Exercise 7 - Collections

Objective

The objective for this exercise is to use various collections, iterate over them and be able to use them in different situations. We also look at how to read in files and combining scala and java code together.

We also look at our own implementation of a List class which helps cover some of the concepts previously seen in functional programming and recursion.

References

Chapter 7 of the slides – number 142 to 173

Overview

1. Create a List of Ints using the factory method for List
2. Using the ArrayBuffer class create a list of Ints from 1 to 10
3. Create a list of Strings using the Cons operator (::) remember – the cons operator is right associative as it ends with a colon. All lists end with the terminating symbol, for scala this is Nil
4. Concatenate your List created in part 1 with your List created using the Array Buffer in part 2. Note how scala is able to deal with the collections being different
5. Write a function that takes in a list and returns the nth item in that list. The list class has two useful methods, head, which returns the first item in the list, and tail that returns the rest of the list. You should use recursion in your solution. (you can use the List you created in part 1)
   1. Use generics to make sure that the function can handle any type of list
6. Create another function that returns the position of the first occurrence of an item in a list. For example: If the list contains (a,b,c,d,e,f,g,h,u,a,b,c,q) then find the position of the letter ‘u’ in the list.
   1. Use a calling rule to make your solution neater if necessary
7. List doesn’t have a remove method. Implement a remove method for a list. Again, the best way to do this is to use a recursive function.
8. Write a function called values that takes in a function, a minimum number and a maximum number
   1. The function should take an Int and return an Int
   2. For every number between the minimum and maximum, it should apply the function
   3. The output of the function should use yield to produce a set of tuples that look like this:

((x, f(x)), (x+1, f(x + 1), (input, output), etc… )

so if the function was called with:

values(x => x \* x, -2, 2)

the output would be

((-2,-4),(-1,1),(0,0),(1,1),(2,4))

So the output is a list of pairs of numbers with the first one as the input value, and the second one the value after the function has been applied.

Hints: Look at the for/yield loop first mentioned in the syntax basics slides (number 38). Try to make the function output the range of values first, then output the answers before combining the two.

1. Create a class called Person that has an ID, name, age and gender fields. The ID and age should be Ints, the name is a string and the gender should be an Option[String] so that field can be considered optional. Ensure that you can get the values of the fields (The option type is described on slide 168)
   1. Override the toString method in the person object and provide a sensible output
2. Create three Person objects from your class, one male, one female and one who didn’t answer.

val p1 = new Person(1, "fred", 45, Some("Male"))

etc.

1. Now define an object called Employees which is a map of each persons ID to the object
2. Iterate over the map, printing out all the values you have stored to check if your map has been correctly defined.
3. Create a function which returns an Option[Person] based on their ID. For example you could call the function using:

findById(1)

which would return a person in the content of the Option. If the ID isn’t found in the map the function should return ‘None’

Test your Employees object by calling findById with valid and invalid IDs. You can check what was returned by calling isDefined on the value returned by the function.

(Note: checking whether an option has a value isn't really best practice, and we'll see a better way to use them when we cover pattern matching)

1. As Scala code compiles and runs on the JVM we can easily call java from within scala programs

Write a function that reads words from a file and produce a map of word -> frequency

You can use a java.util.Scanner to get the input.

val in = new java.util.Scanner(new java.io.File("myfile.txt"))

while (in.hasNext)

process in.next

Your task is to write the process method, passing it "in.next". Do not import process from any other package as this will make your code fail.

The processing can either be completed inside the while loop, or in a separate method called process which takes in a string and adds the word to the map

If you have time, try repeating the exercise with an immutable map