Problem Description

Project Title:
Rhythm & Roster: The Ultimate Playlist Manager

Story Recap:
Meet Alex, a passionate music lover with an ever-growing collection of songs spanning every genre imaginable. Alex noticed that traditional music players didn't offer enough flexibility to organize a diverse musical library. Frustrated with the tedious process of sifting through endless tracks for that perfect song, Alex decided to build a personalized solution. Alex envisioned a Music Playlist Manager that supports multiple playlists tailored to different moods—be it an energetic workout mix, a relaxing chill-out session, or a nostalgic throwback collection. Each playlist stores songs with details like title, artist, album, genre, and duration, and the system lets users easily filter and search through their music collection.

Your task is to help Alex bring this vision to life by designing and implementing the Music Playlist Manager using object-oriented programming principles. The solution must support multiple playlist management, advanced filtering options, and enhanced search functionality.

1. Song Class

Purpose:
Represents a single song with all its details.

Attributes:

- **title:** *String*
  The song's title.
- **artist:** *String*
  The artist's name.
- **album:** *String*
  The album name.
- **genre:** *String*
  The category of the song (e.g., Pop, Rock).
- **duration:** *Integer*
  The length of the song in seconds (or, if preferred, a formatted string).

Methods:

1. **Constructor**

Signature (pseudo-code):

```
Song(String title, String artist, String album, String genre, Integer duration)
```

Parameters:

- `title` : the song title.
- `artist` : the artist's name.
- `album` : the album name.
- `genre` : the song genre.
- `duration` : the song duration (in seconds).
  Return Type:
  None (constructor)
  Purpose:
  Initializes a new `Song` object with the provided details.

2. **displayDetails()**
   Signature (pseudo-code):

```
String displayDetails()
```

Parameters:

- None
  Return Type:
  String
  Purpose:
  Returns a neatly formatted string presenting the song's details.
  Example Output:
  `"Title: Imagine, Artist: John Lennon, Album: Imagine, Genre: Rock, Duration: 183s"`

2. Playlist Class

Purpose:
Manages a collection of `Song` objects, representing a single playlist with a unique name.

Attributes:

- **name:** *String*

The unique name of the playlist.

- **songs:** *List of Song objects*
  A collection of songs contained in the playlist.

Methods:

1. **Constructor**
   Signature (pseudo-code):

   ```
   Playlist(String name)
   ```

   Parameters:

   - `name` : the unique name for the playlist.
     Return Type:
     None (constructor)
     Purpose:
     Initializes the playlist with the given name and an empty collection of songs.

2. **addSong(song)**
   Signature (pseudo-code):

   ```
   void addSong(Song song)
   ```

   Parameters:

   - `song` : a `Song` object to add to the playlist.
     Return Type:
     None
     Purpose:
     Adds the provided song to the playlist's collection.

3. **removeSong(identifier)**
   Signature (pseudo-code):

   ```
   Boolean removeSong(String identifier)
   ```

   Parameters:

   - `identifier` : a unique identifier to locate the song (for example, the song's title).
     Return Type:
     Boolean (returns `true` if removal was successful, or `false` if the song was not found)
     Purpose:

Searches for the song by the given identifier and removes the first matching song from the collection.

4. **getSongs()**
   Signature (pseudo-code):

```
List<Song> getSongs()
```

   Parameters:

   - None
     Return Type:
     List of Song objects
     Purpose:
     Returns the complete list of songs in the playlist.

5. **filterSongs(criteria, value)**
   Signature (pseudo-code):

```
List<Song> filterSongs(String criteria, String value)
```

   Parameters:

   - `criteria` : the attribute to filter by (e.g., `"genre"` or `"artist"`).
   - `value` : the value to match (e.g., `"Rock"` or `"John Lennon"`).
     Return Type:
     List of Song objects
     Purpose:
     Iterates over the songs and returns a subset of songs that match the given criteria.

6. **searchSongs(keyword)**
   Signature (pseudo-code):

```
List<Song> searchSongs(String keyword)
```

   Parameters:

   - `keyword` : a keyword string to search for within the song's attributes (title, artist, album, or genre).
     Return Type:
     List of Song objects
     Purpose:

Searches through the songs in the playlist and returns a list of songs that contain the keyword in any attribute.

3. PlaylistManager Class

Purpose:
Acts as the central controller for managing multiple playlists. It provides functionality to create, delete, and access playlists as well as perform cross-playlist operations.

Attributes:

- **playlists:** *List of Playlist objects*
  A collection that holds all the playlists.

Methods:

1. **Constructor**
   Signature (pseudo-code):

   ```
   PlaylistManager()
   ```

   Parameters:

   - None
     Return Type:
     None (constructor)
     Purpose:
     Initializes the manager with an empty collection of playlists.

2. **createPlaylist(name)**
   Signature (pseudo-code):

   ```
   void createPlaylist(String name)
   ```

   Parameters:

   - `name` : the unique name for the new playlist.
     Return Type:
     None
     Purpose:
     Creates a new `Playlist` object with the provided name and adds it to the collection.

3. **deletePlaylist(name)**
   Signature (pseudo-code):

   ```
   Boolean deletePlaylist(String name)
   ```

   Parameters:

   - `name` : the name of the playlist to delete.

   Return Type:
   Boolean (returns `true` if deletion was successful, or `false` if the playlist was not found)
   Purpose:
   Removes the `Playlist` with the specified name from the collection.

4. **getPlaylist(name)**
   Signature (pseudo-code):

   ```
   Playlist getPlaylist(String name)
   ```

   Parameters:

   - `name` : the name of the playlist to retrieve.

   Return Type:
   Playlist object (or `null` / `None` if not found)
   Purpose:
   Searches for and returns the `Playlist` with the specified name.

5. **listPlaylists()**
   Signature (pseudo-code):

   ```
   List<String> listPlaylists()
   ```

   Parameters:

   - None

   Return Type:
   List of String (e.g., list of playlist names)
   Purpose:
   Returns a list of all playlist names (or details) managed by the `PlaylistManager` .

6. **crossPlaylistSearch(keyword)**
   Signature (pseudo-code):

```
List<Song> crossPlaylistSearch(String keyword)
```

Parameters:

- `keyword` : the keyword to search across all playlists.
  Return Type:
  List of Song objects
  Purpose:
  Iterates over every playlist and applies the `searchSongs(keyword)` method to compile and return a combined list of matching songs from all playlists.

How the Classes Interact

1. **Creating Songs and Adding to a Playlist:**
   - The user creates one or more `Song` objects using the `Song` class.
   - A `Playlist` object is created (via the `PlaylistManager`), and songs are added using the `addSong(song)` method.
2. **Managing Multiple Playlists:**
   - The `PlaylistManager` class maintains a collection of `Playlist` objects.
   - Users can create new playlists, delete existing ones, or select a specific playlist using methods provided by the `PlaylistManager` .
3. **Filtering and Searching:**
   - Within a selected `Playlist` , the user can call `filterSongs(criteria, value)` to obtain a subset of songs.
   - Similarly, the `searchSongs(keyword)` method lets the user search within a single playlist.
   - For broader searches across all playlists, the `crossPlaylistSearch(keyword)` method in the `PlaylistManager` can be used.

Summary

- **Song Class:**
  - **Attributes:**
    - `title: String`

        - artist: String
        - album: String
        - genre: String
        - duration: Integer
    - **Methods:**
        - Song(String, String, String, String, Integer)
        - String displayDetails()

- **Playlist Class:**
    - **Attributes:**
        - name: String
        - songs: List<Song>
    - **Methods:**
        - Playlist(String)
        - void addSong(Song song)
        - Boolean removeSong(String identifier)
        - List<Song> getSongs()
        - List<Song> filterSongs(String criteria, String value)
        - List<Song> searchSongs(String keyword)

- **PlaylistManager Class:**
    - **Attributes:**
        - playlists: List<Playlist>
    - **Methods:**
        - PlaylistManager()
        - void createPlaylist(String name)
        - Boolean deletePlaylist(String name)
        - Playlist getPlaylist(String name)
        - List<String> listPlaylists()
        - List<Song> crossPlaylistSearch(String keyword)