I Jornadas @CorunaDev

Scala
¿Java sin puntos y coma?

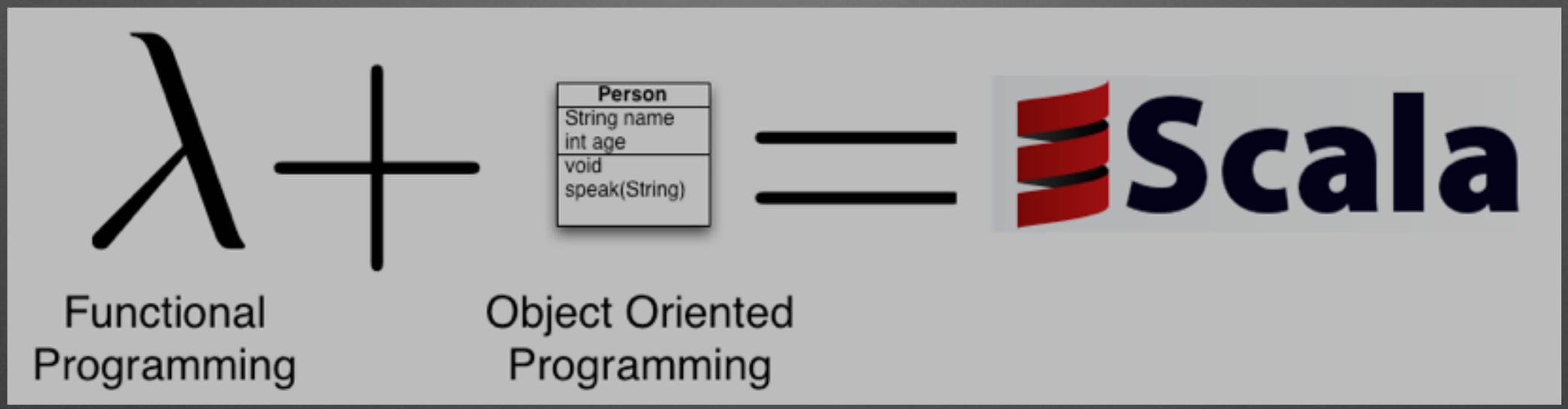
Pedro Vale Ramos
@pedrovr_

POP SPORTS


Scala

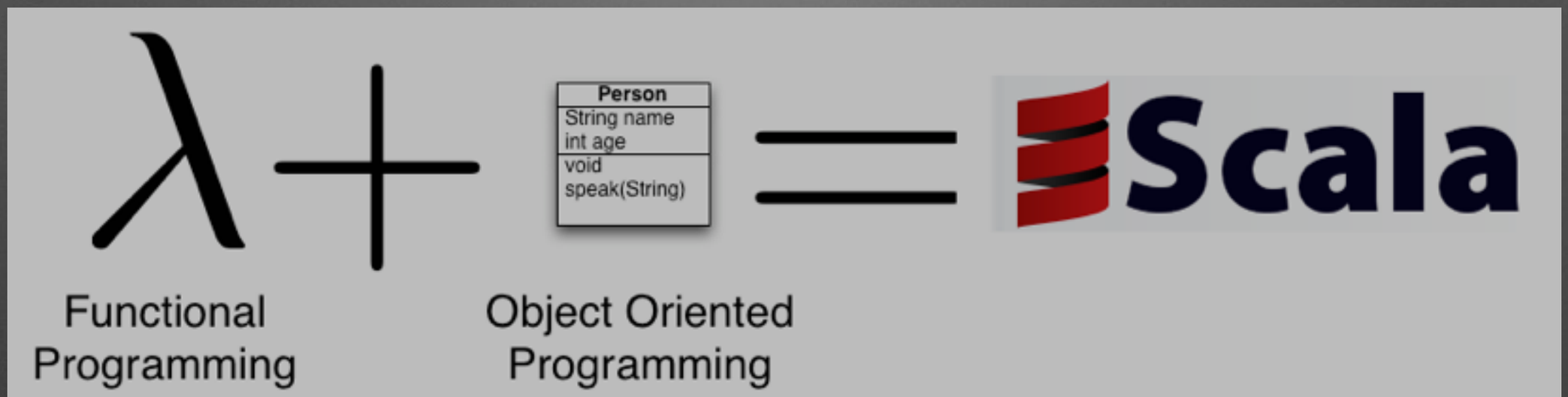
Scala

Lenguaje híbrido



Scala

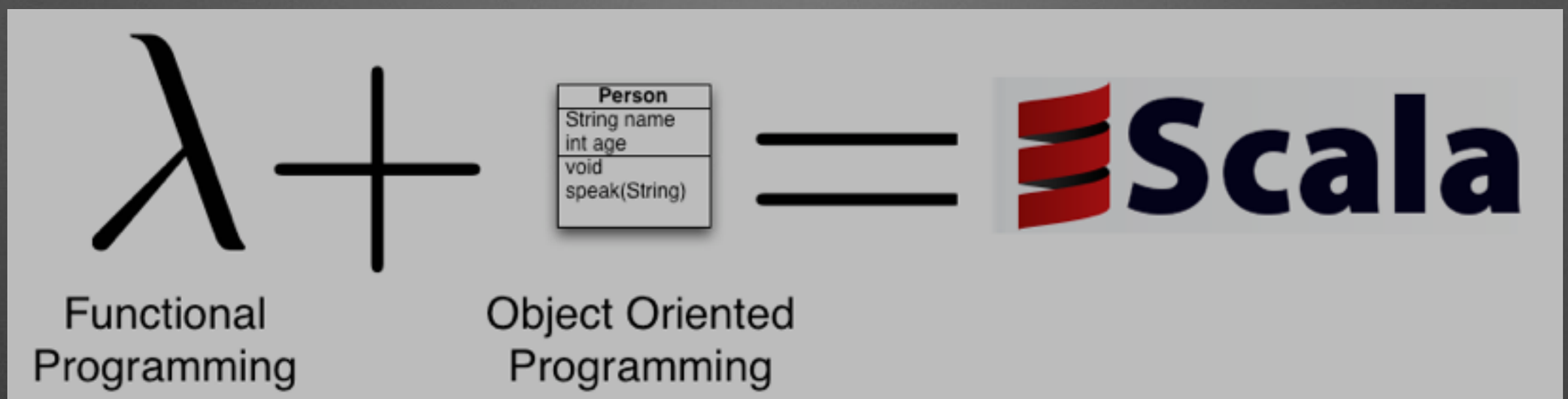
Lenguaje híbrido



Orientado a objetos

Scala

Lenguaje híbrido

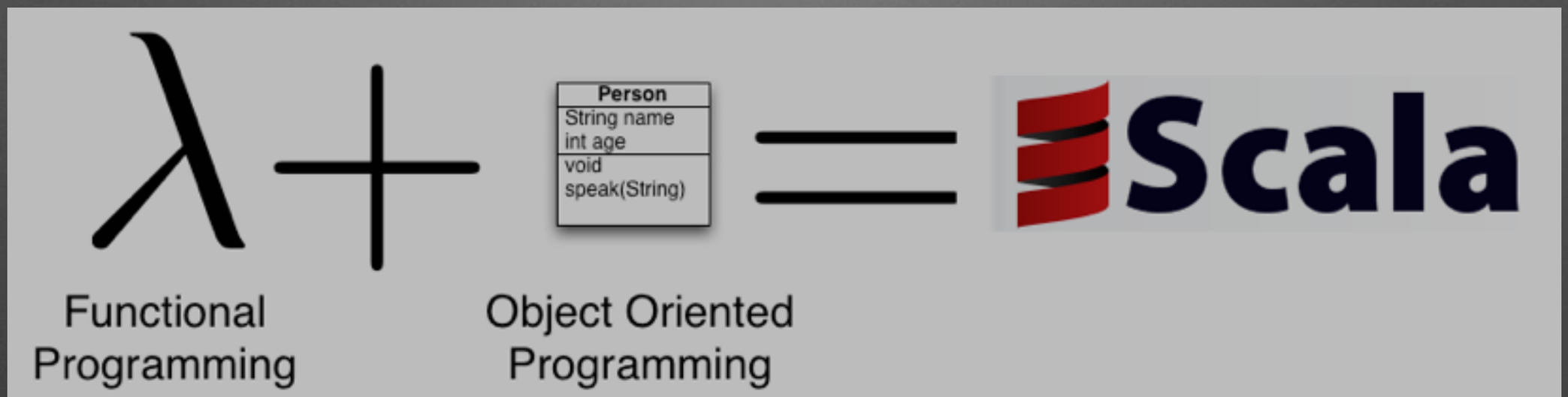


Orientado a objetos

Funcional

Scala

Lenguaje híbrido



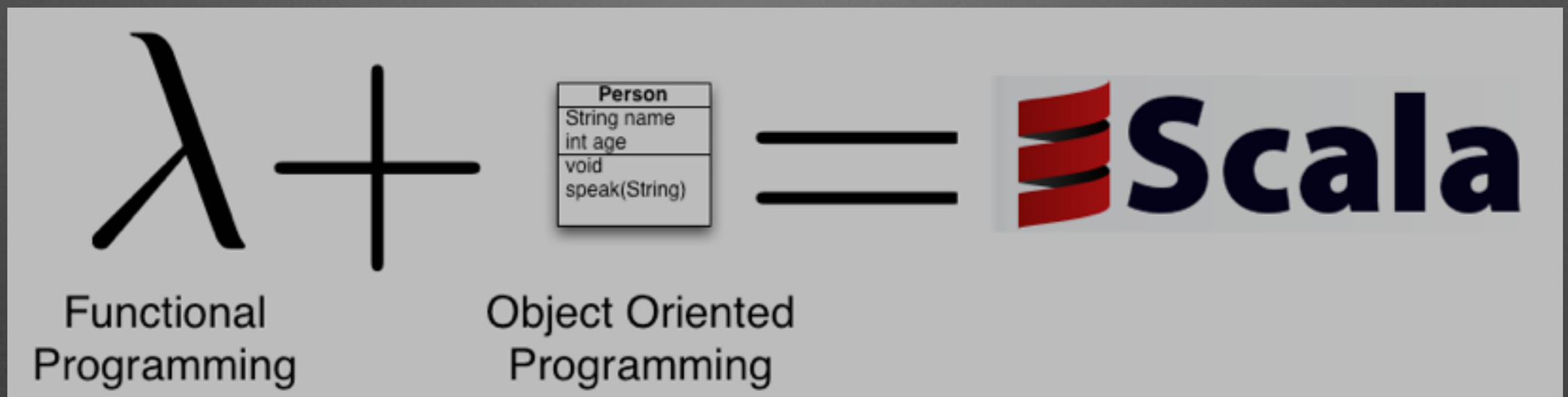
Orientado a objetos

Corre sobre la JVM

Funcional

Scala

Lenguaje híbrido



Orientado a objetos

Corre sobre la JVM

Funcional

Compatible con Java

Orientación a Objetos

Orientación a Objetos

- Clases

```
class Runner extends Person {……}
val runner = new Runner()
```

Orientación a Objetos

- Clases

```
class Runner extends Person {……}
val runner = new Runner()
```

- Case classes

```
case class Point (x: Double, y: Double, tag: String)
val ubicacion = Point(43.2233, -8.5122, “Mi ubicación”)
```

Orientación a Objetos

- Clases

```
class Runner extends Person {……}
val runner = new Runner()
```

- Case classes

```
case class Point (x: Double, y: Double, tag: String)
val ubicacion = Point(43.2233, -8.5122, “Mi ubicación”)
```

- Traits

```
trait ConsoleLogger extends Logger {……}
trait Authenticator {……}
class Races extends ConsoleLogger with Authenticator {……}
```

Orientación a Objetos

- Clases

```
class Runner extends Person {……}
val runner = new Runner()
```

- Case classes

```
case class Point (x: Double, y: Double, tag: String)
val ubicacion = Point(43.2233, -8.5122, “Mi ubicación”)
```

- Traits

```
trait ConsoleLogger extends Logger {……}
trait Authenticator {……}
class Races extends ConsoleLogger with Authenticator {……}
```

- Companion objects

```
object Point {
  def distance (a: Double, b: Double) = {……}
  val ecuador = Point(0,0)
}
```

Funcional

Funcional

- Funciones son de “primer orden”

Funcional

- Funciones son de “primer orden”
  - Funciones reciben funciones

```
def double(x: Long) = x * 2
val dobles = 1 to 10 map { double _ }
```

Funcional

- Funciones son de “primer orden”
  - Funciones reciben funciones

```
def double(x: Long) = x * 2
val dobles = 1 to 10 map { double _ }
```

  - Funciones en variables

```
val triple = (x: Long) => x * 3
val nueve = triple(3)
val triples = 1 to 10.map(triple)
```

Funcional

Funcional

- Valores inmutables

Funcional

- Valores inmutables
  - Aproximación más matemática

Funcional

- Valores inmutables
  - Aproximación más matemática `x ≠ f(x) => x ≠ x + 1`

Funcional

- Valores inmutables
  - Aproximación más matemática `x ≠ f(x) => x ≠ x + 1`
  - En scala se permiten variables mutables
    - `var` para variables
    - `collections.mutable.*` para colecciones

Funcional

- Valores inmutables
  - Aproximación más matemática `x ≠ f(x) => x ≠ x + 1`
  - En scala se permiten variables mutables
    - `var` para variables
    - `collections.mutable.*` para colecciones
- Tuplas

Funcional

- Valores inmutables
  - Aproximación más matemática `x ≠ f(x) => x ≠ x + 1`
  - En scala se permiten variables mutables
    - `var` para variables
    - `collections.mutable.*` para colecciones

- Tuplas
  - Conjuntos de valores encapsulados

```
val person = (“Juan”, 1.78, 65)
```

Funcional

- Valores inmutables
  - Aproximación más matemática `x ≠ f(x) => x ≠ x + 1`
  - En scala se permiten variables mutables
    - `var` para variables
    - `collections.mutable.*` para colecciones

- Tuplas
  - Conjuntos de valores encapsulados

```
val person = (“Juan”, 1.78, 65)
```

  - Se pueden extraer en variables independientes

```
val (name, height, weight) = person
```

Lenguaje

Lenguaje

- Fuertemente tipado con inferencia de tipos

Lenguaje

- Fuertemente tipado con inferencia de tipos
  - Declaramos val o var

```
val name = “Pedro”
var x = 10
val runners = List(“Carlos”,”Pepe”,”Juan”)
```

Lenguaje

- Fuertemente tipado con inferencia de tipos
  - Declaramos val o var

```
val name = “Pedro”
var x = 10
val runners = List(“Carlos”,”Pepe”,”Juan”)
```

  - Si lo necesitamos, fijamos el tipo

```
val name:String = “Pedro”
var x:Int = 10
```

Lenguaje

- Fuertemente tipado con inferencia de tipos
  - Declaramos `val` o `var`

```
val name = “Pedro”
var x = 10
val runners = List(“Carlos”,”Pepe”,”Juan”)
```

  - Si lo necesitamos, fijamos el tipo

```
val name:String = “Pedro”
var x:Int = 10
```

- ¿Donde está el `return`?

Lenguaje

- Fuertemente tipado con inferencia de tipos
  - Declaramos val o var

```
val name = “Pedro”
var x = 10
val runners = List(“Carlos”,”Pepe”,”Juan”)
```

  - Si lo necesitamos, fijamos el tipo

```
val name:String = “Pedro”
var x:Int = 10
```

- ¿Donde está el return?
  - Los bloques {……} retornan el resultado de su última línea

```
val distancia = {
  val dx = x - x0
  val dy = y - y0
  sqrt((dx * dx) + (dy * dy))
}
```

```
val x = if(n > 0) {
  1
} else {
 -1
}
```

Lenguaje

- Fuertemente tipado con inferencia de tipos
  - Declaramos val o var

```
val name = “Pedro”
var x = 10
val runners = List(“Carlos”,”Pepe”,”Juan”)
```

  - Si lo necesitamos, fijamos el tipo

```
val name:String = “Pedro”
var x:Int = 10
```

- ¿Donde está el return?
  - Los bloques {……} retornan el resultado de su última línea

```
val distancia = {
  val dx = x - x0
  val dy = y - y0
  sqrt((dx * dx) + (dy * dy))
}
```

```
val x = if(n > 0) {
  1
} else {
 -1
}
```

  - Se puede escribir return para especificar un retorno

```
for (i <- 1 to 100) {
  if(i % 50) return i
}
```

Lenguaje

Lenguaje

- Options

Lenguaje

- Options
  - Evita el tratamiento de nulos

Lenguaje

- Options
  - Evita el tratamiento de nulos
  - Métodos de utilidad `isEmpty`, `getOrElse`, …

```
case class User(name: String, alias: Option[String])
val users = service.findUsers()
users.map(u => println(u.alias.getOrElse(“”)))
```

Lenguaje

- Options
  - Evita el tratamiento de nulos
  - Métodos de utilidad `isEmpty`, `getOrElse`, …

```
case class User(name: String, alias: Option[String])
val users = service.findUsers()
users.map(u => println(u.alias.getOrElse(“”)))
```

- Pattern matching muy potente

Lenguaje

- Options
  - Evita el tratamiento de nulos
  - Métodos de utilidad `isEmpty`, `getOrElse`, …

```
case class User(name: String, alias: Option[String])
val users = service.findUsers()
users.map(u => println(u.alias.getOrElse(“”)))
```

- Pattern matching muy potente

```
val x = y match {
  case ‘a’ => …
  case ch if exist(ch) => …
  case p: Point => …
  case (x,y) => …
  case Some(y) => …
  case _ => …
}
```

Lenguaje

Lenguaje

- Traits

Lenguaje

- Traits
  - Pueden tener implementación

```
trait Logger {
  def log(msg: String)
  def info(msg: String) = {
    log(“INFO “ + msg)
  }
  def warn(msg:String) …
}
trait ConsoleLogger extends Logger {
  override def log(msg: String) =
    println(msg)
}
```

Lenguaje

- Traits
  - Pueden tener implementación

```
trait Logger {
  def log(msg: String)
  def info(msg: String) = {
    log(“INFO “ + msg)
  }
  def warn(msg:String) …
}
trait ConsoleLogger extends Logger {
  override def log(msg: String) =
    println(msg)
}
```

  - Pueden heredar de una clase

```
trait LoggedException extends
  Exception with Logger {
    def log(msg:String) = {
      println(getMessage())
  }
```

Lenguaje

- Traits
  - Pueden tener implementación

```
trait Logger {
  def log(msg: String)
  def info(msg: String) = {
    log(“INFO “ + msg)
  }
  def warn(msg:String) …
}
trait ConsoleLogger extends Logger {
  override def log(msg: String) =
    println(msg)
}
```

  - Pueden heredar de una clase

```
trait LoggedException extends
  Exception with Logger {
    def log(msg:String) = {
      println(getMessage())
  }
```

  - Pueden contener atributos, tanto fijos como abstractos

```
trait ShortLogger extends Logger {
  val maxLength = 10
  def log(msg:String) = {…}
}
```

```
trait ShortLogger extends Logger {
  val maxLength: Int
  def log(msg:String) = {…}
}
```

Lenguaje

- Traits
  - Pueden tener implementación

```
trait Logger {
  def log(msg: String)
  def info(msg: String) = {
    log(“INFO “ + msg)
  }
  def warn(msg:String) …
}
trait ConsoleLogger extends Logger {
  override def log(msg: String) =
    println(msg)
}
```

  - Pueden heredar de una clase

```
trait LoggedException extends
  Exception with Logger {
    def log(msg:String) = {
      println(getMessage())
  }
```

  - Pueden contener atributos, tanto fijos como abstractos

```
trait ShortLogger extends Logger {
  val maxLength = 10
  def log(msg:String) = {…}
}
```

```
trait ShortLogger extends Logger {
  val maxLength: Int
  def log(msg:String) = {…}
}
```

  - Se “mezclan” de derecha a izquierda

```
class RaceStatus extends ConsoleLogger with ShortLogger {…}
```

Lenguaje

Lenguaje

- Clases del lenguaje supervitaminadas

Lenguaje

- Clases del lenguaje supervitaminadas

  Strings

```
val helloWorld = “Hello World”
helloWorld.map(c => (c.toByte + 1).toChar)
helloWorld.foreach(println _ )
helloWorld.take(4).toLower  //hell
helloWorld == “Hello World” //true
```

Lenguaje

- Clases del lenguaje supervitaminadas

Strings

```
val helloWorld = “Hello World”
helloWorld.map(c => (c.toByte + 1).toChar)
helloWorld.foreach(println _ )
helloWorld.take(4).toLower  //hell
helloWorld == “Hello World” //true
```

Long, Integer, Double,…

```
val rango = 2.to(50) //Range[Int]
val decimal: Double = 2.toDouble
decimal == 2         //true
decimal.isInfinity
decimal.toRadians
```

Lenguaje

- Clases del lenguaje supervitaminadas

Strings

```
val helloWorld = “Hello World”
helloWorld.map(c => (c.toByte + 1).toChar)
helloWorld.foreach(println _ )
helloWorld.take(4).toLower  //hell
helloWorld == “Hello World” //true
```

Long, Integer, Double,…

```
val rango = 2.to(50) //Range[Int]
val decimal: Double = 2.toDouble
decimal == 2         //true
decimal.isInfinity
decimal.toRadians
```

Collections

```
val numeros: Seq[Int] = 1 to 10 toSeq
val dobles = numeros.map(_ * 2)
val pares = numeros.filter(_ % 2)
val primero = numeros.head
val algunos = numeros.drop(2).take(5)
val total = numeros.reduce(_ + _)
val grupos = numeros.groupBy(_ % 3)
```

```
val inversos = numeros.reverse
numeros.foreach(println _)
val tuplas = numeros zip inversos
val withCol = numeros ++ algunos
val withNumber = numeros +: 50
val sinEl5 = numeros - 5
val sinPares = numeros -- pares
```

¿Java sin puntos y coma?

¿Java sin puntos y coma?

- Sumatorio

¿Java sin puntos y coma?

- Sumatorio

```
Integer total = 0;
for(Race race : runner.getRaces()) {
  Integer d = race.getDistance();
  if(d != null) {
    total += d;
  }
}
```

¿Java sin puntos y coma?

- Sumatorio

```
Integer total = 0;
for(Race race : runner.getRaces()) {
  Integer d = race.getDistance();
  if(d != null) {
    total += d;
  }
}
```

```
var total = 0
for(race <- runner.getRaces()) {
  var d = race.getDistance()
  if(!d.isEmpty) {
    total += d.get
  }
}
```

¿Java sin puntos y coma?

- Sumatorio

```
Integer total = 0;
for(Race race : runner.getRaces()) {
  Integer d = race.getDistance();
  if(d != null) {
    total += d;
  }
}
```

¿Java sin puntos y coma?

- Sumatorio

```
Integer total = 0;
for(Race race : runner.getRaces()) {
  Integer d = race.getDistance();
  if(d != null) {
    total += d;
  }
}
```

```
val total = runner.getRaces().reduce(
  _ + _.getDistance().getOrElse(0)
)
```

¿Java sin puntos y coma?

- Sumatorio

```
Integer total = 0;
for(Race race : runner.getRaces()) {
  Integer d = race.getDistance();
  if(d != null) {
    total += d;
  }
}
```

```
val total = runner.getRaces().reduce(
  _ + _.getDistance().getOrElse(0)
)
```

- Manejo de subclases

¿Java sin puntos y coma?

- Sumatorio

```
Integer total = 0;
for(Race race : runner.getRaces()) {
  Integer d = race.getDistance();
  if(d != null) {
    total += d;
  }
}
```

```
val total = runner.getRaces().reduce(
  _ + _.getDistance().getOrElse(0)
)
```

- Manejo de subclases

```
if(employee instanceof Salesman) {
  Salesman s = (Salesman)employee;
  s.sell();
} else if (employee instanceof
    Programmer) {
  ……
} else if … {
…
} else {
  throw new InvalidEmployeeException()
}
```

¿Java sin puntos y coma?

- Sumatorio

```
Integer total = 0;
for(Race race : runner.getRaces()) {
  Integer d = race.getDistance();
  if(d != null) {
    total += d;
  }
}
```

```
val total = runner.getRaces().reduce(
  _ + _.getDistance().getOrElse(0)
)
```

- Manejo de subclases

```
if(employee instanceof Salesman) {
  Salesman s = (Salesman)employee;
  s.sell();
} else if (employee instanceof
    Programmer) {
  ……
} else if … {
…
} else {
  throw new InvalidEmployeeException()
}
```

```
employee match {
  case s: Salesman => c.sell()
  case p: Programmer => p.code()
  case e: Employee => e.work()
  case _ => throw new InvalidEmployeeException()
}
```

Más ejemplos

Más ejemplos

- Manejo de nulos

Más ejemplos

- Manejo de nulos

```
if(time != null) {
  if(distance != null) {
    speed = distance / time;
  } else {
    speed = 0;
  }
} else {
  speed = 0;
}
```

Más ejemplos

- Manejo de nulos

```
if(time != null) {
  if(distance != null) {
    speed = distance / time;
  } else {
    speed = 0;
  }
} else {
  speed = 0;
}
```

```
(time, distance) match {
  case (Some(time),Some(distance)) =>
      distance / time
  case _ => 0
}
```

Más ejemplos

- Manejo de nulos

```
if(time != null) {
  if(distance != null) {
    speed = distance / time;
  } else {
    speed = 0;
  }
} else {
  speed = 0;
}
```

```
(time, distance) match {
  case (Some(time),Some(distance)) =>
      distance / time
  case _ => 0
}
```

- Filtrado de listas

Más ejemplos

- Manejo de nulos

```
if(time != null) {
  if(distance != null) {
    speed = distance / time;
  } else {
    speed = 0;
  }
} else {
  speed = 0;
}
```

```
(time, distance) match {
  case (Some(time),Some(distance)) =>
      distance / time
  case _ => 0
}
```

- Filtrado de listas

```
List<Runner> pedros =
          new ArrayList<Runner>();
for(Runner r : runners) {
  if(r.name.equals(“Pedro”)) {
    pedros.add(r);
  }
}
```

Más ejemplos

- Manejo de nulos

```
if(time != null) {
  if(distance != null) {
    speed = distance / time;
  } else {
    speed = 0;
  }
} else {
  speed = 0;
}
```

```
(time, distance) match {
  case (Some(time),Some(distance)) =>
      distance / time
  case _ => 0
}
```

- Filtrado de listas

```
List<Runner> pedros =
          new ArrayList<Runner>();
for(Runner r : runners) {
  if(r.name.equals(“Pedro”)) {
    pedros.add(r);
  }
}
```

```
val pedros = runners.filter(_.name == “Pedro”)
```

Mi Stack Actual

Mi Stack Actual

https://www.playframework.com/

http://slick.typesafe.com/

http://www.scala-sbt.org/

http://www.scalatest.org/

http://www.postgresql.org/

Bonus Track: Cake Pattern

powered by @rafbermudez

Bonus Track: Cake Pattern

powered by @rafbermudez

```
trait UserRepositoryComponent {
  val userRepository: UserRepository

  class UserRepository {
    ...
  }
}
```

Bonus Track: Cake Pattern

powered by @rafbermudez

```
trait UserRepositoryComponent {
  val userRepository: UserRepository

  class UserRepository {
    ...
  }
}

trait UserServiceComponent {
  this: UserRepositoryComponent =>

  val userService: UserService

  class UserService {
    ...
  }
}
```

Bonus Track: Cake Pattern

powered by @rafbermudez

```
trait UserRepositoryComponent {
  val userRepository: UserRepository

  class UserRepository {
    ...
  }
}

trait UserServiceComponent {
  this: UserRepositoryComponent =>

  val userService: UserService

  class UserService {
    ...
  }
}
```

```
object ComponentRegistry extends
  UserServiceComponent with
  UserRepositoryComponent
{
  val userRepository = new UserRepository
  val userService = new UserService
}
```

Bonus Track: Cake Pattern

powered by @rafbermudez

```
trait UserRepositoryComponent {
  val userRepository: UserRepository

  class UserRepository {
    ...
  }
}

trait UserServiceComponent {
  this: UserRepositoryComponent =>

  val userService: UserService

  class UserService {
    ...
  }
}
```

```
object ComponentRegistry extends
  UserServiceComponent with
  UserRepositoryComponent
{
  val userRepository = new UserRepository
  val userService = new UserService
}


val service = ComponentRegistry.userService
```

Call for Action

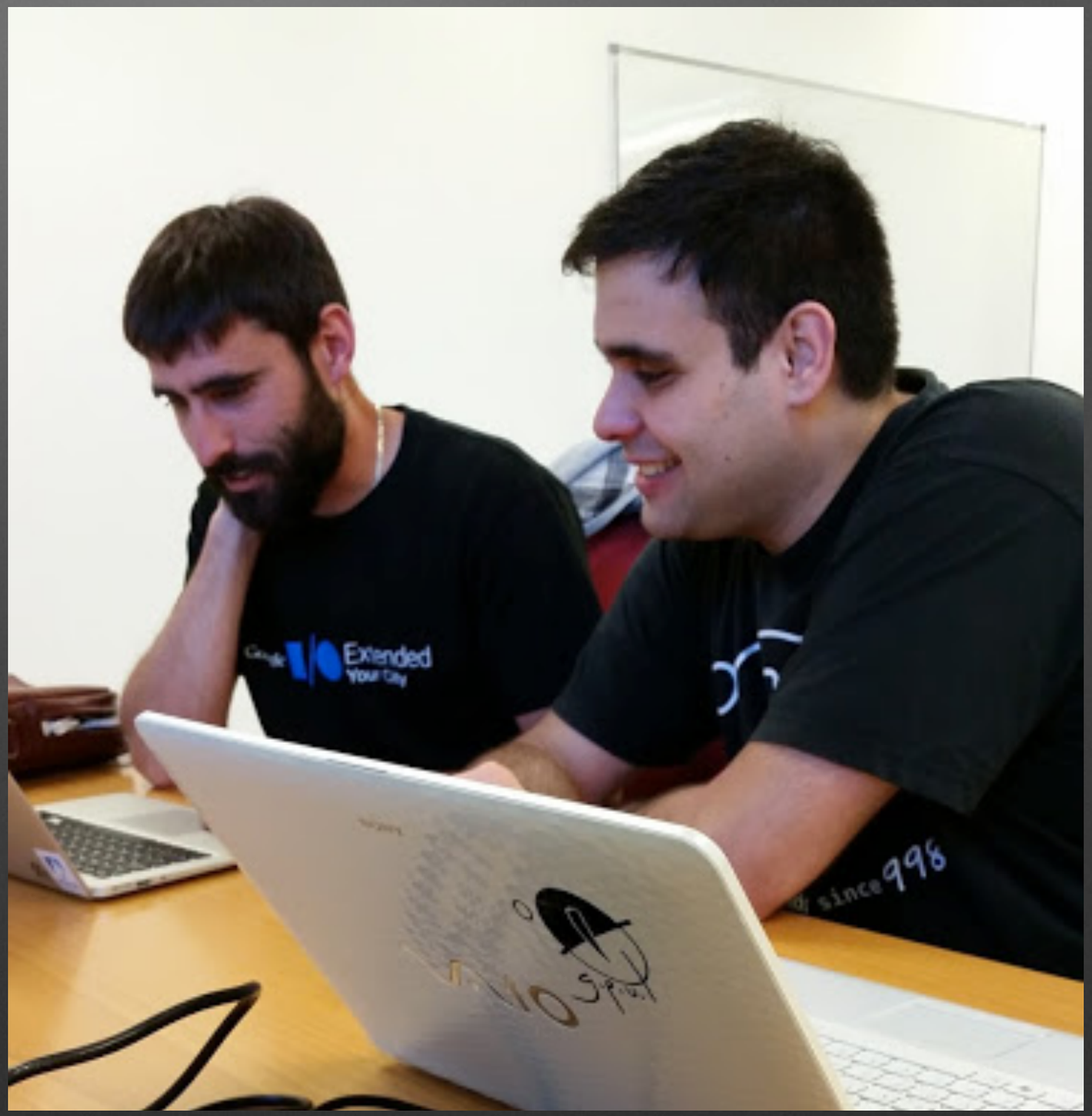
Call for Action

WorkGroup

Call for Action

Scala

WorkGroup



I Jornadas @CorunaDev

Gracias

Pedro Vale Ramos
@pedrovr_

POP SPORTS