

Programación Funcional en Scala

– Tema 3 –

Constructores de Tipos, Type Classes y Funtores

Jesús López González
jesus.lopez@hablapps.com

Programación Funcional en Scala
Habla Computing

Cursos ETSII-URJC 2015

1 Higher Kinded Types

2 Implicits

3 Type Classes

4 Funtores

5 Takeaways

Constructor de Tipos

Un constructor de tipos es un tipo que recibe argumentos tipo para construir un nuevo tipo. Ejemplos de constructores de tipo pueden ser *List* (recibe un argumento tipo), *Option* (recibe un argumento tipo) o *Map* (recibe dos argumentos tipo).

Listing 1: Usando constructores de tipo

```
scala> val opt: Option[Int] = Some(3)
opt: Option[Int] = Some(3)

scala> val lst: List[String] = List("how", "are", "you")
lst: List[String] = List(how, are, you)

scala> val map: Map[Int, String] = Map(1 -> "one", 2 -> "two")
map: Map[Int,String] = Map(1 -> one, 2 -> two)
```

Kinds

- Proper: $(*)$ Ej: String, Int, List[Int], etc.
- First-order: $(* \rightarrow *)$ Ej: Option[_], List[_], etc.
- Higher-order: $((* \rightarrow *) \rightarrow *)$ Ej: Functor[F[_]], etc.

Listing 2: Función show recibe un constructor de tipos $C (* \rightarrow * \rightarrow *)$

```
scala> def show[C[_], _], A, B](arg: C[A, B]): String =  
    arg.toString  
show: [C[_], _], A, B](arg: C[A,B])String  
  
scala> show(Map(1 -> "one", 2 -> "two"))  
res0: String = Map(1 -> one, 2 -> two)  
  
scala> show(List(1, 2, 3))  
<console>:9: error: inferred kinds of the type arguments  
  (scala.collection.LinearSeqOptimized,Int,List[Int]) do not  
  conform to the expected kinds of the type parameters (type  
  C,type A,type B).
```

1 Higher Kinded Types

2 **Implicits**

3 Type Classes

4 Funtores

5 Takeaways

Intuición

Por norma general, los programadores trabajamos de forma explícita. Cuando queremos aplicar una función indicamos de forma muy clara cuáles son los parámetros de entrada. Además, cuando queremos invocar una función, dejamos plasmada nuestra intención de hacerlo en el código. Scala trae consigo varias técnicas para trabajar de forma implícita, que permiten al compilador tomar decisiones de forma autónoma, siempre tratando de intuir las intenciones del programador. Podemos diferenciar tres grupos:

- Implicit Parameters
- Implicit Defs
- Implicit Classes

Listing 3: Implicit Parameters

```
scala> def add(a: Int)(implicit b: Int): Int = a + b
add: (a: Int)(implicit b: Int)Int

scala> add(1)(2)
res2: Int = 3

scala> implicit val x: Int = 2
x: Int = 2

scala> add(1)
res3: Int = 3
```


Listing 4: Implicit Defs

```
scala> case class Point(x: Int, y: Int)
defined class Point

scala> implicit def tupleToPoint(tuple: (Int, Int)): Point =
    |   Point(tuple._1, tuple._2)
tupleToPoint: (tuple: (Int, Int))Point

scala> val p1: Point = tupleToPoint((1, 2))
p1: Point = Point(1,2)

scala> val p2: Point = (1, 2)
p2: Point = Point(1,2)
```

Listing 5: Implicit Classes

```
scala> implicit class IntExtender(val i: Int) {  
  |   def myNewDef: String = "Def not implemented for Int"  
  | }  
defined class IntExtender  
  
scala> 33.myNewDef  
res0: String = Def not implemented for Int
```

Implícitos y Constructores de Tipos

Existe una notación especial para lidiar con constructores de tipos e implícitos bajo una sintaxis más dulcificada. Tal notación es conocida como *context bound* y es muy útil para proveer a la función de ciertas evidencias que resultan necesarias para su aplicación.

Listing 6: Dulcificación de evidencias implícitas en serialize2

```
scala> trait Serializer[A] {  
  |   def serialize(a: A): String  
  | }  
defined trait Serializer  
  
scala> def serialize[A](a: A)(implicit ev: Serializer[A]):  
  String =  
    |   ev.serialize(a)  
serialize: [A](a: A)(implicit ev: Serializer[A])String  
  
scala> def serialize2[A: Serializer](a: A): String =  
    |   implicitly[Serializer[A]].serialize(a)  
serialize2: [A](a: A)(implicit evidence$1: Serializer[A])String
```

1 Higher Kinded Types

2 Implicits

3 Type Classes

4 Funtores

5 Takeaways

1 Higher Kinded Types

2 Implicits

3 Type Classes

4 Funtores

5 Takeaways

- 1 Higher Kinded Types
- 2 Implicits
- 3 Type Classes
- 4 Funtores
- 5 Takeaways**