Music Playlist Manager

Scenario

This project is a music playlist manager. It allows users to explore, filter, and organize a large collection of songs sourced from a CSV file. The user can search songs by artist, genre, or release date, sort songs by title or popularity (number of plays), premium users have the added ability to create playlists and load playlists

Design Paradigm

This project demonstrates:

- Class hierarchy using inheritance
- An interface with an abstract method
- Polymorphism with method overriding and overloading
- Use of collections such as ArrayList, Map, and Set
- File I/O (reading and writing from CSV)
- Stream for sorting and filtering
- Exception handling
- Test-Driven Development (JUnit tests)
- Git for version control

Expected Output

- Load and parse \`songs.csv\` to display song data
- Filter and view songs by genre, artist, or release year
- Find out the top 10 artists/songs (by popularity) per year
- Search songs by name/artist name
- Sort songs by title or number of plays
- Create and manage playlists (for Premium users)
- Favorite songs/artists (For premium users)
- Show statistics like most played artist and average song duration
- Play ads for guest users when playing songs.

Class Hierarchies

Hierarchy 1: Songs

- \`Song\` (base class)

  - \`PopSong\`

- `RockSong`
- `CountrySong`

Hierarchy 2: Users

- `User` (base class)
  - `GuestUser`
  - `PremiumUser`

Interface

Interface: `Playable`

- Abstract method: `void play()`
- Implemented by: `Song` and `Ad`

Purpose: Defines a common behavior for anything that can be "played"

Runtime Polymorphism

- Method `play()` overridden in each subclass of `Song`, and `ad` (Example: when playing a pop song it can print out: “playing a pop hit”, but the message will be different for different genres)
- Method `toString()` overridden in `Song` and `User`
- Method overloading (addSong(Song), addSong(String title, String artist, …))
-

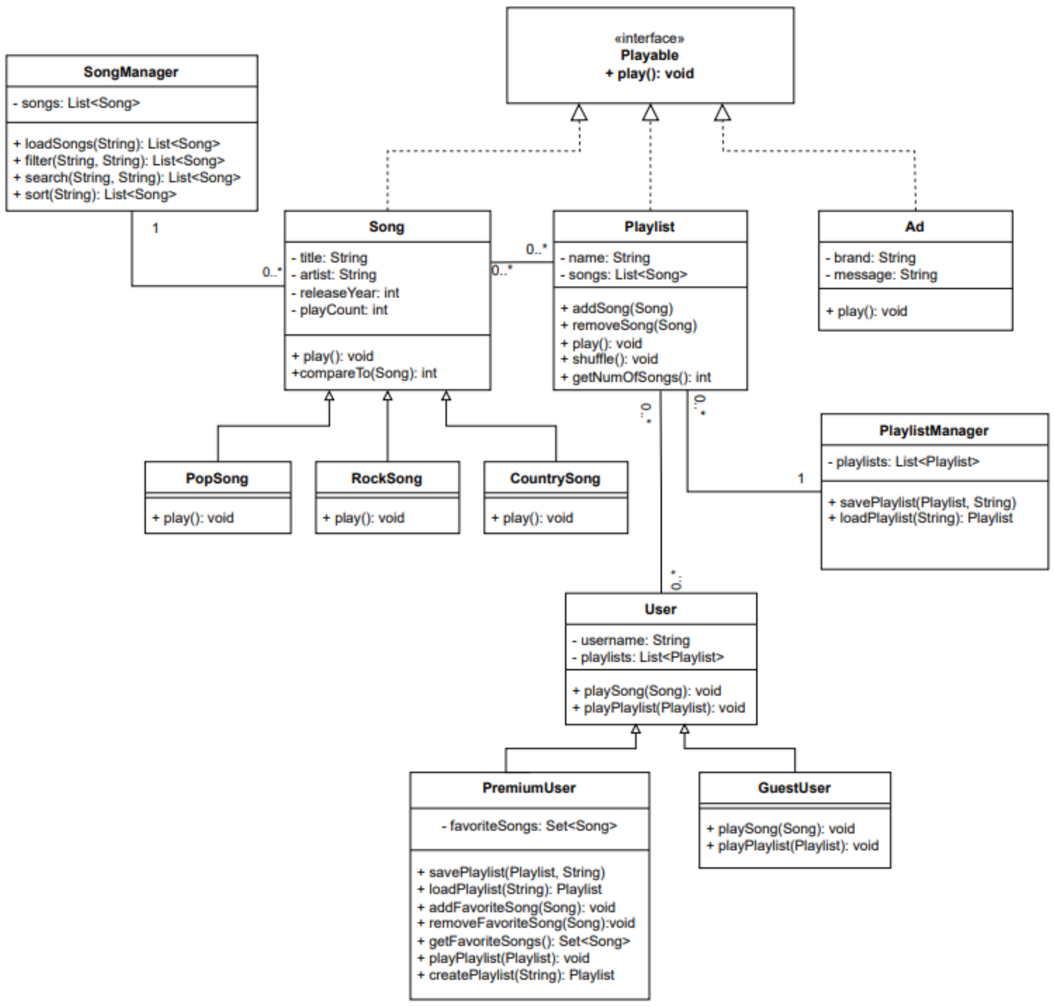
TextIO

- `SongManager` reads songs from `songs.csv` and sort/filters the songs.
- `PlaylistManager` saves user playlists to a text or CSV file

Comparable and Comparator

- `Song` implements `Comparable<Song>` to sort by title
- Custom comparators:
  - `CompareByReleaseDate`
  - `CompareByPlayCount`

UML Class Diagram


«interface»
Playable
+ play(): void
SongManager
- songs: List<Song>
+ loadSongs(String): List<Song>
+ filter(String, String): List<Song>
+ search(String, String): List<Song>
+ sort(String): List<Song>
1
0..*
Song
- title: String
- artist: String
- releaseYear: int
- playCount: int
+ play(): void
+compareTo(Song): int
0..*
0..*
Playlist
- name: String
- songs: List<Song>
+ addSong(Song)
+ removeSong(Song)
+ play(): void
+ shuffle(): void
+ getNumOfSongs(): int
Ad
- brand: String
- message: String
+ play(): void
PopSong
+ play(): void
RockSong
+ play(): void
CountrySong
+ play(): void
0..*
0..*
PlaylistManager
- playlists: List<Playlist>
+ savePlaylist(Playlist, String)
+ loadPlaylist(String): Playlist
1
0..*
User
- username: String
- playlists: List<Playlist>
+ playSong(Song): void
+ playPlaylist(Playlist): void
PremiumUser
- favoriteSongs: Set<Song>
+ savePlaylist(Playlist, String)
+ loadPlaylist(String): Playlist
+ addFavoriteSong(Song): void
+ removeFavoriteSong(Song):void
+ getFavoriteSongs(): Set<Song>
+ playPlaylist(Playlist): void
+ createPlaylist(String): Playlist
GuestUser
+ playSong(Song): void
+ playPlaylist(Playlist): void

What will be implemented for Deliverable 2

- Base classes: `Song` and its subclasses, `User` and its subclasses `, `Ad`
- Interface: `Playable`
- Song loading from CSV in `SongManager`
- Playlist creation in `PlaylistManager`
- Method headers with JavaDoc
- JUnit tests