COMP41100

EXPLORING PROGRAMMING IN RUBY

PRACTICAL FOUR

17th October 2013

Andrew Doyle
Student Number: 12252388

University College Dublin

PART ONE

Task:

The main task this week is to get the iTunes program going, by entering your own data files.

Solution:

The author creating a CSV by exporting a playlist using ITunes. To do this, the first step was to ensure that the **Menu Bar** was shown by expanding the dropdown menu in the top-left hand corner of the program and clicking **Show Menu Bar**. The next step was to click **File, Library, Export Playlist**. This gives the user the option to save as a **.txt** file. Saving this text file as a **.csv** allowed the author to edit the required information in Microsoft Excel.

Information was deleted that was not required until only **Name**, **Artist**, **Album**, **Time**, and **Genre** information were left. Note that **Genre** is the field that was added to the **Song** class as part of the requirements for Part Two. Initially the programme was tested without the **Genre** field; however in order to add this field many of the steps outlined in this section were repeated.

The **ID** field was added to the **CSV** file in Excel, which was completed quickly with Excel`s auto increment feature. Using the data sort feature in excel, the information was sorted alphabetically by the **Album** field.

The main issue that remained at this stage was that the file was not in the same format as the original sample **CSV** file. Instead of information being separated by commas, information was separated by tabbed spaces. Using the find and replace feature in **Rubymine**, the author replaced the spaces with commas. The author also ensured the removal of trailing whitespaces and null fields – both of which would otherwise cause the **iTunes** program to crash.

The above steps were completed with much trial and error, until the program ran without errors. The original **to_s** method is output which on this occasion shows new information that has been read in with the aforementioned file.

PART TWO

Task:

You also need to add a new attribute to the Song class and make sure the program still works with this change throughout. If it requires changes to any methods then those need to be done too.

Solution:

The **Genre** field was added to the CSV file using Excel as discussed in Part 1. The song class was then updated to include this field:

```
    attr_accessor :name, :album, :artist, :time, :genre, :id, :in_libs
    def initialize(name, album, artist, time, genre, id, in_libs)
        @name = name
        @album = album
        @time = time
        @artist = artist
    @genre = genre
        @id = id
        @in_libs = in_libs
    end
```

Changes were required to the **for_each** in the **read_in_songs** method from **reader.rb**. The **genre** field and a new **row[5]** indicator to read from the **CSV** file were added:

```
  CSV.foreach(csv_file_name, :headers => true) do |row|
         songname, artist, album, time,genre, id = row[0],row[1], row[2], row[3],
row[4], row[5]
         unless (songname =~ /#/)
             songs << Song.new(songname,album,artist,time.to_f,genre,id,nil)
         end
       end
```

After completing the above steps, the genre information can be read into the program.

PART THREE

Task:

Finally, define a new method for one of the classes in the iTunes program and make it all work harmoniously.

Solution:

A method was defined in **itunes.rb** called **read_information**. When called, it reads in the **songs** array and allows the user to view a list of unique Artists, Albums or Genres. It also allows the user to see all song information in a tabular format on the console, with the information also written to the **music_library.txt** file. This choice is made possible by carrying out the user`s choice in a **case statement**.

Figure 3.1 – User choices

```
What do you want to view?

Artists               | 0
Albums                | 1
Genres                | 2
All song Information  | 3
Exit                  | 4
```

Case 0 – 2: Creating a unique list of information

Information regarding artists, albums and genres in the library is duplicated many times. It would be useful if the user could view a unique list of the aforementioned items.

Taking **Artists** as an example, a separate array was created called **artists**, which reads in all the **artist** fields from the **songs** array using an **each** loop in conjunction with an **array push**. Using the destructive method **uniq!** ensures that there are no duplicates:

```
#CREATING LIST OF UNIQUE ARTISTS
artists = []
songs.each { |member|
         artists << member.artist
}

artists.uniq!
```

The **artists** array is then **puts** to the console screen in the appropriate **when** of the **case statement**. The procedure for the unique **Albums** and **Artists** list is the same.

Case 3: Displaying all song information

To ensure appropriate column widths the format of the output to the screen is first of all specified together with headings:

```
format = '%-80s %-50s %-100s %-6s %-20s'
puts format % ['SONG NAME', 'ARTIST', 'ALBUM', 'LENGTH','GENRE']
puts format % ['---------', '------', '-----', '------','------']
```

An **each** loop is used to output the information:

```
songs.each {|member|
     puts format % [member.name, member.artist, member.album,member.time,member.genre]
     }
```

The information is then written to the **music_library.txt** file in the program directory:

```
#writing the information to file

File.open("music_library.txt", "w") do |file|
  file.puts format % ['SONG NAME', 'ARTIST', 'ALBUM', 'LENGTH','GENRE']
  file.puts format % ['----------', '------', '-----', '------','------']
    songs.each { |member|
      file.puts format % [member.name, member.artist,
member.album,member.time,member.genre]
    }
end
```

The user is notified that the information has been written to the file:

```
puts "\n" + 'The music library has been written to the file music_library.txt'
```

Case 4: Exit

The user can exit the program at any stage, made possible in this menu option by **exit +2**:

```
# LET THE USER EXIT THE PROGRAM
  when choice == 4 then
  puts "\nThanks..Have a great day!"
  exit + 2
```

Suggested Improvements:

Allow the user to view the list of songs sorted alphabetically by Artist or Genre. The **songs** array would have to be sorted or a new array created with the fields from the **songs** array sorted appropriately.

LIST OF FIGURES