Sergio Couto
s.couto@indizen.com

Introducción a Scala y Programación Funcional

Mayo 2018

indizen
Technology & Business Confluence

ÍNDICE

01 Más colecciones

02 Efectos de lado

03 Tratamiento de errores – Excepciones, Option e Either

04 Testing unitario

05 Currificación

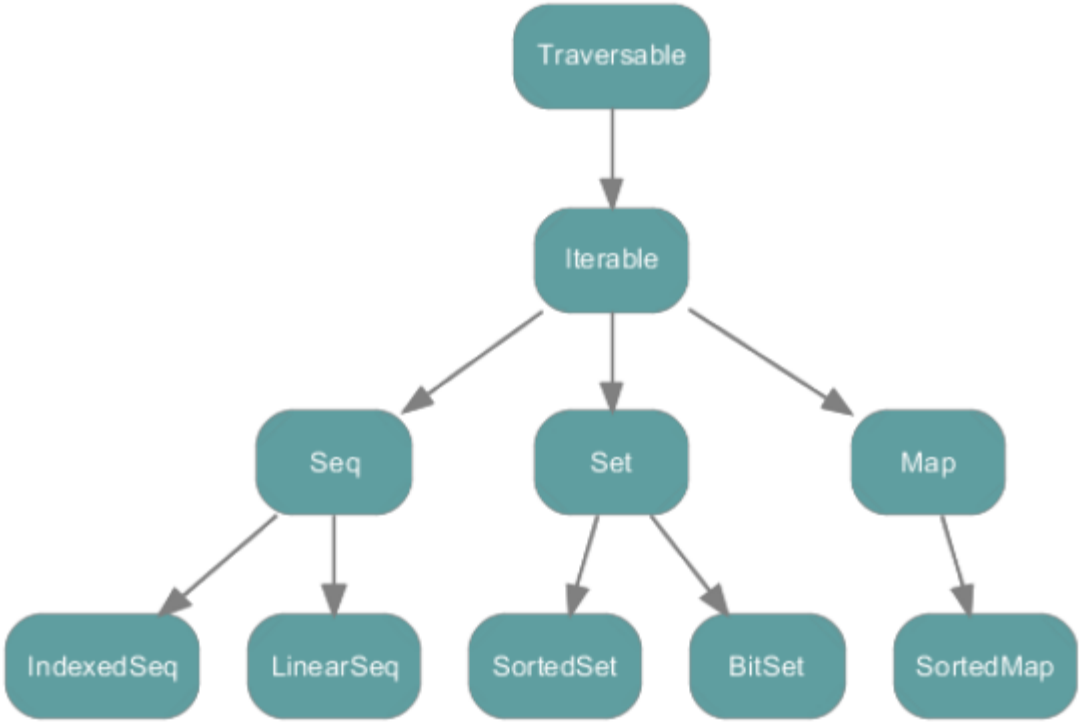
06 Clases

07 Varianza

01 | Más colecciones

indizen
Technology & Business Confluence

- Todas implementan la interfaz Traversable e Iterable
- Misma API o muy parecida.
- Añadir elementos
  - Por el comienzo **+:**
  - Por el final **:+**



- APIs
  - Seq generaliza (*equivalente* a List en Java)
  - List es óptima para LIFO (LinearSeq)
  - Vector es óptima para acceso aleatorio (IndexedSeq
  - Set no permite elementos repetidos (añade y elimina con + y -)
  - Map (añade y elimina con + y -)

01 | **Más colecciones** – Algunos métodos

indizen
Technology & Business Confluence

> Funciones aplicables a Traversable
  - Aplicar sobre cada elemento: foreach, map y flatMap
  - Conversiones: toArray, toList
  - Información: isEmpty, nonEmpty, size
  - Obtener un elemento: head, last, find
  - Obtener un trozo: tail, slice, take, filter, takeWhile, takeRight
  - Eliminar un trozo: drop, dropWhile, dropRight
  - Dividir o agrupoar: partition, groupBy, splitAt
  - Comprobar: forAll, exists, count
  - Crear String: mkString

> Funciones aplicables a Secuencias
  - Ordenación: sorted, sortBy, sortWith
  - Comparación: startsWith, endsWith, contains
  - Información: length
  - Adición: :+ o +:

> Funciones aplicables a Sets
  - Adición: + ,++
  - Eliminación: -, --
  - Comparación: startsWith, endsWith, contains
  - Información: length

01 | **Más colecciones** – Ejercicio con secuencias

indizen
Technology & Business Confluence

> Define una función que añada un elemento al final de una lista
  - def addAtTheEnd(list: List[Int], elem: Int): List[Int] = ???

> Define una función idéntica a la anterior pero que compruebe si el elemento existe antes de añadirlo. Si ya existe, no debe añadirlo
  - def addAtTheEndIfNotExists(list: List[Int], elem: Int): List[Int] = ???

> Define una función que devuelva si una lista de eneros en un palíndromo
  - def isPalindrome(list: List[Int]): Boolean =???

> Define una función que devuelva una lista el doble de su valor para cada elemento par:
  - def doubleIfEven(list: List[Int]): List[Int] = ???

01 | **Más colecciones** – Mapas

indizen
Technology & Business Confluence

- Colección con Clave, valor
  - Cada elemento tiene la sintaxis: clave - > valor
- Recuperación de elementos:
  - m(index), m.get(index)
- Obtener las claves:
  - keys, keySet, keyIterator
- Obtener los valores:
  - values, valuesIterator
- Procesar los valores:
  - mapValues,

- val m = Map(1 - > “one”, 2 - > “two”)
  - m(1)
  - m.get(1)
- A la hora de hacer map, foreach, etc sobre un mapa:
  - Cada elemento se entiende como tupla
  - m.map(x => x._2) = List(one, two)
  - m.map(x => x._1) = List(1, 2)
- Podemos tratarla entera, o descomponerla con pattern matching:
  - m.map{case (k,v) => v} = List(one, two)
  - m.map{case (k,v) => k} = List(1, 2)

01 | **Más colecciones** – Ejercicio con Mapas

indizen
Technology & Business Confluence

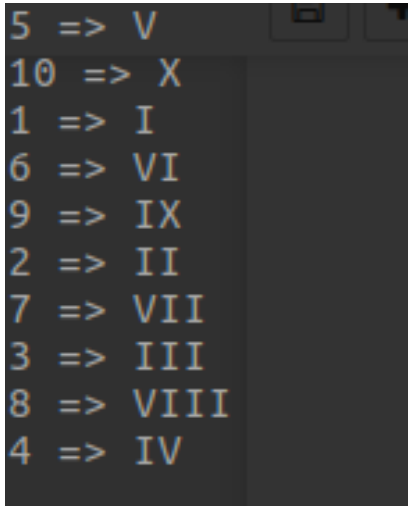
- Ejercicio1: Crea un mapa cualquiera y obtén un elemento. Prueba a obtener un elemento que no exista. ¿Qué pasa? ¿Cuál es la diferencia entre usar m(index) y m.get(index)

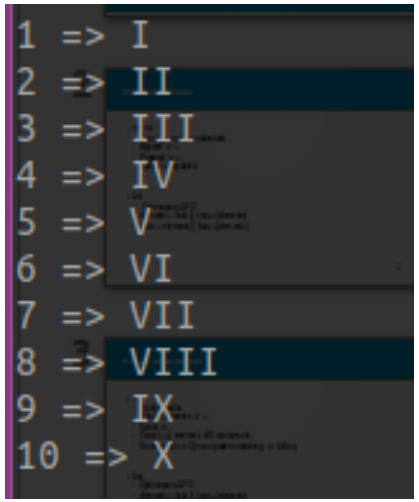
- Ejercicio2: Imprime el mapa *romanos* o uno semejante de forma legible.

- Ejercicio3: Intenta hacerlo ordenado

Pistas:

- Interpolación de Strings
  - `s"Mi edad es ${x}, por lo que naci en ${currentYear-x}"`
  - Recorre el mapa sabiendoque cada elemento es una tupla





02 | Efectos de lado

indizen
Technology & Business Confluence

> Función pura vs función impura

> Ejemplos de efectos de lado:
- Leer o escribir un fichero
- Invocar un servicio web
- Arrancar otro thread
- Lanzar una excepción
- Enviar un email
- Lanzar un misil

> Consecuencias
- Dificultad para mantener un pograma
- Dificultad para comprender un programa
- Pruebas
- Descubrir y solucionar bugs
- Reutilización
- Optimización

02 | **Efectos de lado –** ¿Cómo evitarlos?

indizen
Technology & Business Confluence

- Son necesarios, generalmente, no se pueden evitar
  - Salidas del programa, tratar errores etc..

- Objetivo: Desacoplar parte pura de la parte impura
- Parte pura define qué efectos necesitamos ejecutar
- Parte impura los ejecuta
- Programación monádica
- Tratamiento de errores:
  - Excepciones (try)
  - Try => Success o Failure
  - Option
  - Either

03 | Tratamiento de errres – Excepciones y Try

indizen
Technology & Business Confluence

> Excepciones => Mismo concepto que en Java
  - Salidas del programa, tratar errores etc..
  - ¿Qué hacer cuando la capturo? ¿Qué valor devolver?

> Clase Try del paquete scala.util
> Devuelve Success (encapsulta resultado) o Failure (encapsula mensaje de error)
  - Puede procesarse el resultado con patter matching.
  - Mismo problema que antes, ¿Qué devuelvo?
  - Puedo devolver el propio Try, y que la función llamante se encargue de analizar procesar el error
  - **No es mi problema**

> **Excepción:**

```
try {
  x/y
}catch {
  case e: Exception => println("division entre 0")
}
```

> **Try:**

```
Try(x/y) match {
  case Success(resultado) =>resultado
  case Failure(error) => println(error); x
}
```

  -
  -

03 | Tratamiento de errres – Clase Option

indizen
Technology & Business Confluence

- Dos valores, Some y None
  - Some => encapsula resultado
  - None => No hay resultado
- Debe indicarse el tipo del resultado => Option[T]
- Puede usarse con pattern Matching
- Permite tratar casos especiales sin levantar un efecto de lado
- Recuperación de valores:
  - get, getOrElse

- Ejercicio: Define una función que multiplique un número por un valor opcional. Si no se recibe valor opcional debe multiplicar por defecto 1.5
  - def aplicaInteres(cant: Double, tipo: Option[Double]): Double = ???
- Ejercicio: Define la misma función con la cantidad también opcional. Ahora no siempre podrás dar un resultado, por lo que la salida es otro option.
  - def aplicaInteres(cant: Option[Double], tipo: Option[Double]): Option[Double] = ???

03 | Tratamiento de errres – Clase Either

indizen
Technology & Business Confluence

- Similar a Option, pero aporta información del error
  - Left => Encapsula información del error (por convención)
  - Right => Encapsula el resultado
- Se indica los tipos que puede contener, que no tienen por qué coincidir: Either[U, V]
- Puede usarse con pattern Matching
- Permite tratar casos especiales sin levantar un efecto de lado y aportando información del error
- Recuperación de valores:
- Ejercicio: Redefine la función anterior de modo que aporte información del error
  - def aplicaInteres(cant: Option[Double], tipo: Option[Double]): Either[String, Double] = ???
- Redefínela nuevamente suponiendo que ambos valores de entrada son recibidos con su propio mensaje de error. Si es necesario propaga esos mensajes
  - def aplicaInteres (cant: Either[String, Double], tipo: Either[String, Double]: Either [String, Double] = ???

04 | Testing unitario

indizen
Technology & Business Confluence

> ScalaTest

- FlatSpec => Permite mezclar el test con texto que define el comportamiento esperado
- Matchers => Establece condiciones de éxito tienen por qué coincidir: Either[U, V]

```
val genInteger = for (n <- Gen.choose( min = -500,  max = 500)) yield n
val genIntList = Gen.containerOf[List, Int](genInteger)
```

```
forAll(genIntList, genInteger) { (myList, myInt) =>

  val result = addToList(myList, myInt)
  myList.length +1 shouldEqual result.length
  result.last shouldEqual myInt
}
```

> Scalacheck

- Permite definir propiedades genéricas y generadores de pruebas

| Ejercicios

indizen
Technology & Business Confluence

> Ahora que hemos visto como gestionar casos donde no es posible dar una salida, redefine los métodos second y nth creados en la clase anterior para ello
  - def second (list:List[Int]) = ???
  - def nth (list: List[Int], n : Int) = ??

> Función que devuelva el penúltimo elemento de una lista
  - def penultimate(list: List[Int]): Option[Int] = ???

> Función que duplica cada elemento de la lista x veces
  - def duplicate(list: List[Int], x: Int) : List[Int]

> Función que compruebe si una palabra es un palíndromo
  - def isPalindrome(word: String): Boolean = ???

05 | Currificación

indizen
Technology & Business Confluence

> Una función puede recibir varias listas de argumentos para:
  - Parámetros implícitos
  - Ayudar en inferencia de tipos
  - Aplicarla de forma parcial (con underscore)

> Currificar es transformar una función que usa n elementos en una secuencia de funciones ue usan un argumento

> Útil en procesamiento de estructuras

> Sin currificar
  - def uncurriedSum(x: Int, y: Int) = x+y

> Currificada
  - def curriedSum(x: Int)(y: Int) = x+y

> Ejemplo de uso:
  - val list = List(1,2,3)
  - list.map(curriedSum(5)_)

05 | Currificación - ejercicios

indizen
Technology & Business Confluence

> Define una función uncurry con la siguiente signatura

- def uncurry(f: Int => Int => Int) : (Int, Int) => Int

> Define una función curry con la siguiente signatura

- def curry (f: (Int, Int) => Int) : Int => Int => Int

> Las funciones que acabas de definir sólo valen para enteros, defínelas de forma genérica

Haskel Curry

06 | Clases

indizen
Technology & Business Confluence

- Definicion
  - class Person(var name: String, val age: Int, salario: Int)
- Parámetros
  - private: Parámetro privado
  - var => parametro accesible y reasignable
  - val => parametro accesible
  - Nada => parametro innacesible fuera de la clase
- Creación, lectura y modificación
  - val p = new Person("Pedro", 42, 1100)
  - Convención:
    - Atributo privado: comienza por _
    - Método get sin _, método set acaba en _
- Constructores => el constructor por defecto es la cabecera de la clase
- Constructores extra, 2 alternativas
  - Función this
  - Valor por defecto en atributos

```
class Person (val name: String, val age: Int) {

  def this(name: String) {
    this(name, 0)
  }
}
```

```
class Person (val name: String, val age: Int = 0)
```

06 | Clases - Objects

indizen
Technology & Business Confluence

- Similar a clase singleton de Java
  - class Person(var name: String, val age: Int, salario: Int)
- **Companion Object**
  - Objeto con el mismo nombre que una clase que la acompaña. Se define en el mismo fichero
  - Implementa métodos apply y unapply:
    - apply: Constructor, llama al new
    - Unapply: Recibe un alumno y devuelve un Option con una tupla de sus valores

- **Ejemplo:**

```
class Person(val nombre: String, val age: Int)

object Person {

   def apply(nombre: String, age: Int): Person = ???

   def unapply(p: Person): Option[(String, Int)] = ???

}
```

06 | Clases – Ejercicio companion object

indizen
Technology & Business Confluence

> Define una clase Alumno con los atributos Nombre y apellido

> Define una clase Asignatura con los atributos Nombre, limite de alumnos (por defecto 30) y descripcion (opcional)

> Define sus companion objects

06 | Clases – Traits y case classes

indizen
Technology & Business Confluence

Traits

- Similares a interfaces de Java
- Pueden contener métodos y variables (Pero no parámetros ni constructores)
- Se extienden mediante **extends** o **with** (permite herencia múltiple - **mixin**)
- Existen también clases abstractas, que sí pueden tener constructores (no permite herencia múltiple)

Case classes

- Clases que implementan automáticamente su companion object con sus métodos apply y unapply
- Aportan también un toString por defecto más legible
- Sencillez, evitar escribir de más etc..
- Atributos **val** por defecto (se pueden leer, pero son inmutables)
- Para modificarlos, se emplea el método copy indicándole de 0 a n parámetros
  - p.copy()
  - p.copy(nombre = “other”)
- Se genera también el método equals para la comparación
- No tenemos que escribir explicitamente los métodos => evitar errores

case class Person(nombre: String, apellidos: String)

07 | Varianza

indizen
Technology & Business Confluence

- 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 => Estruct[A] es subtipo de Estruct[B]
  - A <: B => Estruct[A] <: Estruct[B]
- **Contravariante:** Una estructura de subtipos es considerada supertipo de estructura de supertipos
  - Si A es subtipo de B => Estruct[B] es subtipo de Estruct[A]
  - A <: B => Estruct[A] :> Estruct[B]
- **Invariante:** No existe relación entre las estructuras independientemente de la relación entre los tipos
  - A <: B => nada

| Ejercicios

indizen
Technology & Business Confluence

> Función que haga rotar una lista de enteros x lugares hacia la izquierda
  - def rotate(list: List[Int], x: Int): List[Int]
  - def rotate(List(1,2,3,4,5), 2) = List(3,4,5,1,2)
  - def rotate(List(1,2,3,4,5), -2) = List(4,5,1,2,3)

> Define una función que elimine de una lista el primer elemento que satisfaga un predicado. Asegúrate de que sea tail-safe
  - def removeFirstElement(list: List[Int], f: Int => Boolean): List[Int] = ???

> La función anterior vale sólo para enteros, generalízala para cualquier tipo

| Ejercicios

indizen
Technology & Business Confluence

- Usando case classes define una clase alumno (parámetros nombre y apellidos)

- Lo mismo con asignatura los siguientes parámetros:
  - Nombre
  - Plazas: 30 por defecto
  - Descripción: Opcional

- Define una clase Administración con dos métodos, baja y alta que se comportarán de la siguiente forma:
  - Baja: Debe dar de baja un alumno o levantar un error si no es posible (no está matriculado p.e.)
  - Alta: Debe dar de alta a un alumno si hay plazas en la asignatura.

| Ejercicios

indizen
Technology & Business Confluence

- Usando case classes y traits o clases abstractas genera la siguiente estructura para definir dos tipos de alumnos:
  - AlumnoRepetidor
  - AlumnoNuevo
  - Ambos heredarán de Alumno

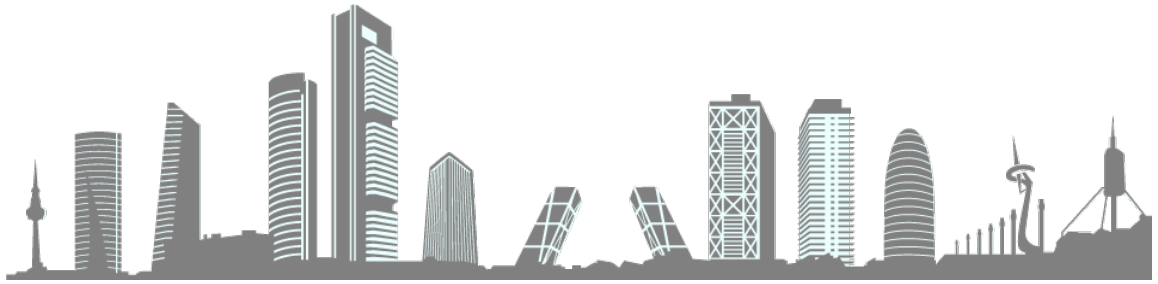
- Lo mismo con las asignaturas
  - AsignaturaConPrioridad
  - AsignaturaSinPrioridad
  - Ambas heredarán de Asignatura

- Define una clase Administración con dos métodos, baja y alta que se comportarán de la siguiente forma:
  - Baja: Debe dar de baja un alumno o levantar un error si no es posible
  - Alta en Asignatura sin prioridad: Debe dar de alta a un alumno si hay plazas en la asignatura.
  - Alta en Asignatura con prioridad: Tienen prioridad los alumnos nuevos. Si se da de alta un nuevo y no hay plazas, debe expulsarse a un repetidor de la asignatura. Pista: Deberás usar una función definida anteriormente en clase


"Con
Indizen
nunca caminarás solo"
indizen
Technology & Business Confluence

indizen

Technology & Business Confluence



GRACIAS

www.indizen.com