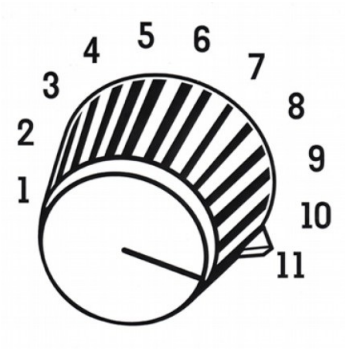
CS 284: Special Assignment 1
Due: 1 April, 11:55pm
(No late submissions allowed!)



1 Assignment Policies

Collaboration Policy. Special Assignment 1 will be done individually: each student must hand in their own answers. It is acceptable for students to collaborate in understanding the material but not in solving the problems or programming. Use of the Internet is allowed, but should not include searching for existing solutions.

Under absolutely no circumstances code can be exchanged between students. Excerpts of code presented in class can be used.

Your code must include a comment with your name and section.

2 Assignment

This assignment consists of implementing a music player. A music player contains a queue of songs, of which the first song is the one currently being played. Since our music player is being developed on a limited budget, the user will only be able to advance to the next song. To increase sales of the players, reporting the total amount of playtime will be offered as a feature to the user. Of course, to create a music player, we must also find a way to represent songs. We care about four attributes:

- The song's name
- The artist's first name
- The artist's last name
- The running time of the song in seconds

Users can share lists of songs with others through playlists. To further reduce the development cost, playlists will be built on top of the `SLL` class, which has been provided for you. To summarize, your job is to:

1. Complete the `Song` class so that songs can be represented.
2. Complete the `Playlist` class so that songs can be stored in a playlist.
3. Use `Song` and `Playlist` to complete the `Player`.

Once you have completed all of the above, corporate will test your solution to see how it performs.

3 Implementation

Since each part of the assignment builds upon previous parts, it is recommended that you tackle them in order, ensuring that all of the required operations work as expected. Errors in earlier parts of the assignment will cause errors in later parts of the assignment and will result in an appropriate deduction in points. The implementation consists of the following 3 classes:

3.1 Song.java

The `Song` class stores information about a song. There are four data fields for storing the name, artist first name, artist last name, and running time of the song in seconds. You must implement the following:

- `public Song(String name, String artistFirstName, String artistLastName, int runTime)`: Creates a new Song setting the Song's `name`, `artistFirstName`, `artistLastName`, and `runTime` properly. Must throw an `IllegalArgumentException` if `name`, `artistFirstName`, or `artistLastName` is empty. Must also throw an `IllegalArgumentException` if `runTime` is equal to or less than 0.

3.2 Playlist.java

The `Playlist` class stores all songs inside of a single-linked list, the implementation of which is handled by the `SLL` class. You must implement the following:

- `public Playlist()`: Creates a new Playlist with no songs.
- `public Playlist(Song song)`: Creates a new Playlist with the supplied song being set to the first one.

- `public int getSize()`: Returns the number of songs currently in the playlist.
- `public Song getAt(int index)`: Returns the song at the supplied index. Must throw an `IllegalArgumentException` if the index is out of bounds.
- `public void addSong(Song song)`: Adds a song to the end of the playlist.
- `public int totalRunTime()`: Returns the total running time of all the songs in the playlist combined. Must throw an `IllegalStateException` if there are no songs in the playlist.
- `public Song longestSong()`: Returns the longest-running song in the playlist. Must throw an `IllegalStateException` if there are no songs in the playlist.

3.3 Player.java

The `Player` class uses a queue to keep track of the songs that it will play. There are four data fields:

1. `private Node<Song> currentSong`, which is the currently playing song. For simplicity, the currently playing song is the first one in the queue.
2. `private Node<Song> lastSong`, which is the last song in the play queue.
3. `private int playQueueSize`, which is the amount of songs in the play queue, including the current one.
4. `private int totalRunTime`, which is a counter of the runtime of all songs that have been played. **It does not reset when the play queue is cleared.**

You must implement the following:

- `public Player()`: Creates a new Player with an empty play queue.
- `public Player(Playlist plist)`: Creates a new Player with all the songs in the supplied playlist loaded into the play queue in the order in which they appear in the playlist (i.e. the first song in the playlist will be the first and current song in the Player). Must throw an `IllegalArgumentException` if the playlist has no songs.
- `public int getTotalRunTime()`: Returns the runtime of the songs that have been played so far.
- `public int getPlayQueueSize()`: Returns the number of songs in the play queue, including the current one.
- `public Song getCurrentSong()`: Returns the song currently being played. Must throw an `IllegalArgumentException` if there is no song currently playing.
- `public ArrayList<Song> getAllSongs()`: Returns an ArrayList of all the songs in the play queue. Must throw an `IllegalStateException` if there are no songs in the play queue at all.

- `public int getRemainingRunTime()`: Returns the runtime of all the songs after the currently playing one in the queue (i.e. how much playing time is left). Must throw an `IllegalStateException` if there are no songs in the play queue. If the currently playing song is the last one, then this method will simply return 0.
- `public void playNextSong()`: If the player is not currently playing anything, this must throw an `IllegalStateException`. Otherwise, this advances to the next song in the play queue and updates `totalRunTime` accordingly.
- `public void clearQueue()`: Clears the play queue of the player. **Does not reset the run time of the player.**
- `public void addSong(Song song)`: Adds a song to the end of the play queue. If there is nothing currently playing, then the song to add will end up being the currently playing song. In this case, also make sure to update the `totalRunTime` appropriately.

4 Examples

The following examples should help when validating your implementation's functionality:

Example 1

```
Song s = new Song("Highway to Hell", "AC /", "DC", 207);
System.out.println(s);
```

Console Output for Example 1

```
Highway to Hell, by AC / DC, runtime 207 seconds.
```

Example 2

```
Song s1 = new Song("Highway to Hell", "AC /", "DC", 207);
Song s2 = new Song("Stairway to Heaven", "Led", "Zepplin", 482);

Playlist p = new Playlist();

p.addSong(s1);
p.addSong(s2);

System.out.println(p);
System.out.println(p.totalRunTime());
```

Console Output for Example 2

```
[Highway to Hell, by AC / DC, runtime 207 seconds.
Stairway to Heaven, by Led Zepplin, runtime 482 seconds.]
689
```

Example 3

```
Song s1 = new Song("Highway to Hell", "AC /", "DC", 207);
Song s2 = new Song("Stairway to Heaven", "Led", "Zepplin", 482);
Playlist p = new Playlist();
```

```
p.addSong(s1);
p.addSong(s2);

Player y = new Player(p);
System.out.println(y.getCurrentSong());
System.out.println("Total runtime: " + y.getTotalRuntime());
System.out.println("Remaining runtime: " + y.getRemainingRunTime());

y.playNextSong();

System.out.println(y.getCurrentSong());
System.out.println("Total runtime: " + y.getTotalRuntime());
System.out.println("Remaining runtime: " + y.getRemainingRunTime());

y.playNextSong();
//The line above does not throw an exception!
//We just end up with a player that's not playing anything.
```

Console Output for Example 3

```
Highway to Hell, by AC / DC, runtime 207 seconds.
Total runtime: 207
Remaining runtime: 482
Stairway to Heaven, by Led Zepplin, runtime 482 seconds.
Total runtime: 689
Remaining runtime: 0
```

5 Submission instructions

Your submission should be a ZIP file called `sa1.zip` containing all the files from the stub (which must have been completed according to the instructions above) and a file `PlayerTest.java` with your test cases. No report is required. Your grade will be determined as follows:

- You will get a 0 if your code does not compile.
- The code must implement the following UML diagrams precisely.
- We will try to feed erroneous and inconsistent inputs to all methods. All arguments should be checked.
- Partial credit may be given for style, comments, and readability.

The UML diagram of `Song`, `Playlist`, and `Player` is given on the next page.

Song
private String name private String artistFirstName private String artistLastName private int runTime
public Song(String name, String artistFirstName, String artistLastName, int runTime) public String getName() public String getArtistFirstName() public String getArtistLastName() public int getRunTime() public String toString()

Playlist
private SLL<Song> songs
public Playlist() public Playlist(Song song) public int getSize() public Song getAt(int index) public void addSong() public int totalRunTime() public Song longestSong() public String toString()

Player
private Node<Song> currentSong private Node<Song> lastSong private int playQueueSize private int totalRunTime
public Player() public Player(Playlist plist) public int getTotalRuntime() public int getPlayQueueSize() public Song getCurrentSong() public ArrayList<Song> getAllSongs() public int getRemainingRunTime() public void playNextSong() public void clearQueue() public void addSong(Song song)