Overview:

The **Java Music Player** is a desktop application developed using the **Swing** GUI toolkit and **javax.sound.sampled** API in Java. The purpose of the application is to allow users to select, play, pause, and loop audio files, specifically in **WAV** format. The music player demonstrates the integration of GUI components with audio playback functionality, providing a simple yet effective tool to manage audio playback in Java-based desktop applications.

Key Features:

1. **File Selection:**
    - The user can select an audio file through a **file chooser** dialog (JFileChooser), which filters files to only show .wav files. Once a file is selected, the file's path is displayed in a text field (JTextField).
2. **Play/Pause Functionality:**
    - A **Play** button allows the user to start the audio playback.
    - The **Pause** button pauses the audio at its current position. Once paused, the button label changes to **Resume**, which, when clicked, resumes the audio from where it was paused. This provides the user with the flexibility to control playback effectively.
3. **Looping Audio:**
    - A **Loop** button allows the user to toggle continuous audio playback (looping). When looping is enabled, the audio will restart automatically once it finishes, without requiring any user interaction. The button’s label changes to **Stop Loop** when looping is active, and clicking it stops the looping feature.

4. **Audio Playback:**
    - The core of the music player is powered by Java's Clip class, which is part of the javax.sound.sampled package. The Clip class allows for the loading and playback of audio files in memory, enabling efficient playback control (start, stop, pause, resume).
    - The application handles basic audio control, such as stopping and starting the clip, as well as stopping playback when the user chooses to pause or stop the audio.
5. **Graphical User Interface (GUI):**
    - The GUI is designed using Java's **Swing** framework, ensuring that it is platform-independent and works on different operating systems.
    - The interface is simple and intuitive, comprising a text field for displaying the selected file path, buttons for file selection, audio playback, pause/resume, and looping control.
    - A **FlowLayout** manager is used to arrange the components in a straightforward, top-to-bottom layout.
6. **Audio Handling and User Interaction:**
    - The application uses a JFileChooser dialog to open a file selection window, allowing the user to choose a .wav audio file.
    - Upon selecting a file, the file path is displayed in the text field, and the user can then press the **Play** button to start playback.
    - The **Play** button starts the audio by loading the audio file into an AudioInputStream, which is then fed into a Clip object for playback.
    - If the user chooses to pause the music, the **Pause** button will stop the audio playback but retain the current position of the track. Pressing the button again resumes playback from that position.

- The **Loop** button can toggle the loop functionality, allowing the audio to play continuously without interruption. When looping is enabled, the **Loop** button changes its label to **Stop Loop**, and pressing it again stops the looping feature.

7. **Error Handling:**
   - The application is designed to catch and display basic exceptions. For example, if the user attempts to load an invalid file or if there's an issue during the audio playback (e.g., an unsupported audio format), the application will display the exception to inform the user.
8. **Extensibility:**
   - While the current implementation only supports .wav files, the architecture is flexible enough to be extended to support other audio formats (like MP3 or OGG) by incorporating additional libraries such as JavaZoom's JLayer for MP3 support.
   - The player can also be enhanced with additional features such as volume control, equalizer settings, track navigation (next/previous track), and playlist management to create a more feature-rich media player.