RESPONSI
PRAKTIKUM PEMROGRAMAN BERORIENTASI OBJEK
2025/2026

IDENTITAS

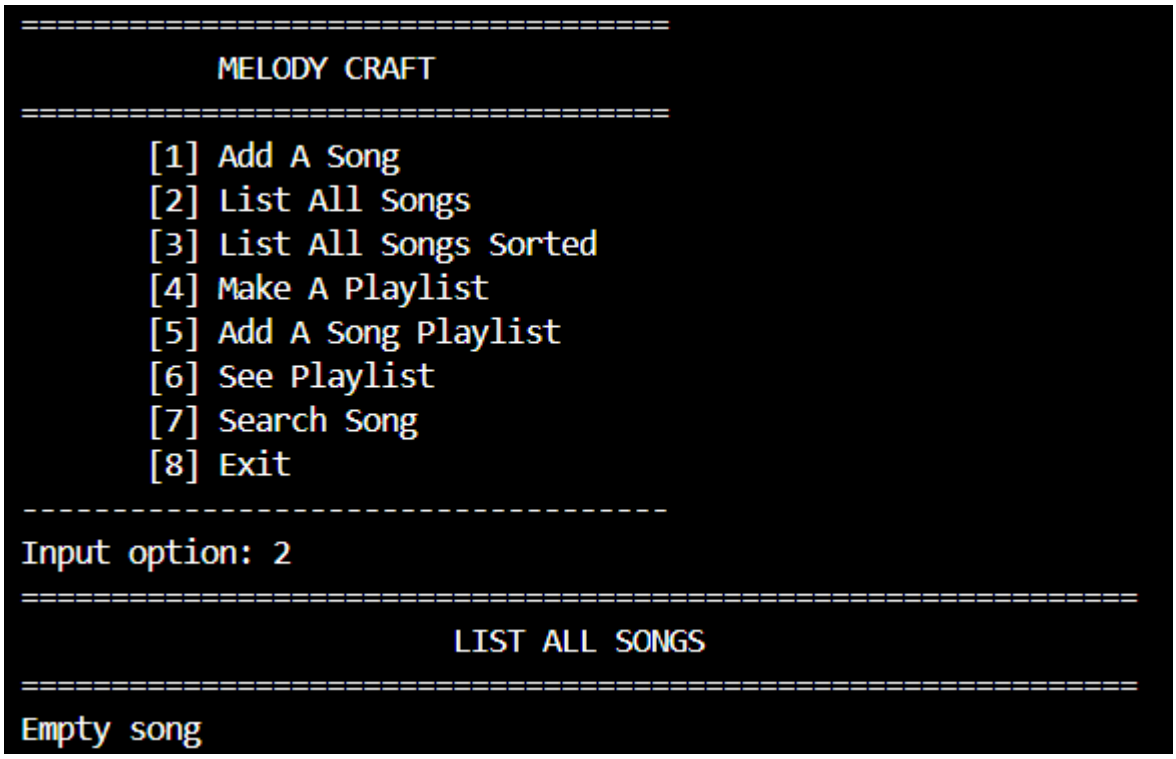
Nama	:	Aprillia Wulandari
NIM	:	L0124040
Kelas	:	C
Judul Program	:	Melody Craft
Deskripsi Program	:	Melody Craft adalah program manajemen musik yang bersifat interaktif. Pengguna dapat menambahkan lagu, membuat *playlist*, mencari lagu berdasarkan judul, menampilkan lagu yang tersedia, serta melihat *playlist* yang dibuat beserta lagu di dalamnya. Seluruh data lagu dan *playlist* akan disimpan secara permanen dalam *file* sehingga dapat digunakan kembali pada saat program dijalankan.

DOKUMENTASI PROGRAM

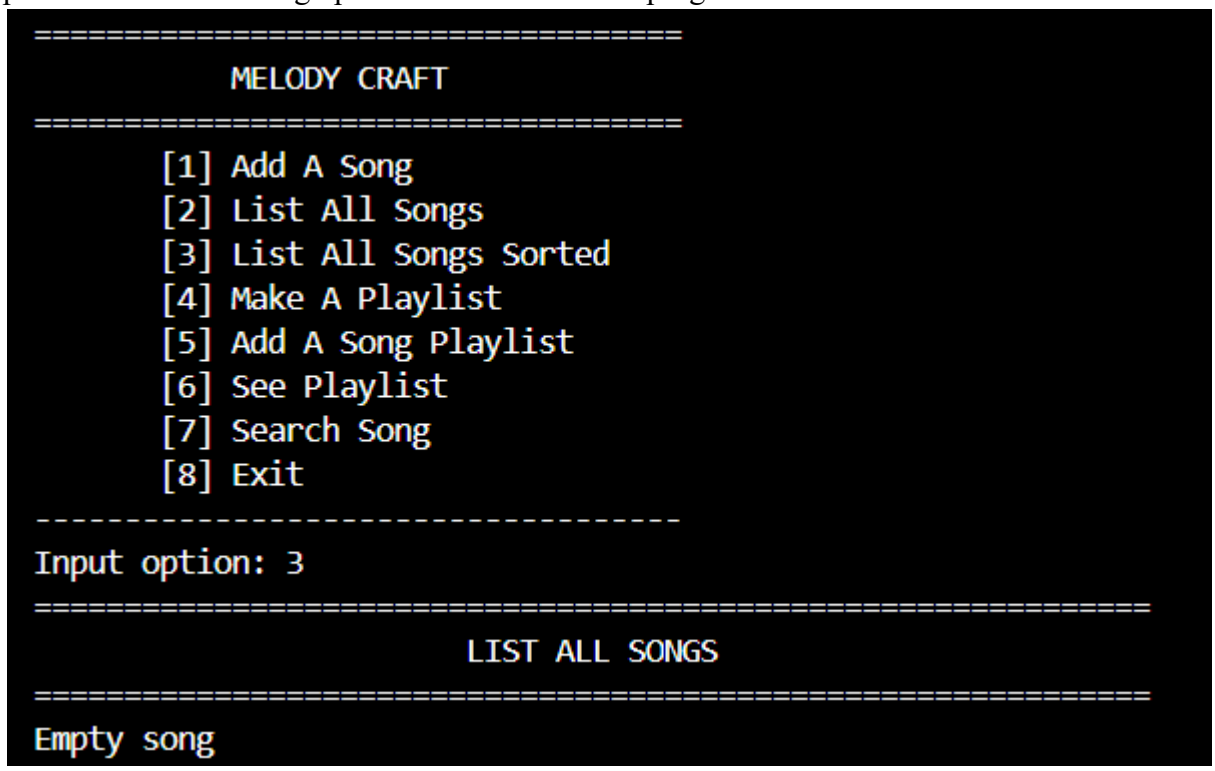
Instruksi Pemakaian:

```
PS C:\Users\liali\OneDrive\Dokumen\OOP\responsi> javac -cp src src\app\Main.java
PS C:\Users\liali\OneDrive\Dokumen\OOP\responsi> java -ea -cp src app.Main
==================================
          MELODY CRAFT
==================================
       [1] Add A Song
       [2] List All Songs
       [3] List All Songs Sorted
       [4] Make A Playlist
       [5] Add A Song Playlist
       [6] See Playlist
       [7] Search Song
       [8] Exit
----------------------------------
Input option:
```

Untuk mengkompilasi program digunakan perintah “javac -cp src src\app\Main.java” di mana opsi “-cp src” menunjukkan folder “src” sebagai lokasi *file*. Sementara itu, untuk menjalankan program, gunakan perintah “java -ea -cp src app.Main”, di mana opsi “-ea” digunakan untuk mengaktifkan *assertion* dan “-cp src” menunjukkan lokasi kelas *Main* yang akan dijalankan. Setelah dijalankan sistem akan menampilkan menu utama Melody Craft dengan delapan pilihan yang dapat dipilih oleh *user*.



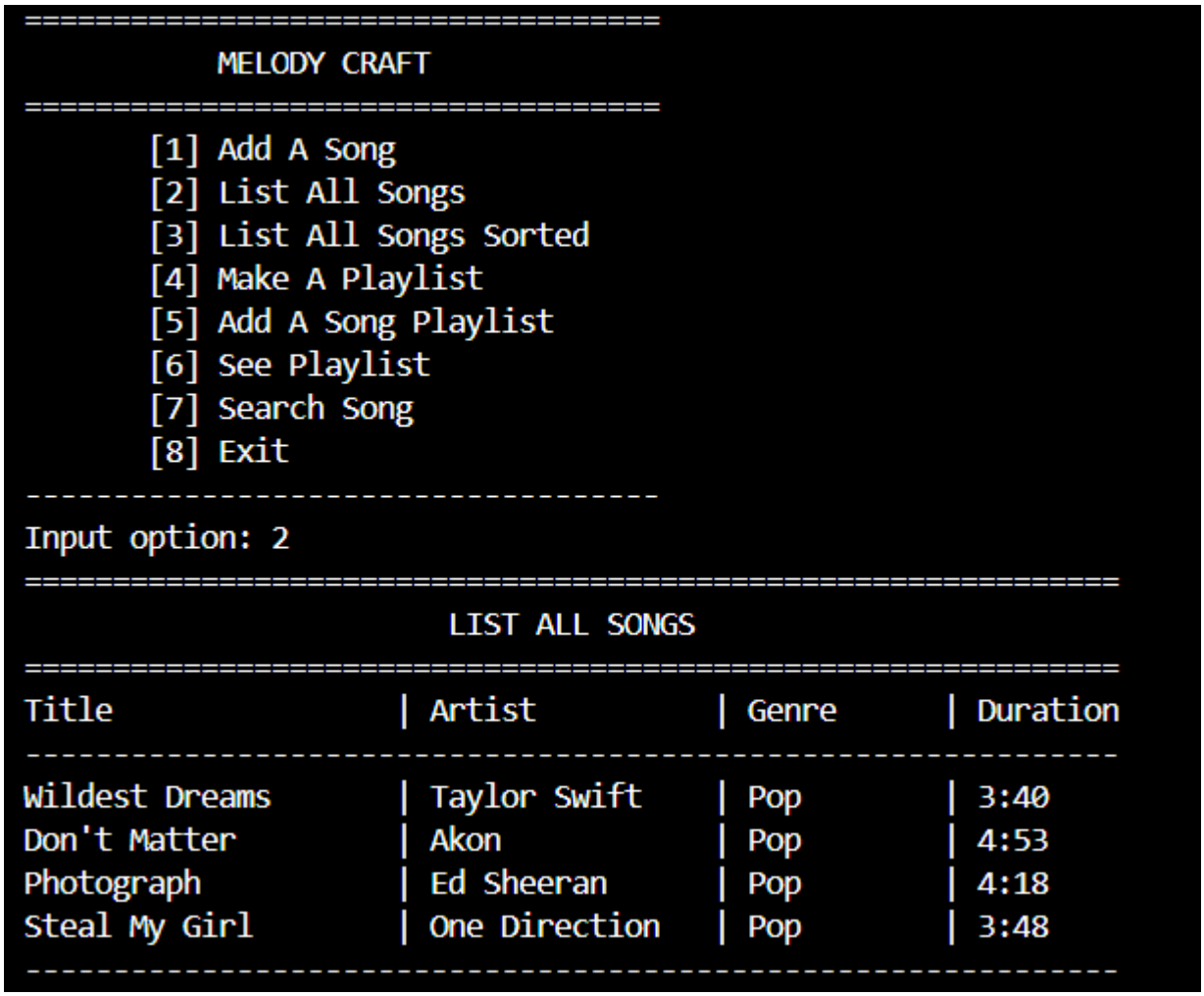
Ketika *user input* opsi ke-2 untuk melihat daftar lagu, sistem akan menampilkan seluruh daftar lagu yang ada. Gambar di atas merupakan contoh pada saat awal *user* mencoba eksekusi program. Terlihat bahwa belum ada lagu yang ditambahkan dan sistem menampilkan pesan "*Empty song*" sehingga *user* perlu menambahkan lagu pada saat awal eksekusi program.



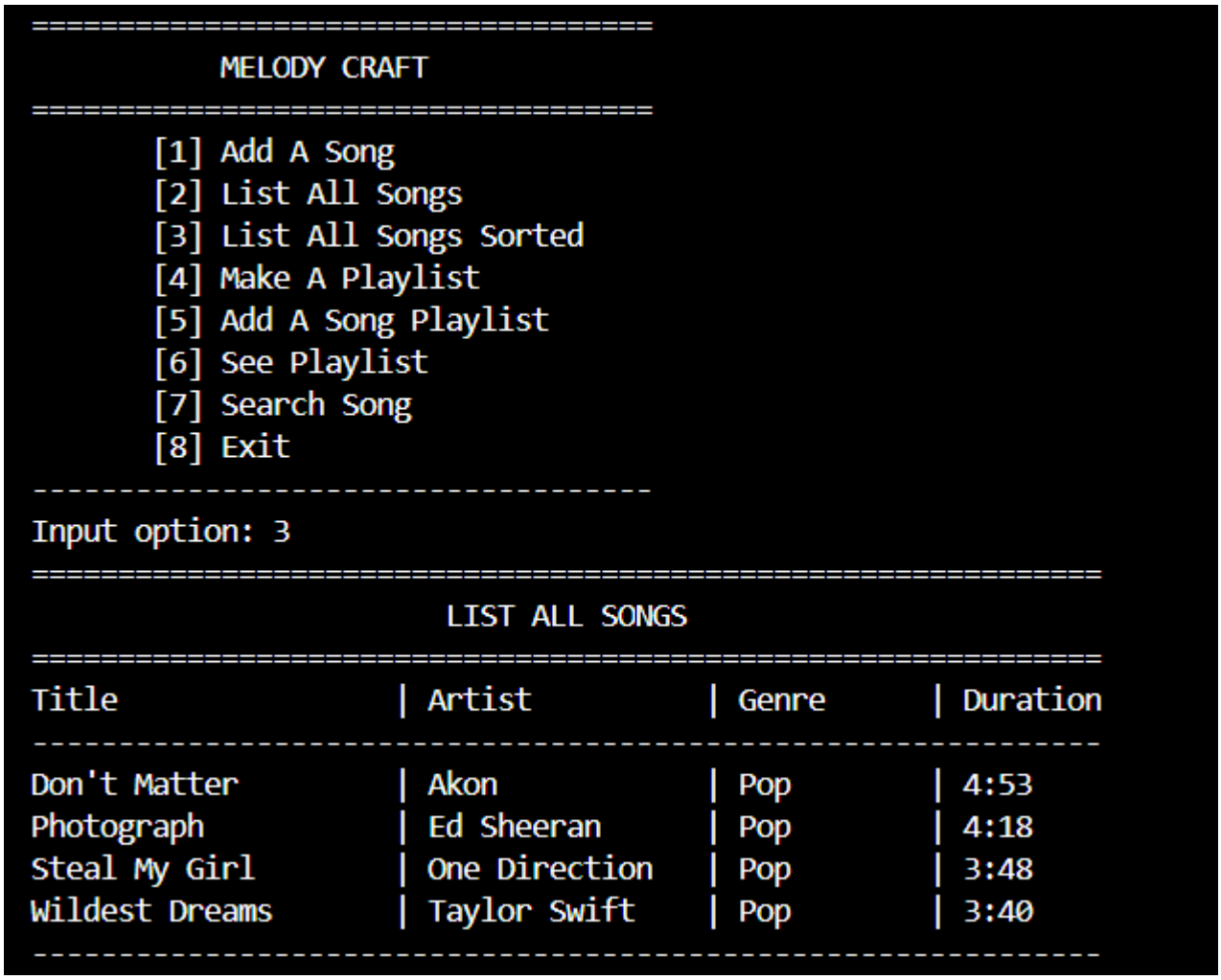
Ketika *user* mencoba *input* opsi ke-3 untuk melihat daftar lagu yang sudah terurut, sistem menampilkan seluruh daftar lagu yang terurut. Jika belum ada lagu yang ditambahkan, sistem akan menampilkan pesan "*Empty song*", sama seperti pada opsi ke-2 sebagai tanda bahwa daftar lagu

masih kosong pada saat awal eksekusi program.

Opsi ke-1 digunakan untuk menambahkan lagu baru. Pada gambar di atas, *user* mencoba untuk menambahkan lagu baru. *User* akan diminta untuk memasukkan data lagu, seperti judul lagu, nama artis, genre, dan durasi dengan format "mm:ss". Jika semua *input* valid, sistem akan menyimpan lagu tersebut dan menampilkan pesan bahwa lagu berhasil ditambahkan.



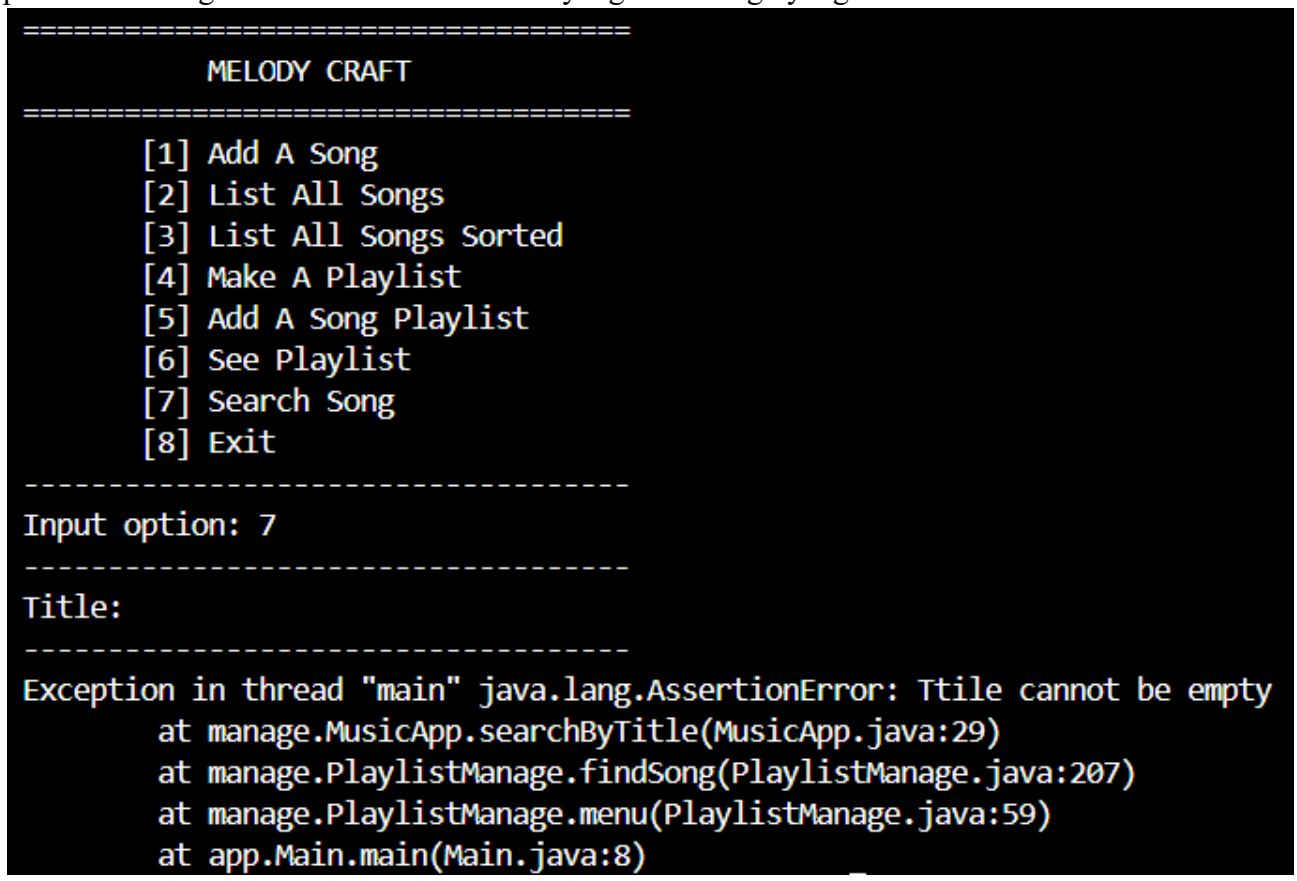
Ketika *user* mencoba *input* opsi ke-2 kembali untuk melihat daftar lagu setelah menambahkan beberapa lagu, sistem akan menampilkan seluruh daftar lagu beserta informasinya. Semua lagu yang sudah ditambahkan sebelumnya, ditampilkan secara berurut sesuai urutan penambahan.



Ketika *user* memilih opsi ke-3 lagi untuk melihat daftar lagu yang sudah terurut, sistem akan menampilkan seluruh daftar lagu beserta informasinya berurutan berdasarkan judul secara alfabet.

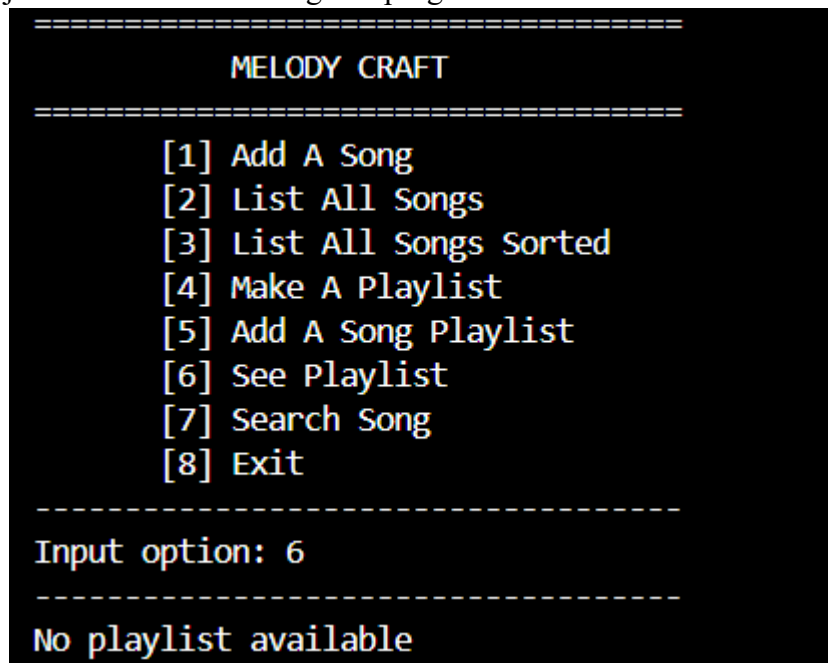
Opsi ke-7 digunakan untuk mencari lagu berdasarkan judul. Pada gambar di atas, *user* mencoba memasukkan judul lagu "steal my girl" sebagai *keyword* pencarian. Jika ditemukan judul tersebut, sistem kemudian menampilkan informasi lengkap lagu tersebut. Pencarian bersifat tidak sensitif terhadap huruf besar maupun kecil, sehingga lagu dapat ditemukan meskipun *input* berbeda format hurufnya.

Gambar di atas merupakan contoh ketika *user* mencoba mencari lagu berdasarkan judul, tetapi lagu tersebut belum ada di dalam daftar lagu. Misalnya *user input* judul "youth", sistem akan menampilkan pesan bahwa lagu tersebut tidak ditemukan yang berarti lagu yang dicari belum ada di dalam daftar.



Gambar di atas merupakan contoh ketika *user* mencoba untuk mengosongkan *input* saat mencari lagu.

Karena judul lagu tidak boleh kosong, sistem akan mengeluarkan *assertion error* dengan pesan bahwa judul tidak boleh kosong dan program akan dihentikan.



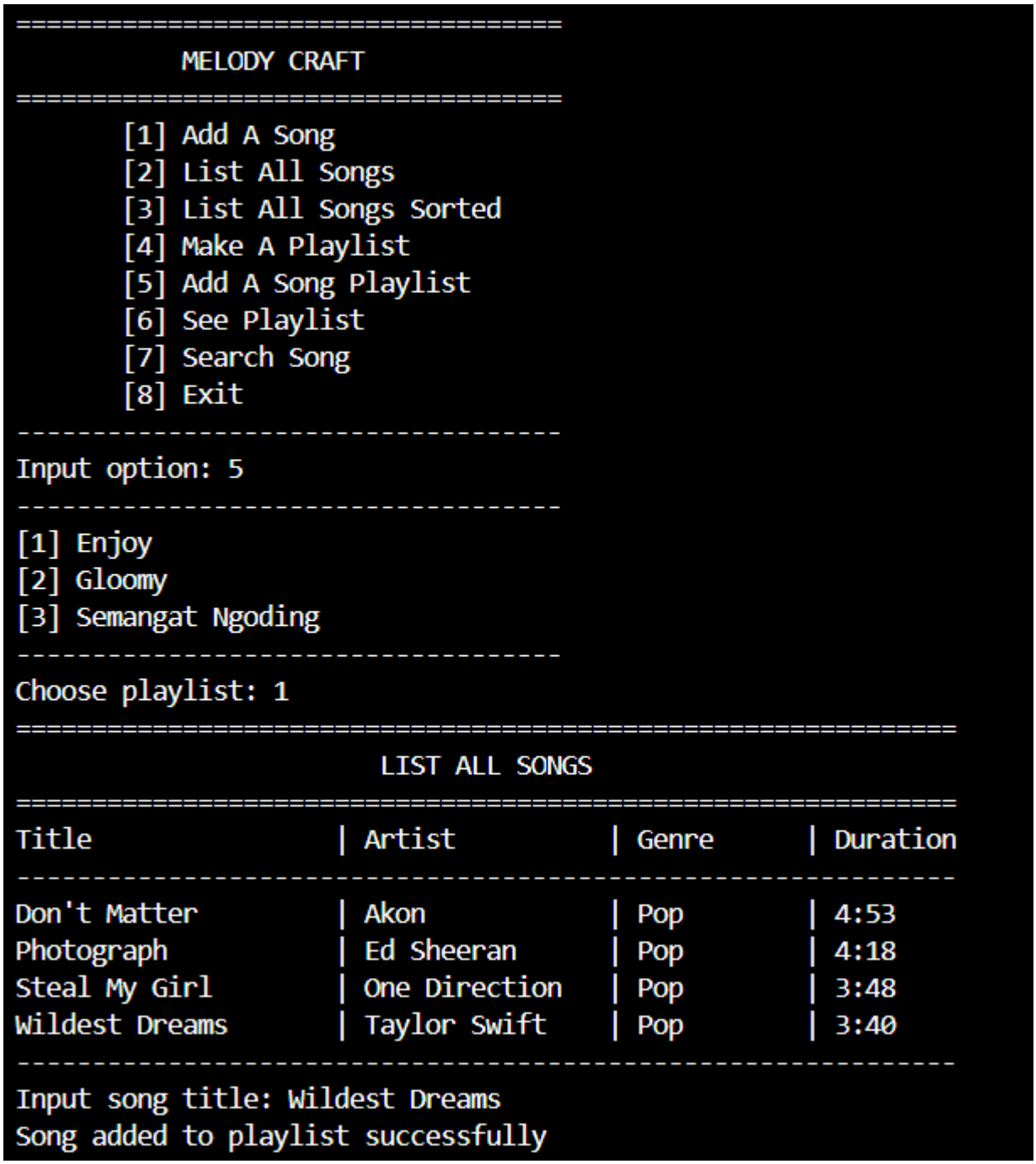
Ketika *user* memilih opsi ke-6 untuk melihat daftar *playlist*, sistem akan menampilkan seluruh *playlist* yang tersedia. Pada gambar di atas terlihat bahwa belum ada *playlist* yang dibuat sehingga sistem menampilkan pesan bahwa tidak ada *playlist* yang tersedia dan *user* perlu membuat *playlist* terlebih dahulu.

Ketika *user* memilih opsi ke-5 untuk menambahkan lagu ke *playlist*, sistem akan meminta memilih *playlist* yang tersedia. Pada gambar di atas terlihat bahwa belum ada *playlist* yang dibuat sehingga sistem menampilkan pesan bahwa belum ada *playlist* yang tersedia seperti halnya opsi ke-6.

Opsi ke-4 digunakan untuk membuat *playlist* baru. Pada gambar di atas, *user* mencoba memasukkan nama *playlist* "Enjoy". Setelah itu, sistem akan menampilkan pesan bahwa *playlist* berhasil dibuat, tetapi *playlist* tersebut masih kosong karena belum ada lagu yang ditambahkan ke dalamnya.

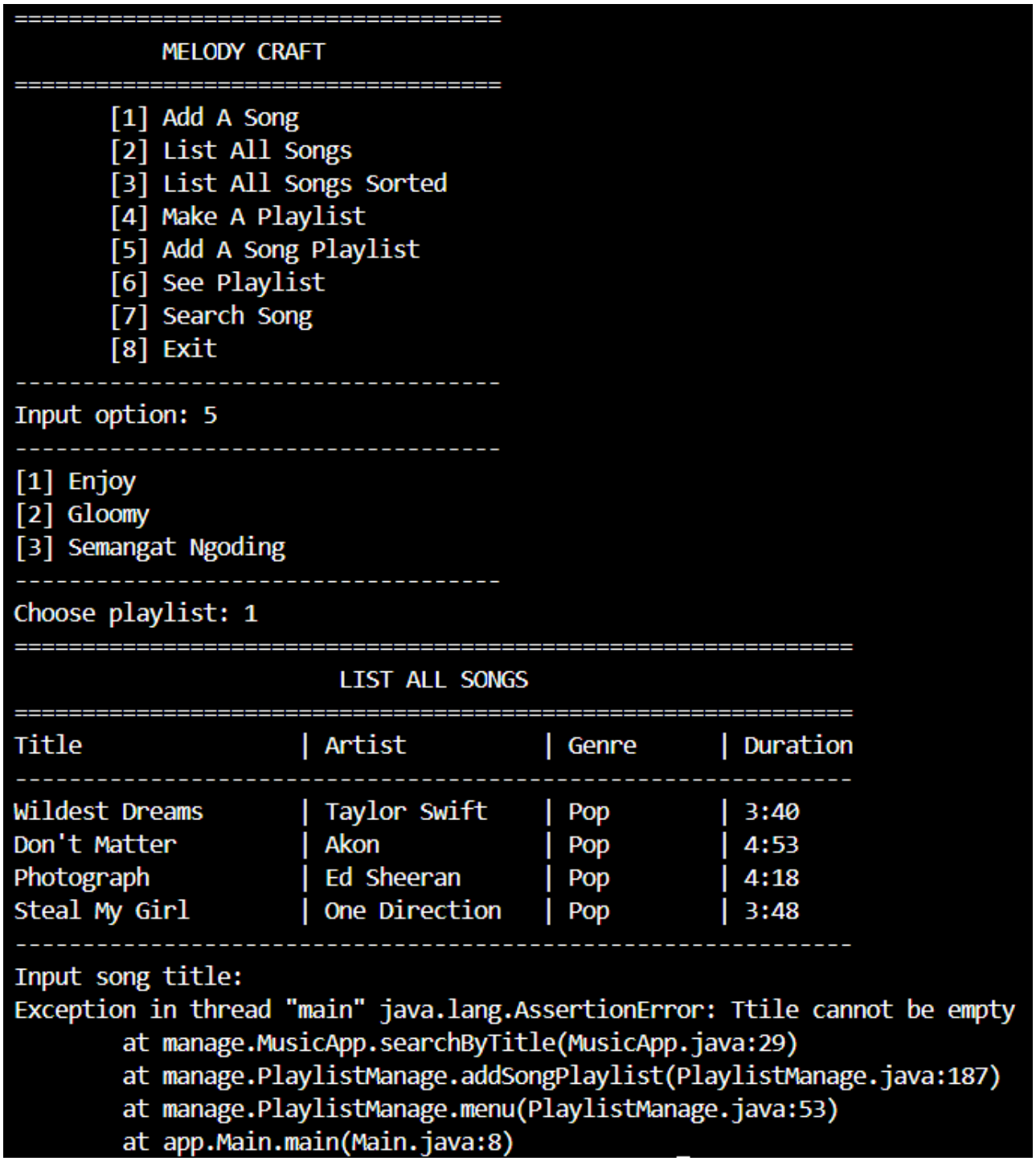
Gambar di atas merupakan contoh ketika *user* mencoba untuk mengosongkan nama *playlist* ketika ingin membuat *playlist* baru. Karena nama *playlist* tidak boleh kosong, maka sistem akan menampilkan pesan bahwa nama *playlist* tidak boleh kosong. Kemudian, sistem akan kembali ke menu utama.

Ketika *user* memilih opsi ke-6 lagi untuk melihat *playlist*, sistem akan menampilkan daftar semua *playlist* yang telah ditambahkan sebelumnya. Pada gambar di atas, terlihat tiga *playlist* yang ditampilkan yaitu *Enjoy*, *Gloomy*, dan Semangat Ngoding. Kemudian, sistem meminta *user* untuk memilih salah satu *playlist*, misalnya *user* memilih *playlist Enjoy*, sistem akan menampilkan daftar lagu pada *playlist* tersebut. Karena belum ada lagu yang ditambahkan ke dalam *playlist* tersebut, maka sistem menampilkan pesan bahwa belum ada lagu yang tersedia di dalam *playlist* tersebut.



Ketika *user* memilih opsi ke-5 lagi, untuk menambahkan lagu ke dalam *playlist*, sistem akan menampilkan daftar *playlist* yang dapat dipilih untuk menambahkan lagu ke dalamnya. Pada gambar di atas, *user* memilih *playlist Enjoy*. Sistem kemudian menampilkan seluruh daftar lagu yang tersedia. *User* harus memasukkan salah satu judul lagu sesuai dengan daftar lagu yang tersedia, misalnya *user input* judul lagu "Wildest Dreams", sistem menampilkan pesan bahwa lagu berhasil ditambahkan ke dalam *playlist* tersebut.

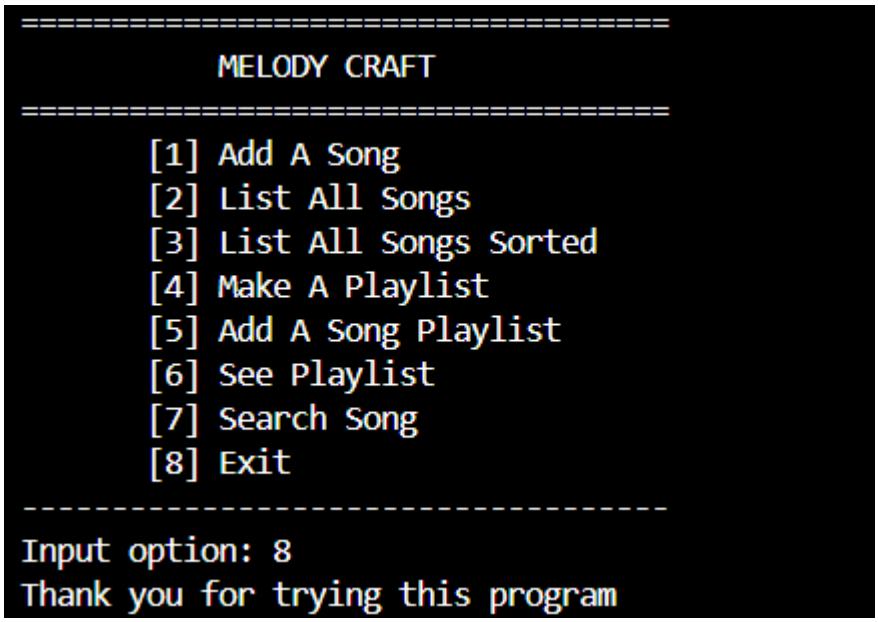
Gambar di atas sebagai contoh ketika *user* mencoba *input* judul lagu tidak sesuai dengan lagu yang tersedia di dalam daftar lagu. Misalnya *user input* judul lagu "the night is still young", sistem akan menampilkan pesan judul tersebut tidak ditemukan di dalam daftar lagu.



Gambar di atas merupakan contoh ketika *user* mencoba untuk mengosongkan *input* saat menambahkan lagu ke dalam *playlist*. Karena judul lagu yang dimasukkan tidak boleh kosong, sistem akan mengeluarkan *assertion error* dengan pesan bahwa judul tidak boleh kosong dan program akan dihentikan.

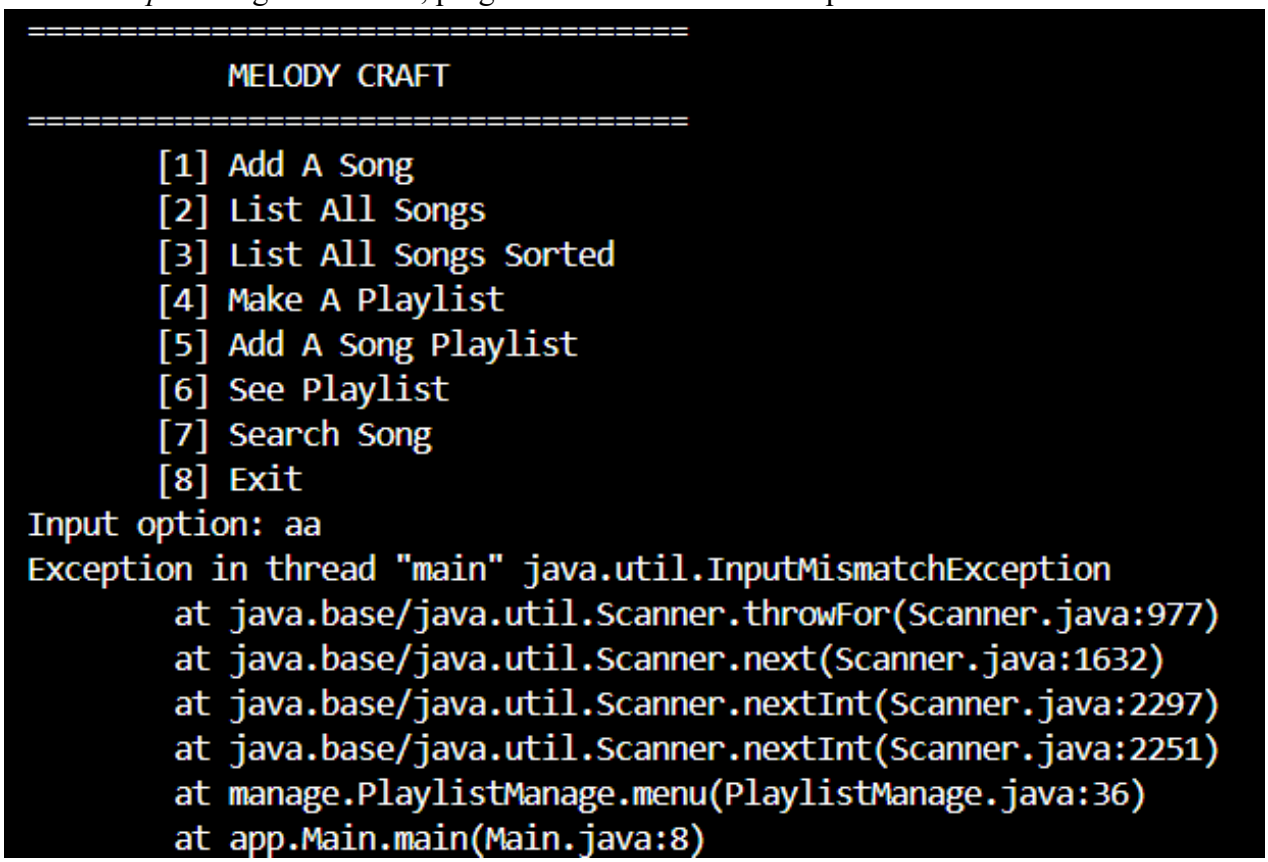
Gambar di atas merupakan hasil dari lagu yang telah berhasil ditambahkan ke dalam *playlist*. Terlihat bahwa lagu “Wildest Dreams” yang sebelumnya ditambahkan melalui opsi ke-5 ditampilkan di dalam *playlist Enjoy* beserta informasi dari lagu tersebut.

Gambar di atas merupakan contoh ketika *user* memasukkan nomor *playlist* yang tidak tersedia, misalnya angka 4. Sistem kemudian menampilkan pesan bahwa *input* yang dimasukkan tidak valid dan setelah itu, sistem kembali ke menu utama.



Opsi ke-8 digunakan untuk menghentikan program. Ketika *user* mencoba memilih opsi tersebut, maka sistem menghentikan semua proses *input* dan keluar dari program. Namun, sebelum itu akan ditampilkan pesan "*Thank you for trying this program*".

Ketika *user* mencoba memasukkan *input* yang tidak tersedia pada menu, misalnya *input* angka 9, sistem akan menampilkan pesan bahwa opsi yang dimasukkan tidak valid dan meminta *user* untuk mencoba *input* ulang. Setelah itu, program akan kembali menampilkan menu utama.



Gambar di atas merupakan contoh ketika *user* memasukkan *input* yang bukan angka, misalnya "aa" pada menu utama. Karena program meminta *input* berupa angka, sistem akan mengeluarkan *InputMismatchException* yang menunjukkan bahwa validasi tipe *input* tidak terpenuhi dan program tidak dapat memproses *input* tersebut sehingga program dihentikan.

Fitur-Fitur:

1. **Folder music**
   a. **Song.java**

```java
package music; //di dalam folder music

public class Song {
    private String title;
    private Artist artist;
    private String duration;

    //constructor
    public Song (String title, Artist artist, String duration) {
        //validasi format durasi menggunakan assertion
        assert duration.matches(regex: "\\d{1,2}:\\d{2}") : "Duration must be in mm:ss format";
        this.title = title;
        this.artist = artist;
        this.duration = duration;
    }
```

Kelas *Song* digunakan merepresentasikan sebuah lagu dengan tiga atribut utama yang dienkapsulasi yaitu *title, artist,* dan *duration*. *Constructor* "Song (String title, Artist artist, String duration)" digunakan untuk menginisialisasi objek lagu baru sekaligus melakukan validasi format durasi menggunakan *assertion*. Pernyatan "assert duration.matches("\\d{1,2}:\\d{2}")" memastikan bahwa durasi yang dimasukkan sesuai dengan format "mm:ss". Sementara itu. "package music" menandakan bahwa kelas *Song* berada di dalam folder "music".

```java
    //getter
    public String getTitle() {
        return title;
    }
    public Artist getArtist() {
        return artist;
    }
    public String getDuration() {
        return duration;
    }
}
```

Getter adalah metode yang digunakan untuk mengambil nilai dari atribut yang bersifat private sehingga tidak mengubah nilai di dalamnya. *Method* "getTitle()" digunakan untuk mengambil judul lagu, "getArtist()" digunakan untuk mengambil objek artis, dan "getDuration()" digunakan untuk mengambil durasi lagu.

   b. **Artist.java**

```java
package music; //di dalam folder music

public class Artist {
    private String name;
    private String genre;

    //constructor
    public Artist (String name, String genre) {
        this.name = name;
        this.genre = genre;
    }
```

Kelas *Artist* digunakan merepresentasikan seorang artis dengan dua atribut utama yang dienkapsulasi yaitu *name* dan *genre*. *Constructor* "Artist (String name, String genre)" digunakan untuk menginisialisasi objek artist baru. Sementara itu. "package music" menandakan bahwa kelas *Artist* berada di dalam folder "music".

```java
    //getter
    public String getName() {
        return name;
    }
    public String getGenre() {
        return genre;
    }
}
```

Getter adalah metode yang digunakan untuk mengambil nilai dari atribut yang bersifat private sehingga tidak mengubah nilai di dalamnya. *Method* "getName()" digunakan untuk mengambil nama artis dan "getGenre()" digunakan untuk mengambil genre dari artis tersebut.

c. Playlist.java

```java
package music;              //di dalam folder music
import java.io.File;        //library utk input output file
import java.util.ArrayList; //library utk struktur data ArrayList
import java.util.Scanner;   //library utk input user


public class Playlist {
    private String name;
    private ArrayList<Song> songs = new ArrayList<>();

    //constructor
    public Playlist (String name) {
        this.name = name;
    }
```

Kelas *Playlist* digunakan untuk menyimpan dan mengelola sebuah *playlist* yang memiliki dua atribut, yaitu *name* untuk menyimpan nama *playlist* dan *songs* digunakan untuk menyimpan banyak objek "Song" dengan *ArrayList*. *Constructor* "Playlist (String name)" digunakan untuk memberi nama awal pada *playlist* ketika pertama kali. Sementara itu, bagian *import* digunakan untuk mengimpor *library* yang dibutuhkan pada program, seperti "java.io.File" untuk operasi *input output file*, "java.util.ArrayList" untuk penyimpanan dinamis, dan "java.util.Scanner" untuk menerima *input* dari *user.* Selain itu, "package music" menandakan bahwa kelas *Playlist* berada di dalam folder "music".

```java
    //getter
    public String getName() {
        return name;
    }
    public ArrayList<Song> getSongs() {
        return songs;
    }

    //method utk menambah lagu ke playlist
    public void addSong (Song song) {
        songs.add(song);
    }
```

Getter adalah metode yang digunakan untuk mengambil nilai dari atribut yang bersifat private sehingga tidak mengubah nilai di dalamnya. *Method* "getName()" digunakan untuk mengambil nama *playlist* dan "getSongs()" digunakan untuk mengambil seluruh daftar lagu yang tersimpan di dalam *playlist*. Selain itu, *method* "addSong (Song song)" digunakan untuk menambahkan objek "Song" baru ke dalam *playlist*.

```java
    //method utk menampilkan isi playlist
    public void displayPlaylist() {
        System.out.println(x: "============================================================");
        System.out.println("                  Playlist " + name);
        System.out.println(x: "============================================================");

        //jika playlist kosong
        if (songs.isEmpty()) {
            System.out.println(x: "Empty songs");
            return;
        }
        //jika playlist tidak kosong
        else {
            System.out.printf(format: "%-20s | %-15s | %-10s | %-5s\n", ...args: "Title", "Artist", "Genre", "Duration");
            System.out.println(x: "------------------------------------------------------------");
            for (Song s: songs) {
                System.out.printf(format: "%-20s | %-15s | %-10s | %-5s\n", s.getTitle(), s.getArtist().getName(), s.getArtist().getGenre(), s.getDuration());
            }
        }
        System.out.println(x: "------------------------------------------------------------");
    }
```

Method "displayPlaylist()" digunakan untuk menampilkan isi *playlist* beserta semua lagu di dalamnya. Pada bagian awal, program akan mencetak *header playlist*, lalu melakukan pengecekan menggunakan *if* untuk memastikan apakah *playlist* kosong. Jika kosong maka akan ditampilkan pesan bahwa tidak ada lagu yang tersimpan. Jika tidak kosong, maka akan ditampilkan semua lagu beserta identitasnya menggunakan perulangan *for*.

```java
    //method untuk memuat file
    public void loadFile() {
        try {
            //playlist disimpan di dalam folder "playlists"
            File file = new File("playlists/" + name + ".txt");
            if (!file.exists()) {
                return;
            }

            songs.clear(); //berisihkan daftar lagu

            Scanner sc = new Scanner(file); //input file
            while (sc.hasNextLine()) {
                String line = sc.nextLine();
                String[] parts = line.split(regex: "\\|");

                //buat object ulang dari file
                Artist a = new Artist(parts[1], parts[2]);
                Song s = new Song(parts[0], a, parts[3]);

                songs.add(s); //tambahkan ke list
            }
            sc.close();
        } catch (Exception e) { //jika gagal memuat file
            System.out.println("Failed load file: " + e.getStackTrace());
        }
    }
}
```

Method "loadFile" digunakan untuk memuat kembali data *playlist* dari *file* yang tersimpan di dalam folder "playlists". Pertama, program akan mengecek jika *file* dengan nama *playlist* tersebut ada. Jika tidak ada, maka proses dihentikan. JIka ada, daftar lagu akan dibersihkan lalu setiap baris *file* dibaca menggunakan *Scanner*, kemudian dipisahkan dengan "split("\\|")" agar dapat membentuk ulang objek "Artist" dan "Song". Setelah objek berhasil dibuat, lagu tersebut dimasukkan kembali ke dalam *ArrayList*, lalu *input* ditutup dengan "sc.close()". Sementara itu, blok *try-catch* digunakan untuk menangani kemungkinan error saat *file* dibaca.

2. Folder manage

a. MusicApp.java

```java
package manage; //di dalam folder manage
import music.*; //import folder music

import java.io.File;        //library utk input output file
import java.io.FileWriter;  //library utk menulis ke file
import java.util.ArrayList; //library utk struktur data ArrayList
import java.util.Scanner;   //library utk input user

public class MusicApp {
    private ArrayList<Song> listSongs = new ArrayList<>();
    private ArrayList<Artist> listArtists = new ArrayList<>();
```

Kelas *MusicApp* digunakan untuk mengelola daftar semua lagu dan daftar semua artis yang memiliki dua atribut, yaitu *listSongs* untuk menyimpan banyak objek "Song" dan *listArtists* digunakan untuk menyimpan banyak objek "Artist", keduanya menggunakan *ArrayList* agar dapat menyimpan data secara dinamis. Bagian *import* digunakan untuk mengimpor *library* yang dibutuhkan, seperti "java.io.File" dan "java.io.FileWriter" untuk operasi *read* dan *write* pada *file*. "java.util.ArrayList" digunakan untuk penyimpanan data secara dinamis, serta "java.util.Scanner" untuk menerima *input* dari *user*. Selain itu, "package manage" menandakan bahwa kelas *MusicApp* berada di dalam folder "manage", sedangkan "import music.*" digunakan untuk mengimpor semua kelas yang berada di dalam folder "music".

```java
    //menambahkan artis ke list
    public void addArtist (Artist a) {
        listArtists.add(a);
    }
    //menambahkan lagu ke list
    public void addSong (Song s) {
        listSongs.add(s);
    }
    //getter
    public ArrayList<Song> getAllSongs() {
        return listSongs;
    }
```

Method "addArtist()" digunakan untuk menambahkan objek "Artist" ke dalam daftar artis, sedangkan "addSong()" digunakan untuk menambahkan objek "Song" ke dalam daftar lagu. Selain itu, *getter* "getAllSongs()" digunakan untuk mengambil seluruh daftar lagu tanpa mengubah isi di dalamnya karena atribut bersifat private.

```java
//mencari judul lagu berdasarkan judul
public Song searchByTitle (String title) throws Exception {
    //memastikan input judul tidak kosong
    assert title != null && !title.trim().isEmpty(): "Ttile cannot be empty";
    //perulangan untuk mencari judul yg sesuai
    for (Song s: listSongs) {
        //jika terdapat judul yang sama
        if (s.getTitle().equalsIgnoreCase(title)) {
            return s; //mengembalikan lagu
        }
    }
    //jika tidak ditemukan judul yang sama
    throw new Exception(message: "Not found the song");
}
```

Method "searchByTitle()" digunakan untuk mencari lagu berdasarkan judul yang dimasukkan oleh *user*. Pertama, dilakukan validasi menggunakan *assert* agar *input* judul tidak boleh kosong. Kemudian, program melakukan perulangan *for* pada *listSongs* untuk membandingkan setiap judul lagu menggunakan "equalsIgnoreCase()" agar tidak sensitif terhadap huruf besar atau kecil. Jika ditemukan judul yang sama, *method* akan mengembalikan objek "Song" tersebut. Jika tidak ada judul yang sama hingga perulangan selesai, program akan melempar *Exception* dengan pesan bahwa lagu tidak ditemukan.

```java
//menampilkan semua daftar lagu
public void listAllSongs() {
    System.out.println(x: "============================================================");
    System.out.println(x: "                      LIST ALL SONGS                        ");
    System.out.println(x: "============================================================");

    //jika daftar lagu kosong
    if (listSongs.isEmpty()) {
        System.out.println(x: "Empty song");
        return;
    }
    //jika terdapat lagu
    else {
        System.out.printf(format: "%-20s | %-15s | %-10s | %-5s\n", ...args: "Title", "Artist", "Genre", "Duration");
        System.out.println(x: "------------------------------------------------------------");
        for (Song s: listSongs) {
            System.out.printf(format: "%-20s | %-15s | %-10s | %-5s\n", s.getTitle(), s.getArtist().getName(), s.getArtist().getGenre(), s.getDuration());
        }
    }
    System.out.println(x: "------------------------------------------------------------");
}
```

Method "listAllSongs()" digunakan untuk menampilkan semua lagu yang telah ditambahkan sebelumnya. Pada bagian awal, program akan mencetak *header*, lalu melakukan pengecekan menggunakan *if* untuk memastikan apakah terdapat lagu yang disimpan. Jika kosong maka akan ditampilkan pesan bahwa tidak ada lagu yang tersimpan. Jika tidak kosong, maka akan ditampilkan semua lagu beserta identitasnya menggunakan perulangan *for*.

```java
//menampilkan semua daftar lagu secara terurut (dari A - Z)
public void listSongsSorted() {
    //melakukan sorting dgn membandingkan huruf pertama dri judul lagu
    listSongs.sort((a,b) -> {
        //mengambil huruf pertama
        char c1 = a.getTitle().toUpperCase().charAt(index: 0);
        char c2 = b.getTitle().toUpperCase().charAt(index: 0);
        //membandingkan dua karakter utk mengurutkan
        return Character.compare(c1, c2);
    });
    listAllSongs();
}
```

Method “listSongsSorted()” digunakan untuk menampilkan seluruh daftar lagu dalam kondisi sudah urut dari A sampai Z. Proses pengurutan dilakukan dengan memanfaatkan *method sort()* pada *ArrayList* dan sebuah *lambda comparator* yang membandingkan karakter pertama dari setiap judul lagu setelah diubah menjadi huruf kapital. Setelah proses *sorting* selesai, *method* “listAllSongs()” dipanggil untuk menampilkan hasilnya.

```
76    //menyimpan seluruh lagu ke file ListSongs.txt
77    public void saveAllSongs() {
78        try {
79            FileWriter writer = new FileWriter(fileName: "ListSongs.txt");
80            //perulangan utk ditulis di dalam file
81            for (Song s: listSongs) {
82                writer.write(s.getTitle() + "|"
83                        + s.getArtist().getName() + "|"
84                        + s.getArtist().getGenre() + "|"
85                        + s.getDuration() + "\n");
86            }
87            writer.close();
88        } catch (Exception e) { //jika gagal menyimpan lagu
89            System.out.println("Failed to save file: " + e.getStackTrace());
90        }
91    }
```

Method “saveAllSongs()” digunakan untuk menyimpan seluruh lagu ke dalam *file* bernama “ListSongs.txt”. Prosesnya dilakukan dengan membuat objek *FileWriter*, kemudian setiap lagu dengan format yang telah ditentukan menggunakan perulangan *for*. Setelah semua daftar lagu ditulis, *file* ditutup “writer.close()”. Jika proses penulisan ke dalam *file* gagal, blok *catch* akan menangani error dan menampilkan pesan kegagalan.

```java
//method untuk memuat seluruh lagu dari ListSongs.txt
public void loadSongs() {
    try {
        File file = new File(pathname: "ListSongs.txt");
        //jika file tidak ditemukan
        if (!file.exists()) {
            return;
        }

        Scanner sc = new Scanner(file); //input file
        while (sc.hasNextLine()) {
            String line = sc.nextLine();
            String[] parts = line.split(regex: "\\|");

            //buat object ulang dari file
            Artist a = new Artist(parts[1], parts[2]);
            Song s = new Song(parts[0], a, parts[3]);

            //tambahkan ke dalam list
            addArtist(a);
            addSong(s);
        }
        sc.close();
    } catch (Exception e) { //jika gagal memuat file
        System.out.println("Failed to load file: " + e.getStackTrace());
    }
}
```

Method "loadSongs()" digunakan untuk memuat kembali seluruh lagu dari *file* "ListSongs.txt" ke dalam program. Prosesnya dimulai dari mengecek apakah *file* tersebut ada. Jika tidak ditemukan, *method* langsung dihentikan. Jika *file* ada, program akan membaca setiap baris dengan *Scanner*, kemudian dipisahkan menggunakan "split("\\|")" agar dapat membentuk ulang objek "Artist" dan "Song". Setelah objek berhasil dibuat, lagu tersebut dimasukkan kembali ke dalam *ArrayList*, lalu *input* ditutup dengan "sc.close()". Sementara itu, blok *try-catch* digunakan untuk menangani kemungkinan error saat *file* dibaca.

b. PlaylistManage.java

```java
package manage; //di dalam folder manage
import music.*; //import folder music

import java.io.File;        //library utk input output file
import java.io.FileWriter;  //library utk menulis ke file
import java.util.ArrayList; //library utk struktur data ArrayList
import java.util.Scanner;   //library utk input user

public class PlaylistManage {
    private MusicApp app = new MusicApp();
    private ArrayList<Playlist> playlists = new ArrayList<>();
    private Scanner sc = new Scanner(System.in);

    //constructor
    public PlaylistManage() {
        app.loadSongs(); //memuat daftar lagu
        loadPlaylists(); //memuat daftar playlist
    }
```

Kelas *PlaylistManage* digunakan untuk mengelola semua fitur program Melody Craft yang memiliki tiga atribut, yaitu objek *MusicApp* untuk mengakses dan memproses data lagu, *ArrayLits <playlist> playlists* digunakan untuk menyimpan banyak *playlist* yang dibuat oleh *user*, dan *Scanner* untuk menerima *input user*. *Constructor* "PlaylistManage()" digunakan untuk memanggil *method* "app.loadSongs() yang memuat seluruh lagu yang sudah disimpan dan "loadPlaylists()" yang memuat semua *playlist* yang tersedia. Bagian *import* digunakan untuk mengimpor *library* yang dibutuhkan, seperti "java.io.File" dan "java.io.FileWriter" untuk operasi *read* dan *write* pada *file*. "java.util.ArrayList" digunakan untuk penyimpanan data secara dinamis, serta "java.util.Scanner" untuk menerima *input* dari *user*. Selain itu, "package manage" menandakan bahwa kelas *MusicApp* berada di dalam folder "manage", sedangkan "import music.*" digunakan untuk mengimpor semua kelas yang berada di dalam folder "music".

```java
    //menu utama
    public void menu() {
        while (true) {
            System.out.println(x: "====================================");
            System.out.println(x: "           MELODY CRAFT             ");
            System.out.println(x: "====================================");
            System.out.println(x: "       [1] Add A Song               ");
            System.out.println(x: "       [2] List All Songs           ");
            System.out.println(x: "       [3] List All Songs Sorted    ");
            System.out.println(x: "       [4] Make A Playlist          ");
            System.out.println(x: "       [5] Add A Song Playlist      ");
            System.out.println(x: "       [6] See Playlist             ");
            System.out.println(x: "       [7] Search Song              ");
            System.out.println(x: "       [8] Exit                     ");
            System.out.println(x: "------------------------------------");
            System.out.print(s: "Input option: ");
            int option = sc.nextInt();
            sc.nextLine(); //membuang newline
```

Method “menu()” digunakan untuk menampilkan menu utama aplikasi Melody Craft yang dapat diakses oleh *user*. Di dalamnya, terdapat perulangan *while(true)* agar menu dapat terus ditampilkan sampai *user* memilih opsi *exit*, serta terdapat delapan pilihan yang dapat dipilih oleh *user*. “int option = sc.nextInt()” digunakan untuk membaca *input* dari *user*, sedangkan “sc.nextLine()” digunakan untuk membersihkan *buffer* setelah membaca angka.

```
39          switch (option) {
40              case 1:
41                  addSong();
42                  break;
43              case 2:
44                  app.listAllSongs();
45                  break;
46              case 3:
47                  app.listSongsSorted();
48                  break;
49              case 4:
50                  createPlaylist();
51                  break;
52              case 5:
53                  addSongPlaylist();
54                  break;
55              case 6:
56                  seePlaylist();
57                  break;
58              case 7:
59                  findSong();
60                  break;
61              case 8:
62                  System.out.println(x: "Thank you for trying this program");
63                  return;
64              default:
65                  System.out.println(x: "Invalid option");
66                  break;
67          }
68      }
```

Bagian *switch-case* digunakan untuk memilih dan menjalankan perintah tertentu sesuai opsi yang dipilih oleh *user*. Pada *case 1*, program akan memanggil *method* “addSong()” untuk menambahkan lagu. *Case 2* digunakan untuk menampilkan seluruh daftar lagu. *Case 3* digunakan untuk menampilkan seluruh daftar lagu juga, tetapi dalam bentuk terurut. *Case 4* digunakan untuk membuat *playlist*. *Case 5* digunakan untuk menambahkan lagu yang berasal dari daftar semua lagu ke dalam *playlist*. *Case 6* digunakan untuk melihat *playlist* beserta lagu di dalamnya. *Case 7* digunakan untuk mencari sebuah lagu berdasarkan judul. *Case 8* digunakan untuk keluar dari program. Sementara itu, *default* digunakan untuk menangani *input* yang tidak valid dari *user* dan akan kembali ke menu utama.

```java
    //method utk menyimpan playlist ke file
    public void savePlaylist (Playlist p) {
        try {
            //memastikan jika folder playlists ada
            File folder = new File(pathname: "playlists");
            //jika belum ada maka akan dibuat
            if (!folder.exists()) {
                folder.mkdir();
            }

            //menulis ulang file playlistnya
            FileWriter writer = new FileWriter("playlists/" + p.getName() + ".txt");
            //loop semua lagu dalam playlist
            for (Song s: p.getSongs()) {
                writer.write(s.getTitle() + "|"
                        + s.getArtist().getName() + "|"
                        + s.getArtist().getGenre() + "|"
                        + s.getDuration() + "\n");
            }
            writer.close();
        } catch (Exception e) { //jika gagal menyimpan playlist
            System.out.println("Failed to save playlists: " + e.getMessage());
        }
    }
```

Method "savePlaylist()" dengan satu parameter (*Playlist p*) digunakan untuk menyimpan seluruh lagu dalam sebuah *playlist* ke dalam *file*. Pertama program akan memastikan bahwa folder *playlists* sudah ada dan jika tidak ada, maka folder tersebut akan dibuat. Setelah itu, *file playlist* ditulis ulang menggunakan *FileWriter*, kemudian setiap lagu dalam *playlist* dituliskan ke dalam *file* dengan format yang telah ditentukan menggunakan perulangan *for*, lalu ditutup dengan "writer.close()". Sementara itu, blok *try-catch* digunakan untuk menangani kemungkinan error saat menyimpan isi *file*.

```java
96      //method untuk menambahkan lagu baru
97      public void addSong() {
98          System.out.println(x: "-----------------------------------");
99          System.out.print(s: "Title\t: ");
100         String title = sc.nextLine();
101
102         System.out.print(s: "Artist\t: ");
103         String artist = sc.nextLine();
104
105         System.out.print(s: "Genre\t: ");
106         String genre = sc.nextLine();
107
108         System.out.print(s: "Duration (mm:ss): ");
109         String duration = sc.nextLine();
110         System.out.println(x: "-----------------------------------");
111
112         //buat objek baru artist dan song
113         Artist a = new Artist(artist, genre);
114         Song s = new Song(title, a, duration);
115
116         //tambahkan ke list song
117         app.addArtist(a);
118         app.addSong(s);
119         //simpan semua lagu ke dalam file
120         app.saveAllSongs();
121
122         System.out.println(x: "Song added successfully");
123     }
```

Method "addSong()" digunakan untuk menambahkan lagu baru ke dalam daftar lagu. Pertama program akan meminta *user* untuk memasukkan judul lagu, nama artis, genre, dan durasi. Kemudian data tersebut digunakan untuk membuat objek *Artist* dan *Song*. Setelah objek berhasil dibuat, lagu beserta artisnya ditambahkan ke dalam *list* melalui *method* "addArtist()" dan "addSong()", lalu seluruh data lagu disimpan ke dalam *file* menggunakan *method* "saveAllSongs()".

```java
//membuat playlist baru
public void createPlaylist() {
    System.out.println(x: "-----------------------------------");
    System.out.print(s: "Playlist name: ");
    String name = sc.nextLine();

    //validasi jika nama playlist kosong
    if (name.isEmpty()) {
        System.out.println(x: "Playlist name cannot be empty");
        return;
    }

    //mengecek apakah playlist dengan nama yang sama sudah ada
    for (Playlist p: playlists) {
        //jika nama sudah ada
        if (p.getName().equalsIgnoreCase(name)) {
            System.out.println(x: "Playlist alredy exists");
            return;
        }
    }

    //jika belum, buat playlist baru
    Playlist p = new Playlist(name);
    playlists.add(p);
    //simpan playlist kosongnya
    savePlaylist(p);
    System.out.println(x: "Playlist created successfully (empty)");
}
```

Method "createPlaylist()" digunakan untuk membuat *playlist* baru berdasarkan nama yang dimasukkan *user*. Pertama akan dilakukan validasi *input user* jika nama yang dimasukkan kosong. Kemudian, akan dicek apakah sudah ada *playlist* lain dengan nama yang sama di dalam daftar. Jika nama yang dimasukkan valid dan belum digunakan, program akan membuat objek *Playlist* baru dan menambahkannya ke dalam *list playlists*, lalu menyimpannya sebagai *file* kosong menggunakan *method* "savePlaylist()".

```
155     public void addSongPlaylist() {
156         System.out.println(x: "-----------------------------------");
157         //jika tidak ada playlist
158         if (playlists.isEmpty()) {
159             System.out.println(x: "No playlist available");
160             return;
161         }
162         //tampilkan list playlist yg ada
163         for (int i = 0; i < playlists.size(); i++) {
164             System.out.println("[" + (i + 1) + "] " + playlists.get(i).getName());
165         }
166         System.out.println(x: "-----------------------------------");
167 
168         System.out.print(s: "Choose playlist: ");
169         int option = sc.nextInt() - 1;
170         sc.nextLine();
171         //cek validasi
172         if (option < 0 || option >= playlists.size()) {
173             System.out.println(x: "Invalid playlist number");
174             return;
175         }
```

Method "addSongPlaylist()" digunakan untuk menambahkan lagu ke dalam *playlist* yang sudah ada. Pertama program akan mengecek apakah *playlist* ada. Jika tidak ada, program akan kembali ke menu utama. Jika ada, maka akan ditampilkan seluruh *playlist* yang tersedia. Selanjutnya, *user* diminta untuk memasukkan pilihan *playlist*. Setelah itu, terdapat validasi jika *user* memasukkan angka negatif atau angka 0.

```
177         Playlist select = playlists.get(option); //object
178         //load isi file playlist ke memori
179         select.loadFile();
180         //list semua lagu yg ada
181         app.listAllSongs();
182         //input judul lagu yg ingin ditambahkan
183         System.out.print(s: "Input song title: ");
184         String title = sc.nextLine();
185         try {
186             //cari lagu berdasarkan judul
187             Song chosen = app.searchByTitle(title);
188 
189             //tambahkan ke playlist
190             select.addSong(chosen);
191             //simpan ulang file playlist
192             savePlaylist(select);
193             System.out.println(x: "Song added to playlist successfully
194         } catch (Exception e) { //jika lagu tidak ditemukan
195             System.out.println(x: "Song not found");
196         }
197     }
```

Setelah *playlist* dipilih, program akan memuat isi *playlist* dari *file* dan menampilkan seluruh

lagu yang tersedia di dalam program, lalu *user* memasukkan judul lagu yang ingin ditambahkan. Judul lagu yang dimasukkan akan dicari terlebih dahulu menggunakan *method* "searchByTitle()". Jika ditemukan, maka lagu tersebut ditambahkan ke dalam *playlist*. Sementara itu, blok *try-catch* digunakan untuk menangani kemungkinan error saat judul lagu dicari atau judul tersebut tidak ditemukan.

```
199     //method utk mencari judul lagu
200     public void findSong() {
201         System.out.println(x: "-----------------------------------");
202         System.out.print(s: "Title: ");
203         String title = sc.nextLine();
204         try {
205             System.out.println(x: "-----------------------------------");
206             //cari musik
207             Song found = app.searchByTitle(title);
208             //menampilkan info lengkap
209             System.out.println("Title\t: " + found.getTitle());
210             System.out.println("Artist\t: " + found.getArtist().getName());
211             System.out.println("Genre\t: " + found.getArtist().getGenre());
212             System.out.println("Duration: " + found.getDuration());
213         } catch (Exception e) {
214             System.out.println(e.getMessage());
215         }
216     }
```

Method "findSong()" digunakan untuk mencari lagu berdasarkan judul yang dimasukkan oleh *user*. Pertama, program akan meminta *user* untuk memasukkan judul lagu, kemudian memanggil *method* "searchByTitle()" untuk melakukan pencarian. Jika lagu ditemukan, program akan menampilkan informasi lengkap dari lagu tersebut. Jika lagu tidak ditemukan atau judul tidak valid, blok *catch* akan menangkap error dan menampilkan pesan yang sesuai.

```
218     //menampilkan lagu yang ada di dalam playlist
219     public void seePlaylist() {
220         System.out.println(x: "-----------------------------------");
221         //jika tidak ada playlist
222         if (playlists.isEmpty()) {
223             System.out.println(x: "No playlist available");
224             return;
225         }
226 
227         //list semua playlist
228         for (int i = 0; i < playlists.size(); i++) {
229             System.out.println("[" + (i + 1) + "] " + playlists.get(i).getName());
230         }
231         System.out.println(x: "-----------------------------------");
232 
233         System.out.print(s: "Choose Playlist: ");
234         int index = sc.nextInt() - 1;
235         sc.nextLine();
236         //cek validasi
237         if (index < 0 || index >= playlists.size()) {
238             System.out.println(x: "Invalid playlist number");
239             return;
240         }
241 
242         Playlist playlist = playlists.get(index); //object
243         playlist.loadFile();        //muat lagi yang ada di dalam playlist tersebut
244         playlist.displayPlaylist(); //menampilka isi playlistnya
245     }
```

Method “seePlaylist()” digunakan untuk menampilkan isi dari *playlist* yang dipilih oleh *user*. Pertama program akan mengecek apakah *playlist* ada. Jika tidak ada, program akan kembali ke menu utama. Jika ada, maka akan ditampilkan seluruh *playlist* yang tersedia. Selanjutnya, *user* diminta untuk memasukkan pilihan *playlist*. Setelah itu, terdapat validasi jika *user* memasukkan angka negatif atau angka 0. Jika *input user* valid, program akan memuat ulang isi dari *file playlist* menggunakan *method* “loadFile()” dan menampilkannya menggunakan *method* “displayPlaylist()”.

```
247     //memuat semua playlist dari folder playlists
248     public void loadPlaylists() {
249         playlists.clear(); //berishkan daftar playlists
250 
251         File folder = new File(pathname: "playlists");
252         //jika folder tidak ada (blm ada playlists)
253         if (!folder.exists()) {
254             return;
255         }
256 
257         //mengambil semua file .txt dalam folder playlists
258         File[] files = folder.listFiles((dir, name) -> name.endsWith(suffix: ".txt"));
259         if (files == null) {
260             return;
261         }
262         //tiap file = 1 playlist
263         for (File f: files) {
264             String fileName = f.getName().replace(target: ".txt", replacement: "");
265             playlists.add(new Playlist(fileName));
266         }
267     }
268 }
```

Method "loadPlaylists()" digunakan untuk memuat ulang seluruh *playlist* yang ada di folder "playlists". Pertama, daftar *playlist* dalam memori dibersihkan, lalu program mengecek apakah folder "playlists" ada. Jika tidak ada, proses dihentikan. Jika ada, program akan mencari semua *file* "*.txt" di dalam folder tersebut karena setiap *file* mewakili satu *playlist*. Jika daftar *file* valid, setiap nama *file* diambil dan ekstensi ".txt" dihapus, kemudian dibuat objek *Playlist* baru sesuai nama *file* tersebut dan ditambahkan ke dalam daftar *playlist*.

3. Folder app

a. Main.java

```
1   package app;                    //di dalam folder app
2   import manage.PlaylistManage;   //import file PlaylistManage dari folder manage
3 
4   public class Main {
        Run | Debug
5       public static void main(String[] args) throws Exception {
6           //mulai aplikasi
7           PlaylistManage pm = new PlaylistManage(); //Object
8           pm.menu(); //memanggil menu
9       }
10  }
```

Kelas *Main* merupakan kelas yang berfungsi sebagai titik awal eksekusi program. Di dalamnya, dibuat sebuah objek *PlaylistManage* bernama "pm", lalu memanggil *method* "menu()" untuk menampilkan menu utama dan mengatur seluruh interaksi *user*. Bagian *import* digunakan untuk mengimpor *library* yang digunakan oleh program, seperti "manage.PlaylistManage" untuk mengimpor kelas *PlaylistManage* yang ada di dalam folder "manage". Selain itu, *package app* menandakan bahwa kelas *Main* berada di dalam folder "app".

NO	MATERI	PERAN
1.	Class, Object, Method, Package, dan Constructor	*Class* digunakan untuk membuat suatu kelas objek seperti *Song, Artist, Playlist, MusicApp, PlaylistManage*, dan *Main*. *Object* digunakan untuk menyimpan dan memanipulasi data seperti data lagu, artis, dan *playlist*. *Method* digunakan untuk membagi program menjadi bagian-bagian kecil yang memiliki tugas tertentu dan dapat dipanggil berkali-kali. Misalnya dalam program yaitu *addSong(), listAllSongs(), createPlaylist(),* dan lainnya. *Package* digunakan untuk mengelompokkan kelas, seperti di dalam *package* "music" terdapat tiga kelas, yaitu *class Song, class Artist,* dan *class Playlist*. *Constructor* digunakan untuk inisialisasi objek baru, misalnya dalam membuat lagu baru di kelas *Song* dibuat objek baru "public Song (String title, String artist, String duration)".
2.	Variabel, Konstanta, dan Tipe Data	*Variabel* digunakan untuk menyimpan data sementara seperti *String title, int option, ArrayList<Playlist> playlists*, dan lainnya yang digunakan pada program. Konstanta digunakan untuk menyimpan nilai tetap, misal format durasi "mm:ss" yang digunakan pada program. Tipe data, seperti *String* digunakan untuk variabel dalam bentuk teks (judul lagu, nama artis, genre, dan lainnya), *int* untuk variabel opsi, *char* untuk manipulasi huruf saat *sorting*, dan *ArrayList* untuk daftar lagu, artist, dan *playlist*.
3.	Conditional dan Looping	*Conditional* digunakan untuk validasi *input user* (*if else*), mengecek keberadaan *playlist* atau lagu (*if else*), dan mengeksekusi menu yang dipilih (*switch case*). *Looping* bagian *for loop* digunakan untuk menampilkan daftar lagu dan daftar *playlist*, sementara itu bagian *while* digunakan untuk membaca *file* dan mengulang menu hingga *user* memilih opsi *exit*.
4.	Exception dan Assertion	*Exception* digunakan untuk menangani error saat mencari lagu yang tidak ada menggunakan *throws Exception* atau saat gagal memuat dan menyimpan *file* menggunakan *try catch*. *Assertion* digunakan untuk memastikan format durasi lagu sesuai "mm:ss" saat membuat objek *Song* (*assert duration.matches()*) dan memastikan saat *user* mencari lagu berdasarkan judul, judul yang dimasukkan tidak kosong (*assert*

		title != null && !title.trim().isEmpty()).
5.	Char dan String	*Char* digunakan pada *sorting* lagu berdasarkan huruf pertama judul (*.toUpperCase().charAt(0)*) sehingga dapat membandingkan huruf secara alfabet. *String* digunakan untuk menyimpan judul lagu, nama artis, genre, dan durasi. *String pattern* (*,matches(regex)*) digunakan untuk validasi *input* durasi. Selain itu, terdapat manipulasi *string* lain, yaitu (*split("\\|")*).
6.	Array dan Collection	*ArrayList* digunakan untuk menyimpan banyak lagu, artis, dan *playlist* sehingga program dapat mengelola data secara dinamis.
7.	Input dan Output	*Input* dari *user* menggunakan *Scanner* untuk memasukkan judul, artis, genre, durasi, dan pilihan menu. *Output* digunakan untuk menampilkan daftar lagu, *playlist*, detail lagu, dan pesan notifikasi atau kesalahan. *File* I/O yaitu *File, FileWriter, Scanner* digunakan untuk menyimpan dan memuat lagu serta *playlist* ke atau dari *file*.
8.	Enkapsulasi	Enkapsulasi digunakan untuk melindungi data dari akses secara langsung, misalnya *private fields* yang ada di dalam *class Song*, yaitu *private String title, private Artist artist,* dan *private String duration*. *Getter* digunakan hanya untuk mendapatkan data tanpa dapat mengubah nilai data tersebut.