objectives of the project include:

- Playing individual audio tracks
- Creating and managing playlists
- Marking songs as favorites
- Searching songs by title (including support for regular expressions)
- Supporting different playback modes (e.g., normal, shuffle, loop)

The application is implemented in the Java programming language using object-oriented principles. It is designed to be modular, maintainable, and easily extensible for future improvements.

The source code of the project is publicly available on GitHub:
https://github.com/tobiceqq/MusicPlayer.git

2 Program description

This is a command-line Java application that serves as a simple music player. It is controlled by user input through textual commands. The focus is on clean structure, separation of responsibilities, and easy future extension. The application is organized in modular parts that interact through well-defined interfaces.

2.1 Architecture and Application Flow

The program runs on a command loop – users enter commands (e.g., **play**, **pause**, **next**) and the application responds accordingly. The **AudioPlayer** component handles playback, while playlist and song management are handled by **PlaylistManager** and **Playlist**.

2.2 Class Responsibilities

The **Song** class represents a single music track, including metadata such as title, artist, and duration, and also stores whether the song is marked as a favorite. The **Playlist** class holds a list of songs and manages the current position in the playback queue as well as the playback mode. The **PlaylistManager** class is responsible for creating, storing, selecting, and removing playlists. The **AudioPlayer** handles the playback of `.wav` files, including features like pause, resume, skip, and auto-play of the next track. The **CommandExecutor** class takes user input and delegates it to the appropriate command implementation, which is defined via the *Command interface*.

2.3 User Features

From the user's perspective, the app allows for selecting and playing playlists, controlling the playback of tracks (including pause, skip, and resume), adding songs to playlists by specifying the file path, marking songs as favorites, and searching tracks by title using regular expressions. The user can also switch between playback modes such as normal, shuffle, and repeat. All interactions happen via the console in real time.

3 System Requirements

The app is developed in **Java**. In order to run the program, a corresponding version of the **Java Development Kit (JDK)** must be installed. The app can be run in any IDE that supports Java, such as **IntelliJ IDEA**, **Eclipse**, or **NetBeans**.

Audio playback is handled using the built-in *javax.sound.sampled* library, which is part of the standard Java API and does not require any external dependencies. The program does not rely on any additional frameworks or third-party libraries, ensuring portability and a lightweight setup.

Other than the JDK, no further installation is necessary. The source code is organized in packages that can be compiled and run directly via the command line or using a build system inside an IDE. The application works with music files in the `.wav` format, so it is required that these files are available in the expected folder structure.

4 Source Code Structure

The app follows a clean object-oriented structure and is divided into several logically separated packages. Each package contains classes with clearly defined responsibilities. The code is structured to promote readability, modularity, and future extensibility.

At the core of the application is the **model** package, which includes the fundamental classes **Song**, **Playlist**, and **PlayMode**. These represent the data layer of the application – music tracks, playlists, and playback modes.

The **playlist** package includes the **PlaylistManager**, which handles creating, storing, and switching between playlists. It provides methods for adding and removing songs, as well as selecting the currently active playlist.

It also contains the **PlaylistLoader** class, which is responsible for preloading predefined playlists and songs into the application at startup. This allows for testing, demonstration, or starting with a pre-configured library of tracks.

The **audio** package contains the **AudioPlayer** class, which handles loading and playing `.wav` files using the *javax.sound.sampled* library. It also manages playback state, pause/resume functionality, and transitions between songs.

The **command** package implements the *Command design pattern*. It includes interfaces and concrete command classes such as **Play**, **Pause**, **Next**, **Favorite**, and more. The **CommandExecutor** processes user input and executes the corresponding command.

In addition, the **utils** package provides formatting and helper functions, such as **ConsoleStyle**, which improves the readability of console output.

All code is grouped in a way that makes it easy to locate specific functionality and to extend or modify the project in the future.

5 Testing and Verifying Functions

The functionality of the application was verified through manual testing in the console environment. Various usage scenarios were tested to ensure the correct behavior of individual commands and to handle edge cases properly.

Testing began with basic commands such as **play**, **pause**, and **next**, verifying that the application correctly responded to user input and changed playback state accordingly. Special attention was given to transitions between songs, automatic playback of the next track, and correct handling of pause and resume functionality.

In addition, the functionality of **playlist management** was tested. This included the creation of new playlists, selection of an active playlist, and adding or removing songs. It was verified that the system maintains the correct state and that changes are reflected immediately.

The search function was tested using regular expressions to ensure correct filtering of song titles. For example, typing a partial name or pattern like "**yeat**" correctly listed matching songs.

Error handling was also tested thoroughly. This included attempts to use commands without having an active playlist selected, trying to mark a nonexistent song as favorite, or entering invalid input. In all cases, the program responded with meaningful error messages and did not crash.

The console output of the application helped verify correct execution. In a final stage, the playback of `.wav` files was tested with several sample tracks to ensure compatibility and that audio output was functional.

6 User Manual

The app is operated entirely through a text-based interface. After launching the program, the user is greeted with a welcome message. To explore available functionality, the user may enter the **help** command, which displays a list of supported commands along with a brief description of each. Commands are typed via the keyboard and executed in real time. The interaction is designed to be intuitive and similar to traditional command-line tools.

To begin using the application, the user must either create a new playlist or select one from existing options. The command **select** prompts the user to enter the name of a playlist to activate it. Once a playlist is selected, the user can issue the **play** command to start playback. Commands such as **pause**, **next**, and **favorite** allow the user to control playback, skip songs, or mark a track as favorite.

When adding a song using the **addsong** command, the user is asked to provide the path to the `.wav` file, the song title, and the artist. The length of the song is calculated automatically. Songs can later be removed using the **removesong** command.

The application also supports **searching for songs** using full or partial names, including regular expressions. Additionally, the user can switch between different playback modes (normal, shuffle, repeat) by entering the respective command (**normalmode**, **shuffle**, or **loop**).

After each action, the application provides feedback in the console, confirming success or reporting errors. The layout and formatting of messages are optimized for readability using a custom utility class (**ConsoleStyle**).

All commands are executed sequentially, and the application remains active until the user decides to quit using the **exit** command.

7 Conclusion

The development of this application provided valuable experience in object-oriented design, modular architecture, and working with audio in Java. Throughout the project, key aspects of user interaction, class responsibilities, and command processing were successfully implemented and refined.

One of the main challenges was managing playback state and ensuring smooth transitions between songs. This was resolved by carefully structuring the **AudioPlayer** component and testing edge cases such as pause-resume timing and switching modes mid-playback.

While all core features were implemented, there is still room for future expansion, such as adding a graphical interface, support for different audio formats, or persistence of user data.

Overall, the project met the original goal of building a functional and user-friendly console music player. It strengthened understanding of design patterns (especially the Command pattern), error handling, and real-time user input in Java.

8 Sources

During the development of the application and preparation of the documentation, several external sources of information were used to better understand Java programming, object-oriented design, and audio handling. The following materials were particularly helpful:

- **Colours In Console in Java Using ANSI Color codes**
- **OpenAI ChatGPT**
- **School presentations and materials**
- **Online IT resources**