1. Code Quality and Innovative Features (5 Marks)

In this section, I will break down the code quality and innovative features in the context of the **console-based music player**. This includes a thorough examination of the design choices, organization, efficiency, and unique aspects of the features implemented in the music player.

Code Quality

1. **Modular and Maintainable Code**:
    - The code is structured into distinct, self-contained classes that each handle specific responsibilities:
        - **Music Class**: This class represents a single song, with attributes like title, artist, and URL. It encapsulates the song's details and provides methods to access them, promoting a clear structure and reusability.
        - **MusicPlayer Class**: This class manages the playlist, handles user input, and performs the music player operations (e.g., playing songs, navigating through the playlist, adding/removing songs). This separation of concerns makes the program easier to maintain and extend.
        - **Main Class (OnlineMusicPlayer)**: This is the entry point of the application, managing user interaction and coordinating the other components. It demonstrates how well the Music and MusicPlayer classes integrate to form a complete user experience.
    - By following this structure, each class has a **single responsibility** (according to SOLID principles), which enhances maintainability and reduces code duplication.
2. **Clear and Descriptive Method Names**:
    - The methods in the program are named intuitively:
        - addSong(), playSong(), nextSong(), previousSong(), showPlaylist(), etc. These method names clearly indicate their purpose and behavior, making the code easy to read and understand without needing excessive comments or documentation.
    - This improves **readability** and **clarity**, helping developers understand the flow of the program and easily navigate between components.
3. **Efficient Data Structures**:
    - The program uses **ArrayList** to store the playlist, which is an efficient and flexible choice. The **ArrayList** allows dynamic resizing as songs are added or removed. This ensures that the music player is flexible, can handle varying amounts of data, and supports constant time access to individual elements.
    - The sorting of the playlist is achieved using Comparator.comparing(), which ensures that the sorting operation is efficient, concise, and easy to understand.
4. **Error Handling**:
    - Proper error handling is integrated into the music player:

        - **Invalid User Input**: The program handles invalid inputs (e.g., non-integer values for menu choices) with a try-catch block. This prevents the program from crashing due to user errors and provides appropriate feedback to the user.
        - **Empty Playlist Handling**: The code checks if the playlist is empty before performing operations like playing, skipping, or displaying songs, which avoids errors or exceptions in such cases.
        - This approach **increases robustness** and ensures the program can run smoothly even when encountering common user mistakes.

5. **Clear Output and Feedback to User**:
    - The program outputs clear and informative messages to the user about the actions taking place. For example:
        - When playing a song, the title, artist, and URL are displayed.
        - When an action is performed (e.g., removing a song), confirmation messages are printed.
        - If the playlist is empty, the user is informed of the situation.
    - This enhances **user experience** by providing adequate feedback, so users always know what's going on and are able to interact meaningfully with the system.

Innovative Features

1. **Playlist Management**:
    - Users can add songs to the playlist, remove them, and view the entire playlist. These are **standard features** for a music player, but the innovation here lies in the **flexibility** of the playlist management system:
        - The playlist is not fixed; it can grow or shrink dynamically as songs are added or removed.
        - Users can easily manage the playlist by interacting with simple commands (like adding, removing, and sorting songs), offering a rich set of options within a console-based environment.
        - The ability to **remove songs by title** and **search songs** by either title or artist are powerful functionalities that make the player more useful and customizable to users.
2. **Search Functionality**:
    - The music player allows users to **search** for songs by either the **song title** or **artist name**. This feature is often associated with more complex applications like streaming platforms (e.g., Spotify, YouTube), but is integrated in this basic console player.
    - The search feature adds a layer of **interactivity** and **user control**, allowing them to quickly find songs in a large playlist, making the program more engaging and user-friendly.

- The search logic performs a case-insensitive search, which means that it handles different capitalizations and spelling variations of song titles and artist names, improving the **robustness** of the search feature.

3. **Sorting Playlist**:
    - The ability to **sort the playlist** by title is a key feature that allows users to organize their music in an intuitive manner. Sorting by title is a basic but useful function for managing large playlists.
    - The sorting algorithm (using Java's Comparator.comparing()) is efficient and implemented in a simple, easy-to-understand way. This enhances the **usability** of the player and introduces an important feature commonly found in full-fledged music apps.
    - This sorting feature adds a layer of **interactivity**, allowing users to personalize their experience further.
4. **Navigating the Playlist (Next/Previous Song)**:
    - The **next** and **previous song** navigation options mimic the behavior of a real-world music player, allowing users to move through their playlist with ease. This makes the console-based application feel like a real music player with essential features for playback control.
    - The navigation is handled with **circular indexing**: after the last song, it loops back to the first song, and before the first song, it loops to the last song. This behavior mimics a real music player and enhances the **user experience** by providing continuous navigation.
5. **Simulating Online Music Playback**:
    - The player **simulates** playing songs by displaying their details (title, artist, URL). This **online streaming feature** in a console-based program offers a glimpse of how an actual online music player might behave, where songs are fetched from an online source (represented by URLs in this case).
    - This feature introduces a level of **realism** and context, as it acknowledges the online nature of music streaming.

Summary of Innovative Features:

- **Playlist management** (add, remove, sort).
- **Search functionality** for songs by title or artist.
- **Navigation through songs** (next/previous).
- **Online music simulation** with URLs and song details.
- **Efficient data structures** like ArrayList for dynamic playlist handling.
- **Sorting functionality** for organizing playlists.

Conclusion:

The **code quality** of the music player is high, with a modular structure, well-defined methods, and efficient error handling. The code is designed to be easily extensible and maintainable. The **innovative features**—such as dynamic playlist management, search functionality, sorting, and next/previous song navigation—make the console-based music player both **functional** and **user-friendly**. These features are often found in more sophisticated applications, but they are thoughtfully implemented here in a console-based format, demonstrating a good balance between simplicity and powerful functionality.