Practical Sheet nº6

Content

- Purely functional state in Functional Programming
- Input / Output (IO)
- Putting It All Together

In this class, we will write our first complete functional programs that have purely functional state and deal with IO.

Open the project provided (with three packages) as a starting point for the exercises of this sheet.

6 - Purely functional State and IO

6.1 - Coin Flip game

Remember the Coin Flip game (provided with this sheet and presented in the TP classes) and solve the following exercises:

a) Modify the game so you can play a new game by pressing 'n' or 'N'

b) After adding the ability to play a new game, modify the program to keep a history of all previously played games (i.e., saving the results) of the "session". When the game is over, the history should be printed in the screen (see below).

```
=== GAME OVER ===
#Flips: 1, #Correct: 1
History:
#Flips: 3, #Correct: 2
#Flips: 5, #Correct: 3
```

c) Use a `nextInt` method referentially transparent, i.e., pure. For this, add the following case class and Trait as starting point:

```
case class MyRandom(seed: Long) extends RandomWithState {


 def nextInt: (Int, RandomWithState) = {
       val newSeed = (seed * 0x5DEECE66DL + 0xBL) & 0xFFFFFFFFFFFFL
       val nextRandom = MyRandom(newSeed)
       val n = (newSeed >>> 16).toInt
       (n, nextRandom)
 }
```

```
 def nextInt(n: Int): (Int, RandomWithState) = {
       val newSeed = (seed * 0x5DEECE66DL + 0xBL) & 0xFFFFFFFFFFFFL
       val nextRandom = MyRandom(newSeed)
       val nn = ((newSeed >>> 16).toInt) % n
       (if(nn<0) -nn else nn, nextRandom)
 }
}
```

```
trait RandomWithState {
       def nextInt: (Int, RandomWithState)
       def nextInt(n: Int): (Int, RandomWithState)
}
```

6.2 – Implementing a Table with a List

Consider the `Turma` project from Practical Sheet nº5 with a new `main` and the `IO_Utils` object (provided with this sheet) to deal with IO and change of state in a purely functional way (i.e., without using `var`). Illustrate the process of state change by using the `changeNP` method (the process to be applied to all other methods that change the state of the class i.e., `changeNT` and `insert` are analogous). Therefore, you should complete the new `main` method of the `Main` object (Main.scala) that follows:

```
def main(args: Array[String]): Unit = {

  val state: Turma = Turma("id22", List((11,"as",RegimeOPT.Ordinario,
Some(1), Some(20)) ,(12, "Jose", RegimeOPT.TrabEstud, Some(13), None)))

  mainLoop(state)

  @tailrec
  def mainLoop(state: Turma) {

    IO_Utils.showPrompt()
    val userInput = IO_Utils.getUserInput()

    // handle the result
    userInput match {
      case "C" => {
          IO_Utils.showOptionsNum()
          val userInputNum = IO_Utils.getUserInput()

        //???
```

```
        // val newState = state.changeNP( ??? )
        //???

          mainLoop(newState)
      }
      case _   => {
        IO_Utils.printTurmaState(state)
      }
    }
  }
}
```

Immutable Maps (collection) - How to add, update, and remove elements.

Create an immutable map:

```
scala> val a = Map("ID1" -> "Jose")

val a: scala.collection.immutable.Map[String,String] = Map(ID1 -> Jose)
```

Add one element:

```
scala> val b = a + ("ID2" -> "Pedro")

val b: scala.collection.immutable.Map[String,String] = Map(ID1 -> Jose, ID2 -
> Pedro)
```

Update a key/value pair:

```
scala> val c = b + ("ID2" -> "Rodrigo")

val c: scala.collection.immutable.Map[String,String] = Map(ID1 -> Jose, ID2 -
> Rodrigo)
```

Remove one element:

```
scala> val d = c - "ID1"

val d: scala.collection.immutable.Map[String,String] = Map(ID2 -> Rodrigo)
```

6.3 – Putting it all together (using Maps)

Consider the *PuttingItAllTogether* project (provided with this Sheet) representing a very simplistic menu-driven (text-based) Database system (using a Map) that will have the following capabilities:

a) Add an Entry;
b) Remove an Entry;
c) Update an Entry;
d) Show the content of the map;

The capability *Add* is already implemented therefore, you should only complete the remaining ones.