Exercise 15 HIGHER KINDED TYPES

Overview

Monoids and Functors are certain kinds of containers which can be combined in interesting ways.

Functors are mappable types: types which can run a one-argument function over their contents.

Monoids are appendable types: types which can join their values together.

In this exercise you will use these features to stream-line handling of user information.

Objective

Review the powerpoint, demonstrations and other notes for this chapter, then:

Using Monoids and Functors, print out a user's profile information: their name, age and other details.

Questions

// six variables have been predefined here  
// but suppose there come from a profile information form:  
  
**val** myName: Option[String] = **"Sherlock"**.some  
**val** myAge: Option[Int] = 28.some  
**val** myTitle: Option[String] = None  
  
**val** myHobbies: List[String] = *List*(**"forensic science"**, **"violin playing"**)  
**val** myJobs: List[String] = *List*(**"detecting"**, **"collecting"**)  
**val** myFavPeople: List[String] = *Nil*

Part 1: Monoid

//Q. define the variables  
// mname: Option[String]  
// mage: Option[Int]  
// mdetails: List[String]  
//  
// using |+|  
  
// mname is the users title followed by name if both have Some(),  
// otherwise handle Nones appropriately  
  
// mage is the users age +1 if it has Some(),  
// otherwise handle None appropriately  
  
// mdetails is the list containing all hobbies,  
// jobs and favourite people  
  
//Q. using a for-comprehension,  
// print s"Next year $n is $a"  
// where $n is the name, and $a the age  
  
//Q. using a for-comprehension,  
// print all the details  
  
  
//below is one way of implementing the Monid type class for Boolean  
  
//Q. revise this definition   
// so that a person is allowed in only when both isOpen and isAdult  
  
**implicit object** BoolMonid **extends** Monoid[Boolean] {  
 **val** *zero* = **false  
 override def** append(left: Boolean, right: => Boolean): Boolean = left || right   
}   
  
**val** isAdult = age.getOrElse(0) >= 18  
**val** isOpen = **false  
  
if**(isAdult |+| isOpen) {  
 *println*(**"Go in!"**)  
} **else** {  
 *println*(**"Stay out!"**)  
}

Part 2: Functor

*//Q. redefine fname, fage and fdetails as above  
// however:  
  
//Q. make the whole name uppercase   
//Q. define age without using |+|   
//Q. make all the details lowercase  
  
//HINT: map  
  
  
//Q. using for-comprehensions as above,  
// print out the user's information*

Extra

*//Q. define a Functor instance for Ticket  
  
//Q. using toUsd,  
// convert the tickets to US dollars and print out each price***trait** Ticket[A] {  
 **val** price: A  
}  
  
**case class** SingleTicket[A](**val** price: A) **extends** Ticket[A]   
**case class** GroupTicket[A](**val** price: A, **val** groupSize: Int) **extends** Ticket[A]  
  
**val** toUsd = (p: Int) => **s"$**{p \* 1.3} **USD"  
val** tickets = *List*(*SingleTicket*(10), *SingleTicket*(11), *GroupTicket*(40, 4))

Solution

**val** myName: Option[String] = **"Sherlock"**.some  
**val** myAge: Option[Int] = 28.some  
**val** myTitle: Option[String] = None  
  
**val** myHobbies: List[String] = *List*(**"forensic science"**, **"violin playing"**)  
**val** myJobs: List[String] = *List*(**"detecting"**, **"collecting"**)  
**val** myFavPeople: List[String] = *Nil*

Part 1  
**val** mname = myTitle |+| myName  
**val** mage = myAge |+| 1.some   
**val** mdetails = myHobbies |+| myJobs |+| myFavPeople  
  
**for**(n <- mname; a <- mage) println(**s"Next year $**n **is $**a**"**)  
**for**(d <- mdetails) println(d)  
  
  
**implicit object** BoolMonid **extends** Monoid[Boolean] {  
 **val** zero = **false  
 override def** append(left: Boolean, right: => Boolean): Boolean = left || right   
}   
  
**val** isAdult = age.getOrElse(0) >= 18  
**val** isOpen = **false   
  
if**(isAdult |+| isOpen) {  
 println(**"Go in!"**)  
} **else** {  
 println(**"Stay out!"**)  
}

Part 2

**val** fname = (myTitle |+| myName) map { \_.toUpperCase }  
**val** fage = myAge map { \_ + 1 }  
  
**val** fdetails = (myHobbies |+| myJobs |+| myFavPeople) map { \_.toLowerCase }  
  
**for**(n <- fname; a <- fage) println(**s"Next year $**n **is $**a**"**)  
**for**(d <- fdetails) println(d)

Extra

**trait** Ticket[A] {  
 **val** price: A  
}  
  
**case class** SingleTicket[A](**val** price: A) **extends** Ticket[A]   
**case class** GroupTicket[A](**val** price: A, **val** groupSize: Int) **extends** Ticket[A]  
  
**val** a = SingleTicket(10)  
**val** b = GroupTicket(20, 2)  
**val** c = SingleTicket(12)  
  
  
**val** toUsd = (p: Int) => **s"$**{p \* 1.3} **USD"  
val** tickets = List(SingleTicket(10), SingleTicket(11), GroupTicket(40, 4))  
  
**implicit object** TicketFunctor **extends** Functor[Ticket] {  
 **def** map[A, B](fa: Ticket[A])(f: A => B): Ticket[B] = fa **match** {  
 **case** SingleTicket(price) => SingleTicket(f(price))  
 **case** GroupTicket(price, size) => GroupTicket(f(price), size)  
 }  
}  
  
  
**for**(t <- tickets) println(t)