

Taller de Scala

TALLER DE CONCEPTOS BÁSICOS DE SCALA

The world is how we shape it

soprasteria

Objetivos

- 
- 01 Introducción a Scala
 - 02 Básicos
 - 03 Preparación del entorno
 - 04 Kata: Números romanos
 - 05 Kata: LinkedLists
 - 06 Kata: BowlingParser



Scala

01

Introducción a Scala

Introducción a Scala

¿Qué es Scala?

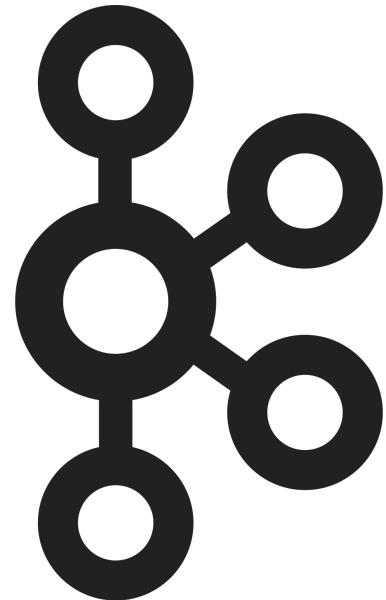
- Scala es un lenguaje de programación que combina características de lenguajes orientados a objetos y funcionales
- Apareció en 2003
- Diseñado por Martin Odersky
- Se ejecuta sobre la JVM
- Es un lenguaje compilado y estáticamente tipado



Introducción a Scala

¿Por qué aprender Scala?

— Es utilizado en Big Data y Backend



Introducción a Scala

¿Por qué aprender Scala?

— Es más conciso y productivo que Java tradicional

Java vs Scala WordCount

```
public class WordCountJava {  
    public static void main(String[] args) {  
        StringTokenizer st  
            = new StringTokenizer(args[0]);  
        Map<String, Integer> map =  
            new HashMap<String, Integer>();  
        while (st.hasMoreTokens()) {  
            String word = st.nextToken();  
            Integer count = map.get(word);  
            if (count == null)  
                map.put(word, 1);  
            else  
                map.put(word, count + 1);  
        }  
        System.out.println(map);  
    }  
}
```

```
> runMain WordCountJava "a b a c a b"  
[info] Running WordCountJava a b a c a b  
{a=3, b=2, c=1}
```



```
object WordCountScala extends App {  
    println(  
        args(0)  
        .split(" ")  
        .groupBy(x => x)  
        .map(t => t._1 -> t._2.length))  
}
```

```
> runMain WordCountScala "a b a c a b"  
[info] Running WordCountScala a b a c a b  
Map(b -> 2, a -> 3, c -> 1)
```

Introducción a Scala

¿Por qué aprender Scala?

— Es intercambiable con Java

- └ Podemos beneficiarnos de su madurez (java.time, multithreading, seguridad...)
- └ Nos permite usar librerías escritas en Java

— Marca el futuro de Java

- └ Java funcional (Stream, Function, Optional...) → Java 8 (2014)
- └ Records (JEP 395) → Java 16 (2021)
- └ Pattern Matching (JEP 407) → Java 17 (2021)



Scala

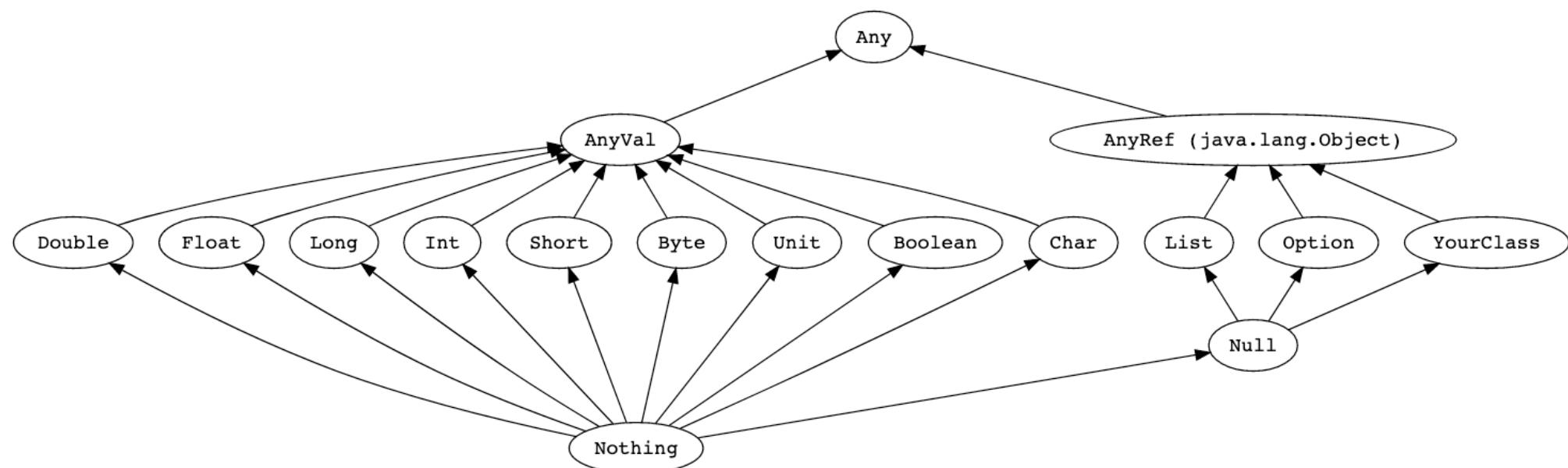
02

Básicos

Básicos

Características del lenguaje

— En Scala todo son objetos



Básicos

Características del lenguaje

— Classes

```
class Point(var x: Int, var y: Int) {  
  
    def move(dx: Int, dy: Int): Unit = {  
        x = x + dx  
        y = y + dy  
    }  
  
    override def toString: String =  
        s"($x, $y")  
}  
  
val point1 = new Point(2, 3)  
println(point1.x) // prints 2  
println(point1) // prints (2, 3)
```

- └ Los atributos x e y son públicos y no finales
- └ La clase Point ya tiene un constructor con todos los atributos
- └ Admite abstract, final
- └ Interpolación de Strings: s"...."

Básicos

Características del lenguaje

Traits

```
trait Iterator[A] {  
    def hasNext: Boolean  
    def next(): A  
}  
  
class IntIterator(to: Int) extends Iterator[Int] {  
    private var current = 0  
    override def hasNext: Boolean = current < to  
    override def next(): Int = {  
        if (hasNext) {  
            val t = current  
            current += 1  
            t  
        } else 0  
    }  
}  
  
val iterator = new IntIterator(10)  
iterator.next() // returns 0  
iterator.next() // returns 1
```

- └ Define dos métodos abstractos, por lo que no se puede remplazar por una expresión lambda
- └ Permite hacerlas “finales” fuera del archivo con el modificador sealed (ver JEP 409)

Básicos

Características del lenguaje

— Objects

```
package logging

object Logger {
  def info(message: String): Unit = println(s"INFO: $message")
}

import logging.Logger.info

class Project(name: String, daysToComplete: Int)

class Test {
  val project1 = new Project("TPS Reports", 1)
  val project2 = new Project("Website redesign", 5)
  info("Created projects") // Prints "INFO: Created projects"
}
```

- └ Es un singleton
- └ No se pueden instanciar
- └ Sus métodos son “estáticos”
- └ Se pueden combinar con otras clases cuando se requieren métodos estáticos

Básicos

Características del lenguaje

— Case classes

```
case class Message(sender: String, recipient: String, body: String)
val message1 = Message("guillaume@quebec.ca", "jorge@catalonia.es", "Ça va ?")

println(message1.sender) // prints guillaume@quebec.ca
message1.sender = "travis@washington.us" // this line does not compile
```

- └ Son clases, pueden implementar tratos y tener métodos propios
- └ Sus atributos son públicos y finales (JEP 395)
- └ Se invoca el constructor omitiendo "new"
- └ Ideales para pattern matching (JEP 405)
- └ Pueden ser case objects como None (Option) o Nil (List)

Básicos

Características del lenguaje

— Tuples

```
val ingredient = ("Sugar", 25)
```

```
println(ingredient._1) // Sugar  
println(ingredient._2) // 25
```

```
val (name, quantity) = ingredient  
println(name)      // Sugar  
println(quantity) // 25
```

- └ Son contenedores de valores, pudiendo almacenar muchos valores de distintos tipos
- └ Son case classes, y tienen un comportamiento especial en el pattern matching

Básicos

Características del lenguaje

— Pattern matching

```
import scala.util.Random

val x: Int = Random.nextInt(10)

x match {
  case 0 => "zero"
  case 1 => "one"
  case 2 => "two"
  case _ => "other"
}
```

- └ Switch con esteroides
- └ El caso `_` es equivalente al default

Básicos

Características del lenguaje

— Pattern matching y case classes

```
sealed trait Notification

case class Email(sender: String, title: String, body: String) extends Notification

case class SMS(caller: String, message: String) extends Notification

case class VoiceRecording(contactName: String, link: String) extends Notification

def showNotification(notification: Notification): String = {
  notification match {
    case Email(sender, title, _) =>
      s"You got an email from $sender with title: $title"
    case SMS(number, message) =>
      s"You got an SMS from $number! Message: $message"
  }
}
```

- Permiten hacer match con el tipo e incluso con los valores de la instancia del case class

Básicos

Características del lenguaje

— Pattern matching y pattern guards

```
notification match {
    case Email(sender, _, _) if importantPeopleInfo.contains(sender) =>
        "You got an email from special someone!"
    case SMS(number, _) if importantPeopleInfo.contains(number) =>
        "You got an SMS from special someone!"
    case other =>
        showNotification(other) // nothing special, delegate to our original
}
```

03

Preparación del entorno

Configuración de IntelliJ IDEA



The world is how we shape it

soprasosteria



04

Kata: Números romanos

Kata: Números romanos

¿Cómo usarlos?

— Son 7 símbolos

I → 1

V → 5

X → 10

L → 50

C → 100

D → 500

M → 1.000

— Se ordenan de mayor a menor de izquierda a derecha sumándose sus valores

Ejemplo: XVII → 17 o MMXXII → 2.022

— No se puede repetir tres veces seguidas el mismo número

Level 3 [Tahoma Regular 14 pt]

Level 4 [Tahoma Regular 12 pt]

Kata: Números romanos

¿Cómo usarlos?

— **No se puede repetir más de tres veces seguidas el mismo número**

Ejemplo: IX en lugar de VIIII

— **Tampoco se puede repetir un símbolo cuando exista valor para la suma**

Ejemplo: M en lugar de DD

— **Números naturales entre 1 y 3.000**

— **Base 10**



Kata: RomanDecoder

Kata: RomanEncoder

The world is how we shape it

sopra  steria



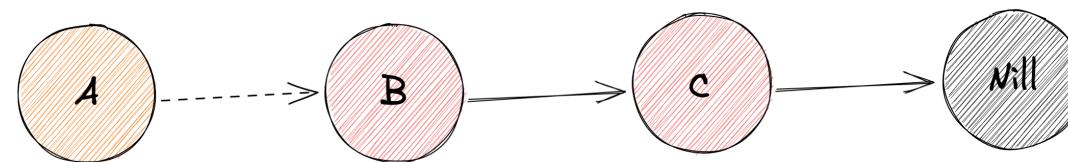
05

Kata: **LinkedLists**

Kata: LinkedLists

¿Qué son?

— Concepto



— LinkedList vs ArrayList

	ArrayList	LinkedList
get()	O(1)	O(n)
add()	O(1)	O(1) amortized
remove()	O(n)	O(n)

Algorithm	array ArrayList	LinkedList
access front	O(1)	O(1)
access back	O(1)	O(1)
access middle	O(1)	O(N)
insert at front	O(N)	O(1)
insert at back	O(1)	O(1)
insert in middle	O(N)	O(1)

Kata: LinkedLists

¿Qué son?

— Se trata de la lista por defecto en Scala

```
@SerialVersionUID(3L)
sealed abstract class List[+A]
  extends AbstractSeq[A]
  with LinearSeq[A]
  with LinearSeqOps[A, List, List[A]]
  with StrictOptimizedLinearSeqOps[A, List, List[A]]
  with StrictOptimizedSeqOps[A, List, List[A]]
  with IterableFactoryDefaults[A, List]
  with DefaultSerializable {
```

```
val ints = List(1, 2, 3)
val names = List("Joel", "Chris", "Ed")
```

```
val list = 1 :: 2 :: 3 :: Nil
```

— Tiene dos implementaciones

```
case object Nil extends List[Nothing] {
  override def head: Nothing = throw new NoSuchElementException("head of empty list")
  override def headOption: Option[Nothing] = None
  override def tail: Nothing = throw new UnsupportedOperationException("tail of empty list")
  override def last: Nothing = throw new NoSuchElementException("last of empty list")
  override def init: Nothing = throw new UnsupportedOperationException("init of empty list")
  override def knownSize: Int = 0
  override def iterator: Iterator[Nothing] = Iterator.empty
  override def unzip[A1, A2](implicit asPair: Nothing => (A1, A2)): (List[A1], List[A2]) = EmptyUnzip

  @transient
  private[this] val EmptyUnzip = (Nil, Nil)
}
```

```
final case class ::[+A](override val head: A, private[scala] var next: List[A @uncheckedVariance])
  extends List[A] {
  releaseFence()
  override def headOption: Option[A] = Some(head)
  override def tail: List[A] = next
}
```

Kata: LinkedLists



The world is how we shape it

soprasteria



06

Kata: BowlingParser

Kata: BowlingParser

El juego



- └ Consiste en derribar más bolos que el rival
- └ Cada jugador cuenta con 10 turnos
- └ En cada turno, hay que derribar 10 bolos

Kata: BowlingParser

Sistema puntuación

— Existen cuatro tipos de tirada (frame)

└ Open

En dos tiradas, el jugador no ha derribado los 10 bolos

Su puntuación es la suma de las tiradas

└ Spare (/)

El jugador ha derribado los 10 bolos en la segunda tirada

Su puntuación es la suma de las tiradas (10) más la siguiente tirada

└ Strike (X)

El jugador derriba los 10 bolos en la primera tirada

Su puntuación es 10 más las dos siguientes tiradas

└ Bonus

El jugador derriba los 10 bolos en su último turno (primera o segunda tirada), lo que le permite tener una tirada extra

La puntuación del frame es la suma de las tres tiradas

1	2	3	4	5	6	7	8	9	10
8-	7-	53	9/	9/	X	8-	51	3/	9-
8	15	23	42	62	80	88	94	113	122

1	2	3	4	5	6	7	8	9	10
8/	9-	44	72	9-	X	X	8-	35	9/7
19	28	36	45	54	82	100	108	116	133

Kata: BowlingParser



The world is how we shape it

soprasteria

Links



Scala Tour



Codewars



Functional
Programming in Scala



Github



LinkedIn



Gracias.



Text on several lines.