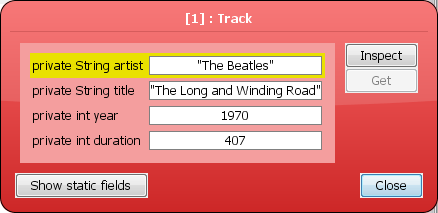
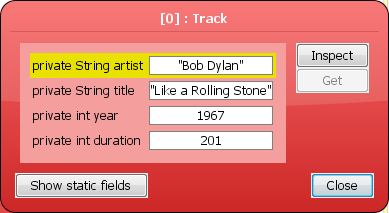
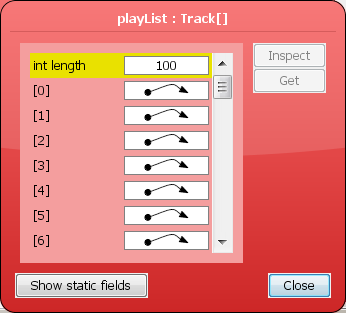
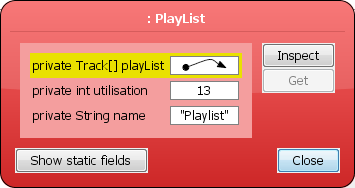
# Project Description

We want to develop an application for playing music on a mobile phone or a similar device. The application has three classes – a Playlist class, a Track class and a Driver class. The following diagram shows how the application will be structured.



Driver

Playlist

Track

Write the code for the three classes (i.e. Driver, Playlist and Track).

# Track Class

The Track class will be used to store the details of each individual track. The Track class should store the following information (i.e. instance values)

* A String for the track title
* A String for the name of the artist
* An int for the year the track was released
* An int for the track duration (in seconds)

The Track class should provide two constructor(s). One that creates an instance of the Track class using just the title and artist (the other two instance variables should be set to 0). A second that creates an instance of the Track class using all four data items (i.e. the title, artist, year and duration).

A pair of ‘set and a get methods for each instance variable in the class with the following headers

public void setTitle(String title) // This allows the title to be changed

public String getTitle() // This returns the current title

public void setArtist(String artist) // This allows the artist name to be changed

public String getArtist() // This returns the current artist name

public void setYear(int year) // This allows the year of release to be changed

public int getYear() // This returns the current year of release

public void setDuration(int seconds) // This allows the track duration to be changed

public int getDuration() // This returns the current duration

The class should also provide a **toString** method to provide a neat display style for Track instances. The track duration should be displayed in MM:SS (minutes:seconds) format.

Finally, the class should **implement Comparable** and include a **compareTo** method that determines the natural ordering of Track instances. Tracks should be ordered by title (case insensitive).

# Playlist Class

The Playlist class will be used to store a list of the tracks that the user wants to play. At any time the number of tracks in a playlist can be zero (the playlist is empty) or some number greater than zero. In other words, there is no limit on the number of tracks the playlist can contain. Each playlist will store the following information (i.e. instance values)

* A String for the name of the playlist
* A collection of Track objects. **You have to decide what type of collection structure to use to store the tracks.**

The Playlist class should support/provide the following operations/methods

* **Two** **constructor** methods that can be used to create an empty Playlist. The method headers are as follows

// Create a PlayList with a default name (e.g. My PlayList)

public PlayList()

// Create a PlayList with the specified name

public PlayList(String playListName)

* A **toString** method that can be used to return the contents of PlayList in a readable format.
* A pair of methods to allow the PlayList name to be inspected (get) or modified (set) with the following headers

public void setName(String name) // This allows the name to be changed

public String getName() // This returns the current name

* An **add** method to allow a new track to be added to the list. Tracks are ALWAYS added to the end of the list.

The **add** method header is overloaded as follows

// Add a Track where only the title and artist are known

// The year and duration should be set to zero

public void add(String title, String artist)

// Add a Track where ALL of the data is known

public void add(String title, String artist,

int year, int duration)

// Add a previously created instance of the Track class

public void add(Track t)

* A **remove** method to allow tracks to be deleted from the playlist. The **remove** header is as follows

public boolean remove(String title)

The method is passed the title of the track to be removed.

If there is a track in the playlist with the same title the track should be removed and the method returns true. You should use a case-insensitive comparison when matching track titles.

If there is no track in the playlist with the specified title the method should return false and the list should be left unchanged.

* A **showList** method that displays the playlist on the screen in sequential order (i.e. in the order that the tracks appear in the list). The **showList** header is as follows

public void showList()

If there are no tracks in the list the method should display the message “The list is empty”.

* A **playAll** method that plays all the tracks in the list either in sequence or randomly.

Passing the value false as a parameter plays the tracks sequentially (i.e. in the sequence they appear in the list).

Passing the value true as a parameter plays the tracks randomly. When playing tracks randomly each track should be played only once. In addition, playing the tracks randomly should NOT alter the sequence of the tracks in the list.

The method header is as follows

public void playAll(boolean random)

* A **playOnly** method that plays tracks in the list that satisfy specified criteria. The **playOnly** method is overloaded as follows

public void playOnly(String artist)

public void playOnly(int year)

The first method will play all the tracks in the list that **contain** (i.e. anywhere – at the start, in the middle, at the end) the specified text (case insensitive) in the artist name. For example, suppose some of the tracks in a playlist have the artist name “The Eagles.” Using playOnly(“eagles”) or playOnly(“EAGLES”) or playOnly(“THE eagles”) should play all the tracks in the list that have the specified strings.

The second method will play all tracks in the playlist that have the specified year as their year of release.

The tracks should be played in the sequence they appear in the list.

If none of the tracks match the specified value then no output should be produced.

We can’t actually play the tracks so for testing purposes displaying the track details on the screen will be considered equivalent to playing the track.

You may find it useful to develop additional “helper” methods that might simplify or improve the efficiency of some operations, or might reduce the amount of code you have to write. ALL helper methods should be private.

# Driver Class

The Driver class will be used to test the other classes and will contain just a main method that uses ALL of the methods provided in the other two classes to show that they have been tested and work correctly. A sample Driver class called **DriverForAssignment.java** will be provided shortly.

### Submission Requirements

You have the option of undertaking the project alone or as part of a team. If you plan to do this assignment with the same team as before then you DO NOT have to do anything. If you wish to form a new team please email [Dermot.Shinners-Kennedy@ul.ie](mailto:Dermot.Shinners-Kennedy@ul.ie) by **16h00 Thursday 2 April 2020 (GMT)** with the names and ID numbers of the team members. Teams are restricted to a maximum of four members. Management of the team is entirely a matter for the team members.

All team members are awarded the same score for the project.

An electronic copy of ALL of the .java files in your solution **MUST** be emailed to [Alan.T.Ryan@ul.ie](mailto:Alan.T.Ryan@ul.ie) .

**NB:** In **EACH AND EVERY** one of the .java files you submit you should include prominent comments in the code that contain the ID NUMBER(s) and NAME(s) of (all of) the author(s).

**NOTE: ONLY .java files and any text files required should be submitted. No exe’s, tar’s or any other file type should be uploaded. It is YOUR responsibility to ensure that the files submitted are the correct versions. It should NOT be necessary to install additional software to test your submission.**

Submissions must be received on or before **16h00 Thursday 30 April 2020 (GMT).**

**This project accounts for 25% of the overall marks available for the module.**