Overview

This task allows you to build an application that reads in an album with multiple tracks from a file then displays the album and track information to the Terminal.

Purpose:	Learn how to declare and work with arrays of records and enumerations.
Task:	Demonstrate the use of: Arrays; Enumerations; Records and Loops in the context of the requirements described in the instructions.
Time:	This task should be completed before the start of Week 9.

Resources to read before doing the task

Books:	Flanagan & Matsumoto (2008) The Ruby Programming Language, O'Reilly. Chapter 11 of Pine (2014) Learn to Program (2nd) Pragmatic Programmer.

Submission details

You must submit the following files to Doubtfire:

- Album File Handling **source code** (album_file_handling.rb)
- **Screenshot** of the program running in Terminal

Make sure that your task has the following in your submission:

- Code must follow Ruby coding convention used in the unit (layout and use of case).
- You are storing and working with multiple values in an array.
- You are using records/classes and an enumeration to store the values.
- The code must run and meet both requirements above your tutor

Instructions

In this task you will need to develop a program that reads in an album with multiple tracks from a file, then displays the album and track information to the Terminal. You will build on the skills developed in your other Pass and Tutorial tasks to develop the program.

Steps to complete the task:

1. Download and extract the resources which consist of the Ruby source code file **album_file_handling.rb** and the data file **album.txt**.

2. Requirements:

- The program must read in a single **`album`** and a number of **`tracks`** for the album as well as a **`genre`** for the album from a file.
- You can have as many genres as you like, but these must be defined using an **enumeration**.
- For each track you must read in a **track name** and a **track location** (i.e., filename and path in disk drive).
- Your application must read the album and track information from the provided **album.txt** data file.
- You can re-use the code from previous tasks (5.1 and 5.2) as well.

3. Some hints that may be useful for your code development:

First, have a look in the **album.txt** file, and identify the meaning of each line, and how it is linked to the **`Album` record** that you want to design.

```
Greatest Hits
Neil Diamond
1
3
Crackling Rose
sounds/01-Cracklin-rose.wav
Soolaimon
sounds/06-Soolaimon.wav
Sweet Caroline
sounds/20-Sweet_Caroline.wav
```

Secondly, in Ruby, you can store the values of **array** into the attribute of **record**. For example, let us consider a **record** of `Album` defined with two attributes `title` and `tracks` below

```ruby
class Album
    attr_accessor :title, :tracks
    def initialize (title, tracks)
        @title = name
        @tracks = tracks
    end
end
```

You can create a new **array** named `tracks` having two elements, assign some values to it, and then store it as an attribute of an `Album` **record** as follows:

```ruby
tracks = Array.new()
tracks[0].name = 'Track 1 song'
tracks[0].location = 'track1.mp3'
tracks[1]name = 'Track 2 song'
tracks[1].location = 'track2.mp3'
album = Album.new('The title of Album', tracks)
```

Thirdly, about the use of enumeration, `Genre` has four elements, each corresponds to a number in the list `1,2,3,4`. This numeric enumeration is linked to the global variable `$genre_name`, which is an array of names (in strings).

```ruby
module Genre
  POP, CLASSIC, JAZZ, ROCK = *1..4
end
$genre_names = ['Null', 'Pop', 'Classic', 'Jazz', 'Rock']
```

To access an element of this enumeration, use the syntax `Genre::POP` or so. The Terminal will show `'Pop'` when you print the element of array using the corresponding enumeration's element `Genre::POP`

```
puts $genre_names[Genre::POP]
```

You may simply regard `Genre::POP`, `Genre::CLASSIC`, `Genre::JAZZ`, `Genre::ROCK` as *nicks* (so that programmer can recall easily), instead of having to remember the coded values `1,2,3,4` that links to the actual names. Same way as in Task 3.3, instead of remember the HEX code `0xff_ff0000`, we can simply refer to the *nick* `Gosu::Color::RED` for the red colour.

4. There is a minimum requirement for naming and design before your code can be accepted regardless of how well the code is functioning. You will be given feedback on the quality of your code as well as your artefact naming.

5. Once done, proceed with the submission (same as previous task). Make sure you start this task early so that you can have time to fix it when your tutor required to get it marked off before the deadline.

You now have another portfolio piece. This will help demonstrate your learning from the unit.