Coursework Documentation

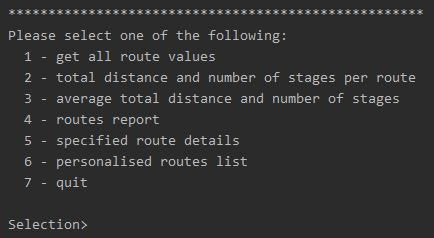
Testing and evaluation

Albert MAS Compés

2020

# **Testing**

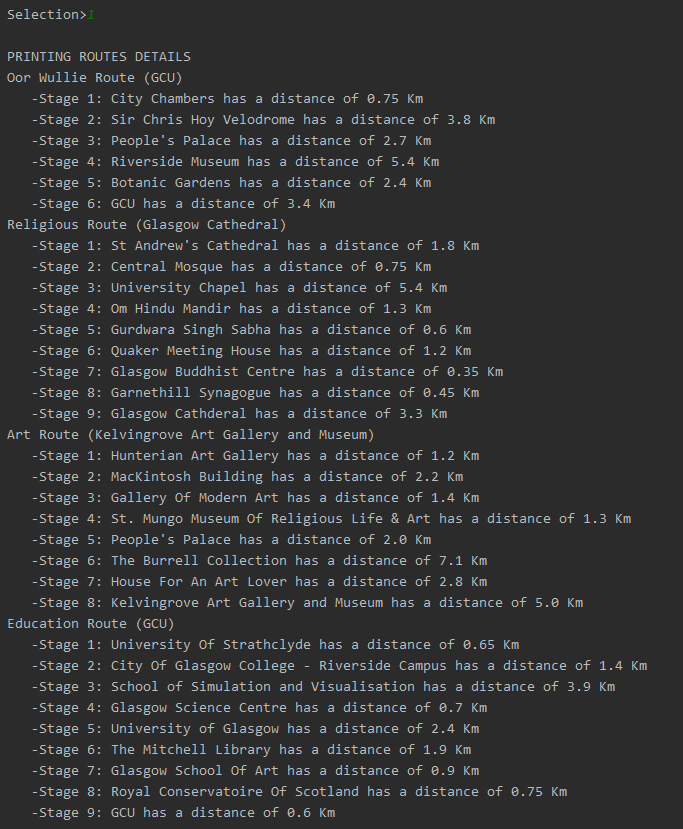
## Menu



When the application is initialised, a menu will appear displaying all the available operations.

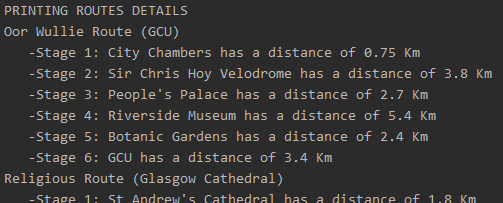
Under the menu, the users will find a text saying “Selection>” where they will be able to write the option they want to activate. After performing the desired option -or introducing a wrong command- the menu will appear once again to allow the user select another option.

## Operation 1

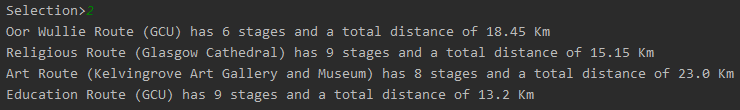


Operation 1 will print each route’s details. The format I chose is to print first the name of the route and then print each stage in order. The format used to show the values of each stage clearly shows the important data. It is also very distinguishable from the routes names so that the user can quickly scan the data and not get lost in the text, even if a lot more routes are added.

Here we can see the first fragment in more detail:



## Operation 2



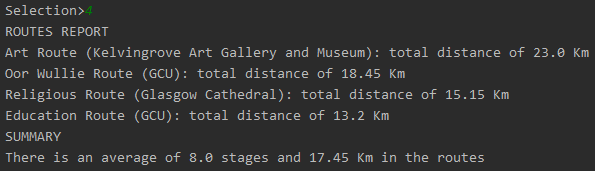
Operation 2 prints a summary of all routes for a more compact view. It shows each route’s name, number of stages and total distance.

## Operation 3



Operation 3 prints an average of the stages and distances in all routes combined.

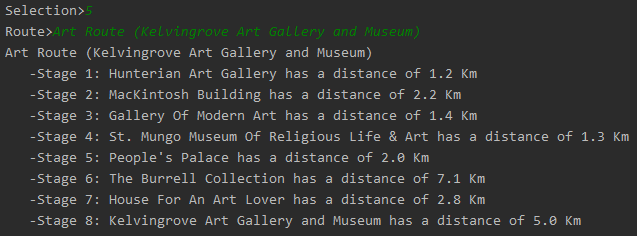
## Operation 4



Operation 4 prints all the routs in descending order of the total distance. It shows the name of the route and the total distance. I chose to not show more specific details such as number of stages and distance per stage, as you can get this information on Operation 1.

Then, it shows a summary with the average stages and distance of all routes combines (like Operation3).

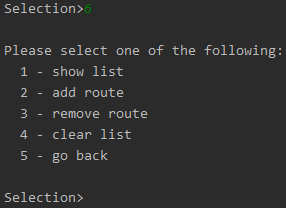
## Operation 5



Operation 5 lets the user introduce the name of a route and -provided that it’s in the data base- the details of that route are printed in the same format used in Operation 1. If the user introduces a wrong route, a message will be printed saying that the route was not found.

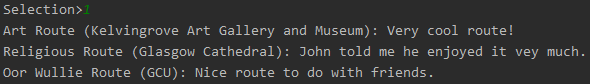


## Operation 6



Operation 6 allows the user to create a personalised list of routes and add a comment to each one. It is the more extensive, and for that reason, selecting it will print another menu with the available options. These options are:

1. Show the list

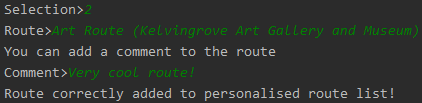


If there are routes in the list, they will be printed with the saved comment.

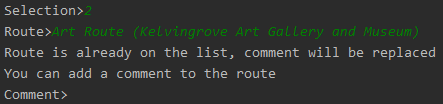


If there are no routes in the list yet, a message will say so.

1. Add a new route to the list



After selecting this option, you will have to write the name of the route to add. Then you will be able to write a comment for that route.



If the route is already in the list, a message will warn you that instead of adding it, you will just be changing the comment.



If the route doesn’t exist you will be warned too.

1. Remove an existing route from the list



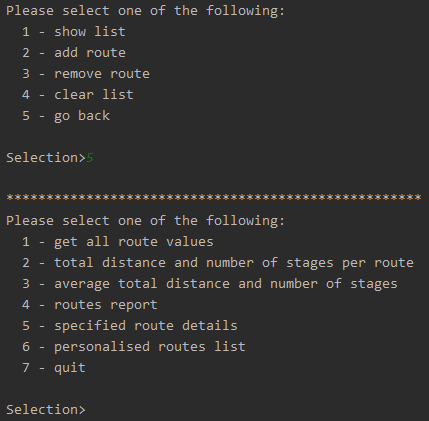
After selecting this option, you will have to write the name of the route to remove.

1. Clear the whole list



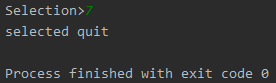
A message will tell you that the list has been correctly cleared after selecting this option.

1. Go back to the main menu



Choosing Go back, the application will return to the main menu. You can always choose option 6 again to see the list with the routes you have selected and their respective comments (as long as you don’t close the application).

## Quit Application



Selecting quit will correctly terminate the application.

# **Evaluation**

When I started learning Scala at the beginning of this module, barely knowing any Java, I was a bit concerned with learning a new programming language in such a short time, but throughout the lab exercises I realised that Scala was pretty intuitive. Some functions like *fold* or *map* did make me scratch my head for a while, but overall I didn’t find learning Scala very hard at all.

During the development of this assignment, I really enjoyed a lot of the convenient aspects of Scala like the foreach function, that really saved me quite a lot of time comparing it to a *for* loop. I also liked how clean the code looked when I made use of functions like *match*, which clearly differentiated each case without using lots of *if* and *else*.

A thing I didn’t quite understand was the suitability of using immutable vars -like *List*- that had to be overwritten when adding or removing an element against a mutable *val* -like *ListBuffer*- that could be modified (and vice versa). I did know the difference but I wasn’t sure which one was better for what I wanted to do. I did appreciate being able to convert from a mutable list to an immutable one by just typing *.toList*, though.

The approach I took was -following the steps used in the menu created in lab 5- to have a menu looping that invoked a different function for each operation that the user could select. This function asked the user for any input required, like the name of a route, and then called another function to perform the required operations. The function then printed the returned data in a correct format.

Although I have always programmed in an imperative style, in this assignment I have tried to use a functional style as much as possible. For that reason, I have used -wherever possible- *foreach*, *pattern matching*, *partial function application* and more.

I am used to working in C++, so if I had to do this application my own way, my approach would probably be to divide this exercise in different classes that contained each part of the exercise. That way I find it easier to structure the code and get smaller, more readable classes that focus in specific subjects instead of putting everything in a single class with lots of functions.

Overall, I do think that Scala is a very good option for developing an application of this sort. Programming in Scala, you can enjoy a lot of useful functions and tools that speed up the development process, that you wouldn’t find in lower level languages like C++. As I said, I’m not very used to a functional programming style, but I think that it has a lot of potential.