

CURSO DE PROGRAMACION SCALA

Sesión 5

Sergio Couto Catoira

Índice

- › Introducción a implícitos
- › Estructuras de datos funcionales
- › Varianza
- › Funciones variádicas
- › Mejoras sobre la inferencia de tipos

Implícitos

- › Permiten definir parámetros sin necesidad de pasárselos explícitamente a la hora de la llamada
- › En la API pueden verse muchos ejemplos:
 - `def sortBy[B](f: (A) ⇒ B)(implicit ord: math.Ordering[B]): List[A]`
- › Puede pasarse explícitamente si fuese necesario
- › Si hay varias opciones en el ámbito, salta un error:
 - Ambiguous implicit values

Estructuras de datos funcionales

- › Trait que representa el tipo
- › Case object que representa el caso de que esté vacío
- › Case class que representa el caso de que tenga elementos

Varianza

- › Anotaciones de varianza (variance annotation)
 - +A indicaría que covariante(positivo)
 - A indicaría que es invariante
 - -A indicaría que es contravariante(negativo)
- › Covariante: Una estructura de subtipos es considerada subtipo de la estructura de supertipos
 - Si A es subtipo de B \Rightarrow List[A] es subtipo de List[B]
- › Contravariante: Una estructura de subtipos es considerada supertipo de estructura de supertipos
 - Si A es subtipo de B \Rightarrow List[B] es subtipo de List[A]

Ejercicio

- › Crea las funciones `sum` y `product` en el companion object `Lista` que aparece en el código. Recuerda usar `pattern matching`

Pattern matching

- › Sabrías lo que devolvería la siguiente sentencia de pattern matching sobre el TDA lista

```
Lista(1,2,3,4,5) match {  
  case Cons(x, Cons(2, Cons(4,_))) => x  
  case Vacio => 42  
  case Cons(x, Cons(y, Cons(3, Cons(4, _)))) => x+y  
  case Cons(h, t ) => h + sum(t)  
  case _ => 101  
}
```


Funciones variádicas

- › Permiten recibir entre 0 y n parámetros separados por coma
 - `def apply(as: A *): Lista[A]`
- › Es azúcar sintáctico (syntactic sugar) para pasar secuencias de elementos.
- › Convertir una lista en secuencia: `_*` (Sólo para pasarla como argumento a una función variádica)



Ejercicio

- › Crea una función tail que devuelva la cola de la lista.
- › Crea una función setHead, que reciba una lista y una cabecera y reemplace la cabecera de la lista. Si la lista recibida es vacía, debe devolver una lista con un único elemento.

Ejercicio

- › Generaliza la función tail en la función drop, que elimina n elementos de la lista. Recuerda hacerla tail safe.
- › Genera una función dropWhile que elimina los elementos de la lista mientras cumplan la función recibida. Debe recibir una lista y una función $A \Rightarrow \text{Boolean}$

Data sharing en estructuras de datos funcionales

- › Data sharing vs copia pesimista
 - Data sharing => Comparto objetos inmutables
 - Copia pesimista => Cada método hace una copia, por si algún otro método modifica algo
- › Todos los métodos que se generaron en esta sesión, devuelven otra lista.
- › Sin embargo, en muchos casos devolvemos el mismo tail. No se hace copy, ni clone. Es el mismo objeto.
- › Es seguro porque es inmutable.
- › Nos permite implementar funciones de forma eficiente tanto en tiempo como en memoria. No se hacen copias innecesarias de datos.

Ejercicio

- › Crea tests unitarios para las funciones definidas anteriormente. Ten en cuenta los siguientes casos:
 - Lista vacía
 - Lista de un único elemento
 - Lista con elementos

Inferencia de tipos

- › Cuando se envía una función anónima a una función genérica, es necesario especificar el tipo de los parámetros
 - `dropWhile(lista, (x: Int) => x > 5)`
- › El primer parámetro, ya es una lista de enteros, por lo tanto es redundante indicarle el tipo de la función anónima

Inferencia de tipos

- › Scala permite una forma para que, currificando la función, sea capaz de inferir el tipo de la función anónima.
- › `def dropWhile[A](l: Lista[A], f: A => Boolean)`
 - `dropWhile(lista, (x: Int) => x>5)`
- › `def dropWhile[A](l: Lista[A])(f: A => Boolean)`
 - `dropWhile(lista)(x => x>5)`
 - `dropWhile(lista) (_ >5)`

Ejercicio

- › Define la función `dropWhile` como se muestra en la página anterior. Modifica o duplica los tests para ella.

Ejercicio

- › Define la función `init` que recibida una lista, devuelva una Lista con todos los elementos excepto el último.

- ```
def init[A] (l: Lista[A]): Lista[A] = {
 @annotation.tailrec
 def loop(acc: Lista[A], rest: Lista[A]): Lista[A] = {
 }
 loop(Lista(), l)
}
```