Practical Sheet nº5

Content

- Algebraic Data Types (Tree)
- Functional Object-Oriented Programming
- Option type and Try

5.1 – Pattern Matching over elements of an Algebraic Data Type (Tree)

Considering the following code:

```
//Example.scala  (Case Class and Companion Object)


sealed trait MyTree[+A]

case object Empty extends MyTree[Nothing]

case class Node[A](value: A, left: MyTree[A], right: MyTree[A]) extends MyTree[A]


case class Example[A](myField: MyTree[A]) {

  def maximum() = Example.maximum(this.myField.asInstanceOf[MyTree[Int]])

  def depth() = Example.depth(this.myField)

  def map[B](f: A => B):MyTree[B] = Example.map(this.myField)(f)

}


object Example{

  def maximum(t: MyTree[Int]): Option[Int] = { ??? }

  def depth[A](t: MyTree[A]): Int = { ??? }

  def map[A,B](t: MyTree[A])(f: A => B): MyTree[B] = { ??? }


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

    val tree1 = Node(42, Node(8, Empty, Empty), Empty)

    val t = Example(tree1)

    println(s"Maximum element of the tree: ${ t.maximum()}")

    println(s"Depth of the tree: ${ t.depth()}")

    println(s"Map: ${ t.map((x:Int)=>x*2)}")

  }

}
```

a) Complete the function `maximum` that returns the maximum element in a `MyTree[Int]`.

b) Complete the function `depth` that returns the maximum path length from the root of a tree to any leaf.

c) Complete the function `map`, analogous to the method of the same name on `List`, that modifies each element in a tree with a given function.

5.2 - Implementing a Table by a List

Consider the representation of information related to the evaluation of students enrolled in a discipline, that has in its evaluation system a theoretical component and a practical component, by the following data types. Note that each student may **or may not** have already obtained a grade in one of the components of the discipline (theoretical or practical), which is represented using the `Option` type.

file: `RegimeOPT.scala`

```
package Ficha5
object RegimeOPT extends Enumeration {
       type RegimeOPT = Value
       val Ordinario, TrabEstud = Value
}
```

file: `Turma.scala`

```
package Ficha5
import Ficha5.RegimeOPT.RegimeOPT
import Ficha5.Turma._

case class Turma(id: String, alunos: Alunos){
  ???
}
object Turma {
  type Nome = String
  type Numero = Int
  type NT = Option[Float]
  type NP = Option[Float]
  type Regime = RegimeOPT
  type Aluno = (Numero, Nome, Regime, NT, NP)
  type Alunos = List[Aluno]

  def trabs(t:Turma):Alunos = { ??? }
}
```

file: Main.scala

```
package Ficha5

object Main {

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

      val  t:  Turma  =  Turma("id22",  List((11,"as",RegimeOPT.Ordinario,  Some(8),
Some(20)) ,(12, "Jose", RegimeOPT.TrabEstud, Some(13), None)))

    println("Workers-students: " + t.trabs())

  }

}
```

Define the following methods:

a) `trabs` that filters workers-students only.
b) `searchStudent` that performs the search for a student by their student number. Note, the type returned is `Option[Aluno]`
c) `finalGrade` that calculates a student's final grade according to the formula NF = 0.6NT + 0.4NP. Note that it is only possible to calculate this grade if the student number provided exists in the class, and both NT and NP components reach the minimum grade of 9.5 values otherwise None should be returned. Use the Option `get` method to access its value.
d) `approved: List[(Nome, Float)]` that presents all students' grades approved, i.e. with a final grade greater than or equal to 10 values.
e) `changeNP` and `changeNT`, which allow to change the practical and theoretical grades of a student in a class. Suggestion: use the List `updated`, `indexWhere` and `apply` methods instead of the `searchStudent function`.

A search in this table becomes more efficient if the information is kept in order. Being the search key the student number, the information should be maintained according to this element.

f) Write an ordering `insert` method to add new students in a class.
g) Redefine the previous `search` method, considering that the table is now represented by an ordered list.

5.3 Implementation of a Table by a Binary Search Tree

Remember that these trees are characterized by the following property (order invariant): the content of any node located to the left of a node X is necessarily smaller than the content of X, which in turn is necessarily smaller than the content of any not located to its right. Admitting the occurrence of equal elements, one of these restrictions is relaxed for "less or equal".

Considering a `MyTree[Aluno]` trait (see exercise 5.1), type `Alunos` is now a `MyTree[Aluno]` (i.e. `type Alunos = MyTree[Aluno]`) instead of a `List[Aluno]`. Write the following methods (their meaning is the same as the one considered in the previous section):

a) `trabs (returning List[Aluno])`
b) `searchStudent`
c) `finalGrade`
d) `approved`
e) `changeNP` and `changeNT`
f) `insert`