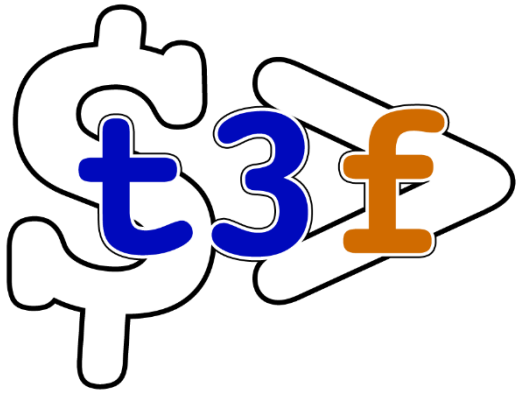
Leganés
12-13 Febrero 2015



Ignacio Navarro Martín @inavarromartin

Introducción a Scala

Except where otherwise noted, this work is licensed under: http://creativecommons.org/licenses/by-nc-sa/3.0/

ÍNDICE

- ★ Background
- ★ Enlaces e información
- ★ Inmutabilidad, valores y funciones
- ★ Tipado
- ★ Estructuras de datos
- ★ Orientación a objetos
- ★ Programación Funcional
- ★ Conclusiones

Scala: Background

Crece contigo

Background

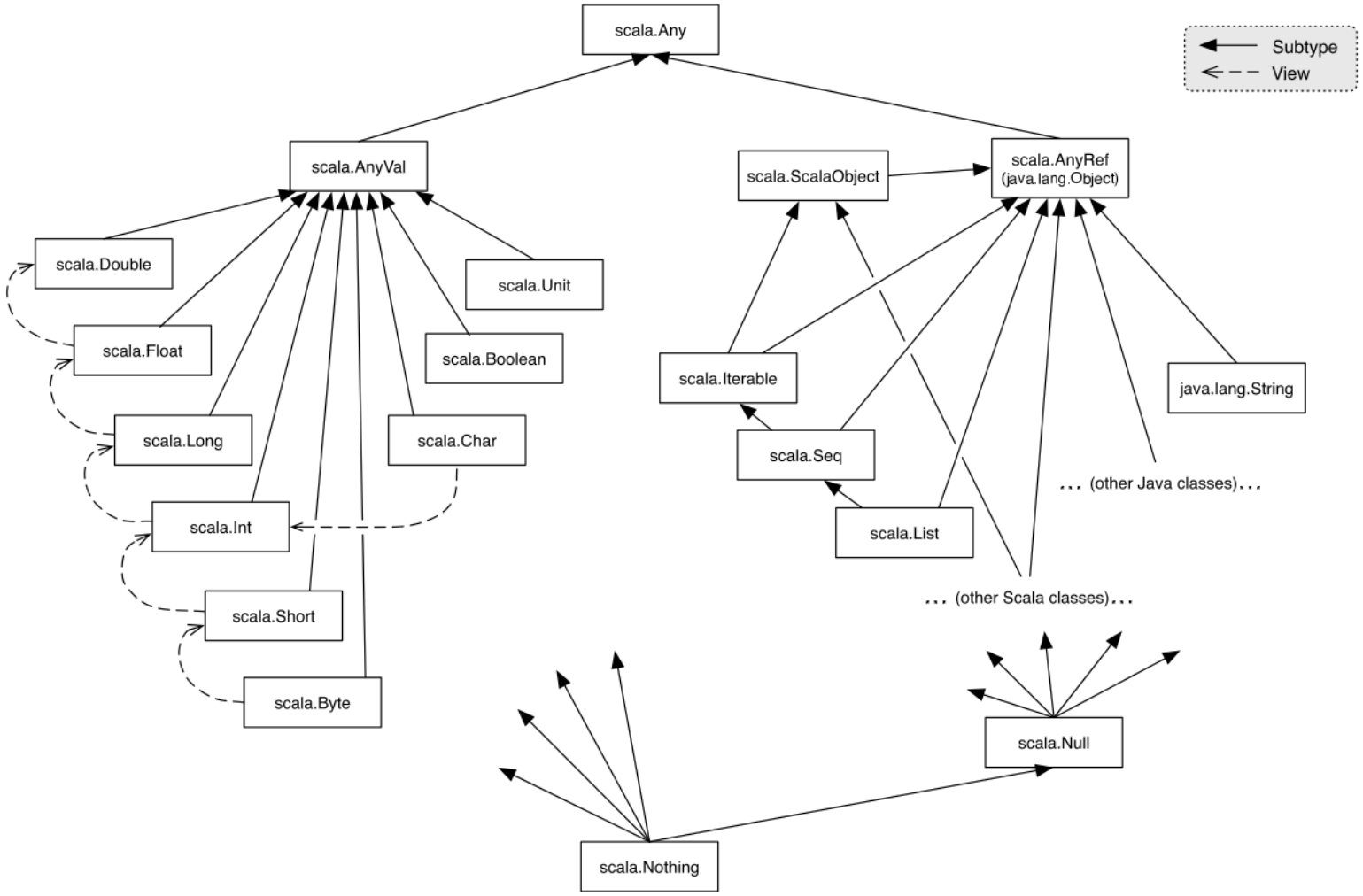
- Creado por Martin Odersky (2003)
- Tipado Estático e Inferencia de tipos
- Mezcla *OOP* y *FP*
- Concurrencia de Actores (akka) y con Futuros
- Bibliotecas muy maduras
- Open Source (BSD compilador, código fuente y principales librerías)
- Excelente Interoperabilidad con Java (Compila a Bytecode)
- Muy popular
  - Twitter
  - Linkedin
  - Sony Pictures
  - Siemens
  - Novell

Orientación a objetos y Programación Funcional

- Orientación a objetos:
  - Traits
  - Mixins
  - Case classes
  - Herencia *múltiple*
  - Tipado
  - Companion objects
  - Implícitos

- Programación Funcional:
  - **Inmutabilidad**
  - Lambdas
  - Opciones
  - Tipado
  - Currying
  - ADT
  - Pattern Matching
  - Monads / Monoides
  - Funciones de orden superior
  - Iteratee, Pipes (streams)
  - Lazy variables
  - Tail recursion
  - Funciones parciales

Tipado



- Todo es un objeto (Incluyendo valores numéricos y funciones)
- Jerarquía de tipos
- Invarianza, Contra Varianza, Covarianza
- Herencia de tipos
- Tipos dependientes (Roadmap Scala)

Actores (Akka)

- Scala usa **actores** como **modelo de concurrencia**
- Esta basado en los actores de Erlang
  - Asíncrona mediante envío de mensajes
  - Modelo de actores
  - Compartir datos **inmutables**
- Supervisión de actores frente a fallos
- Modular
- Clusters
- Soporta
  - Otros modelos de concurrencia: Futuros y Agentes
  - Otros sistemas: Apache Camel, ZeroMQ...

Enlaces e Información

Transparencias & Documentación

- Urls
  - Transparencias & Código https://github.com/pirita/TechFest2015
  - Scala interpreter http://www.simplyscala.com/ http://scalakata.com/
  - Scala js http://www.scala-js-fiddle.com/
  - Documentación http://www.scala-lang.org/documentation/

- Guías
  - http://danielwestheide.com/scala/neophytes.html
  - https://github.com/vhf/free-programming-books/blob/master/free-programming-books.md#scala
  - https://twitter.github.io/scala_school/

- Libros (Gratis)
  - Programming in scala http://www.artima.com/pins1ed/
  - A Scala tutorial for Java Programmers http://www.scala-lang.org/docu/files/ScalaTutorial.pdf
  - Scala by Example http://www.scala-lang.org/docu/files/ScalaByExample.pdf

- Cursos
  - Principios de programación funcional en Scala https://www.coursera.org/course/progfun
  - Principios fundamentales de la programación reactiva https://www.coursera.org/course/reactive

Instalar Scala

- Scala funciona sobre la JVM
  - Java 1.6 (Scala 2.11.X): http://goo.gl/QxAhfq
  - Scala: http://www.scala-lang.org/download/
  - http://www.scala-js.org/

- Editores / IDEs
  - Emacs (**¡Muy recomendable!**))
    - Ensime: https://github.com/ensime/ensime-server
    - sbt-mode: https://github.com/hvesalai/sbt-mode
  - Vim http://derekwyatt.org/2013/12/31/coding-scala-with-vim.html
  - SublimeText http://manuel.bernhardt.io/2013/09/20/scala-with-sublimetext/
  - Intellij (**¡Mejor IDE para Scala!**)
    - Editor para lenguajes de la JVM https://www.jetbrains.com/idea/
    - Plugin Scala (JVM y JS) https://plugins.jetbrains.com/plugin/?id=1347
  - Netbeans http://wiki.netbeans.org/Scala
  - Eclipse
    - Scala-ide: http://scala-ide.org/
  - Activator https://typesafe.com/get-started (**¡¡Muchos ejemplos!!**)

- Gestion dependencias
  - SBT http://www.scala-sbt.org/
  - Maven http://maven.apache.org/
  - Leiningen http://leiningen.org/
  - Gradle https://gradle.org/

Bibliotecas

- Scala puede usar todas las bibliotecas/Frameworks de Java
  - Spring, Google Guice, Hibernate...

- Y también existen bibliotecas y frameworks para Scala
  - Play https://www.playframework.com/ (Web framework)
  - Akka http://akka.io/ (Actors)
  - Spray / Akka.http http://spray.io/ (Api rest)
  - **Scalaz** https://github.com/scalaz/scalaz (Functional Programing ++)
  - Monocle https://github.com/julien-truffaut/Monocle (Lens for everything)
  - Lift http://liftweb.net/ (Web framework)
  - Scalatra http://www.scalatra.org/ (Web framework)
  - Scala check http://www.scalacheck.org/ (Property-based testing)
  - Specs2 http://etorreborre.github.io/specs2/ (Testing)
  - Slick http://slick.typesafe.com/ (Functional relational mapping)
  - Spark https://spark.apache.org/ (Big data y machine learning)
  - Shapeless https://github.com/milessabin/shapeless (Generic Programming)

- Y no olvidemos a Scala.JS http://www.scala-js.org/

ScalaMad: Scala Programming @ Madrid

- Meetup sobre scala http://www.meetup.com/Scala-Programming-Madrid/
  - Charlas y talleres periódicos sobre Scala
  - Accesible a todos los niveles
  - Se pueden proponer temas
  - Youtube https://www.youtube.com/channel/UCGskAkcw_kmOvGOzPsbIkkw

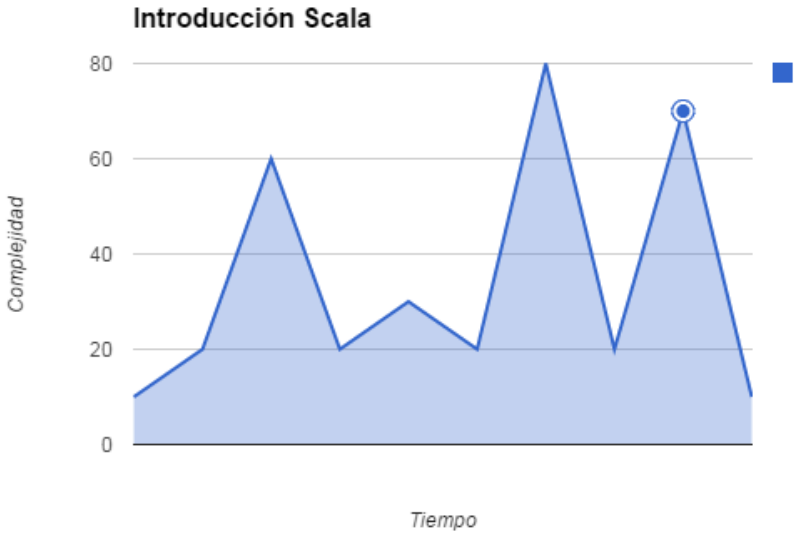
- Algunas de las próximas charlas son
  - **16 Febrero** Introduccion teoria de Categorias
  - **6 Marzo** Scala for dummies

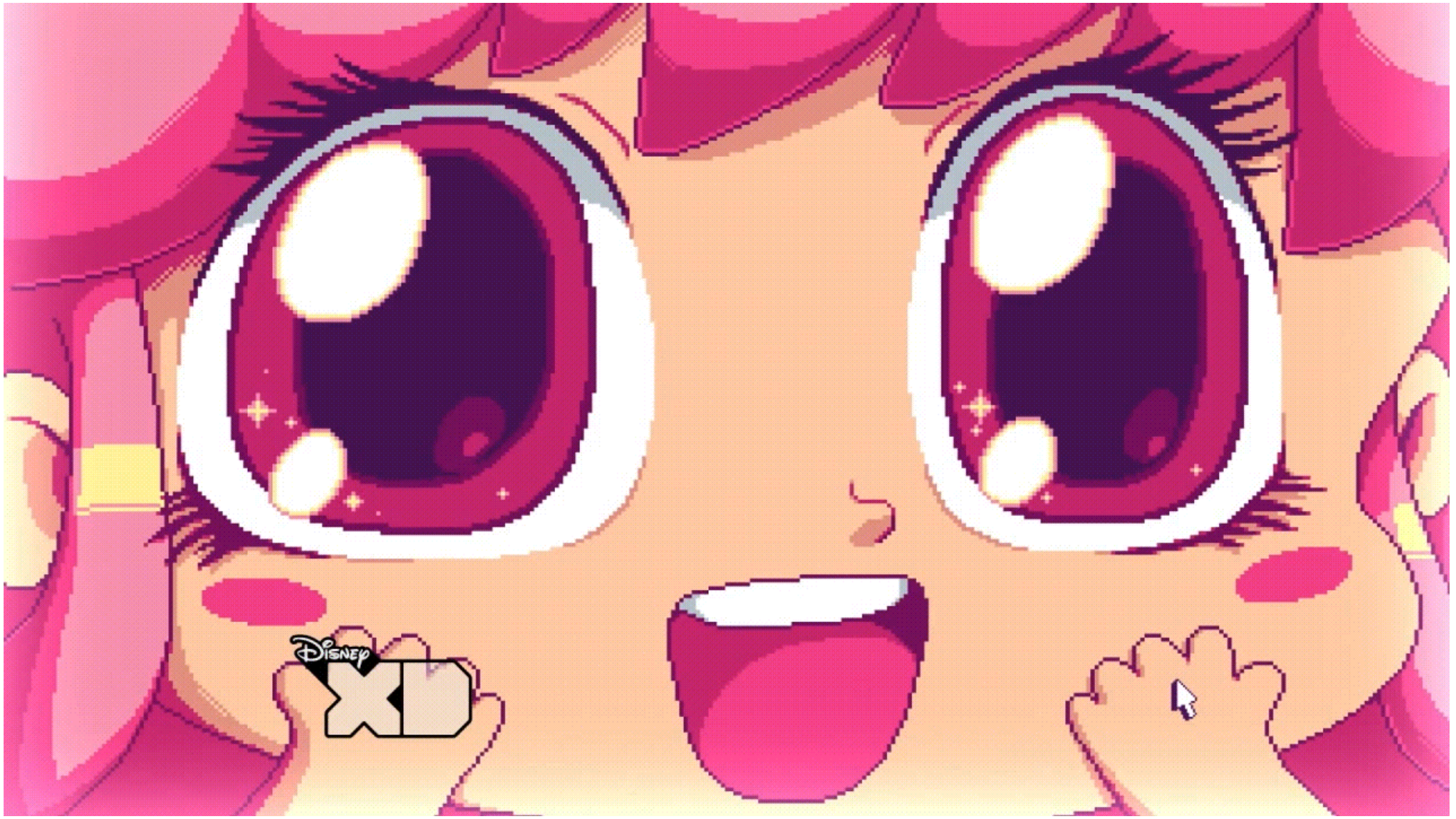
Referencias fotográficas

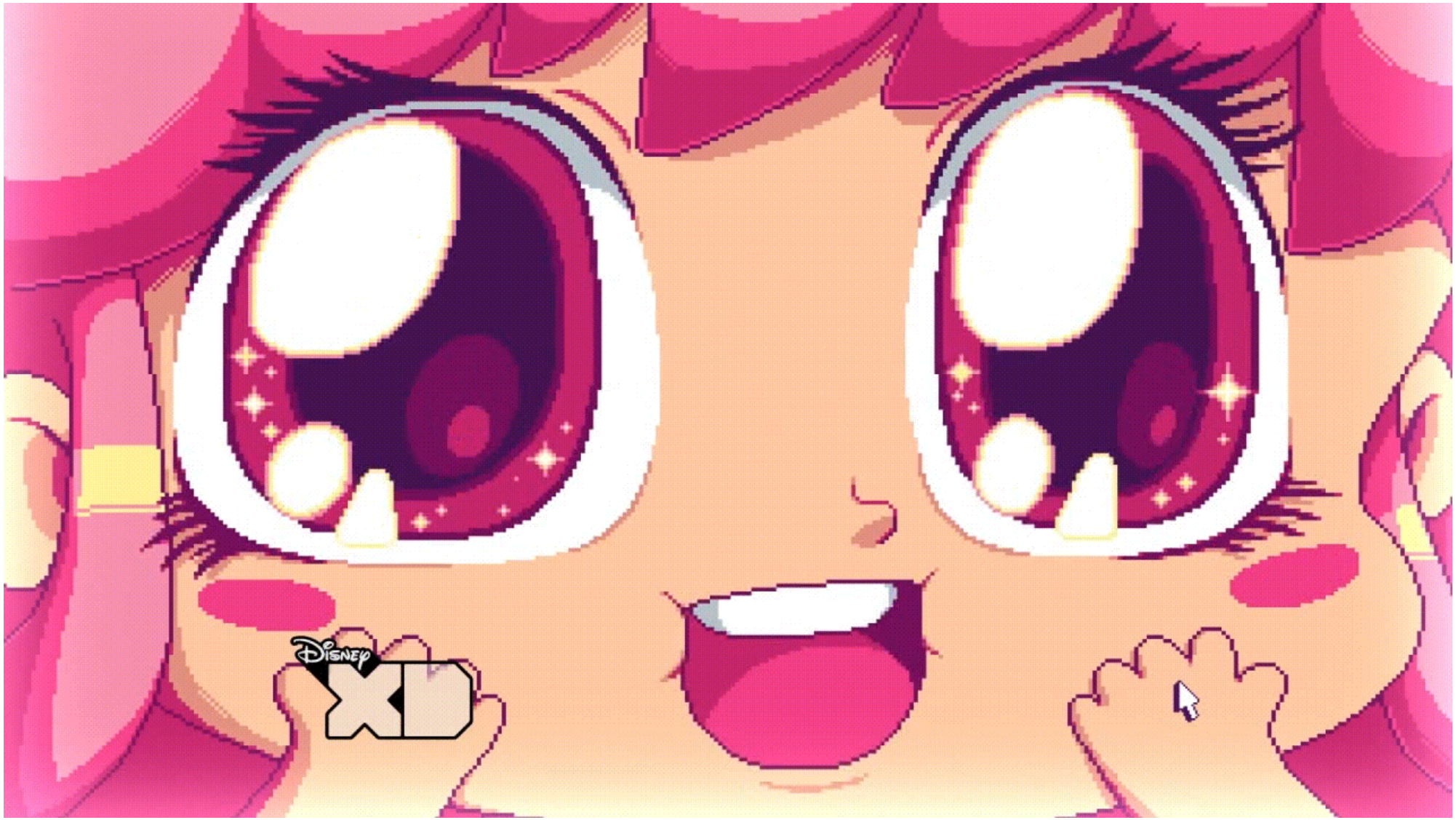
Scala Stair https://www.flickr.com/photos/gilles_dubochet/7327041044/
Scala Logo http://en.wikipedia.org/wiki/Scala_(programming_language)#mediaviewer/File:Scala_logo.png
Scala-chan http://tototoshi.github.io/slides/picture-show-introduction/intro/scalachan.png
OOP Lego http://www.legolegolego.com/wp-content/uploads/2008/09/soho-building.jpg
FP Happy http://nadirkeval.com/wp-content/uploads/2014/04/girlhappy.jpg
Ahkka http://en.wikipedia.org/wiki/%C3%81hkk%C3%A1#mediaviewer/File:Akka_mountain.jpg
Jerarquía de tipos http://www.scala-lang.org/old/sites/default/files/images/classhierarchy.img_assist_custom.png
Pyramids http://upload.wikimedia.org/wikipedia/commons/a/af/All_Gizah_Pyramids.jpg
Scala Meetup http://photos3.meetupstatic.com/photos/event/d/0/4/a/highres_250073322.jpeg
Haskell xkcd http://xkcd.com/1312/
Scala Lambda http://384uqqh5pka2ma24ild282mv.wpengine.netdna-cdn.com/wp-content/uploads/2014/01/lambda.jpg
Tipos http://www.the-house-of-literature.co.uk/Images/printing_letters.jpg
Coffe http://www.free-picture.net/albums/Drinks/coffee/cup-coffee-beens.jpg
Errors http://vignette2.wikia.nocookie.net/gravityfalls/images/f/fd/S1e20_ian_worrel_town_explosion.jpg/revision/latest?cb=20130830073203
Discworld http://s172.photobucket.com/user/frankwm1/media/DiscworldMap.jpg.html?t=1252649975
42 http://cienciaoficcion.com/wp-content/uploads/2014/06/tumblr_llqmm37xgQ1qiec6oo1_1280.jpg
Nada https://elasticgirl.files.wordpress.com/2009/02/lanada.jpg
Tail Recursion http://xkcd.com/1270/
Monad transformer https://twitter.com/deech/status/532707884304306176
Giffany https://www.youtube.com/watch?v=UfY2KqmKjB4

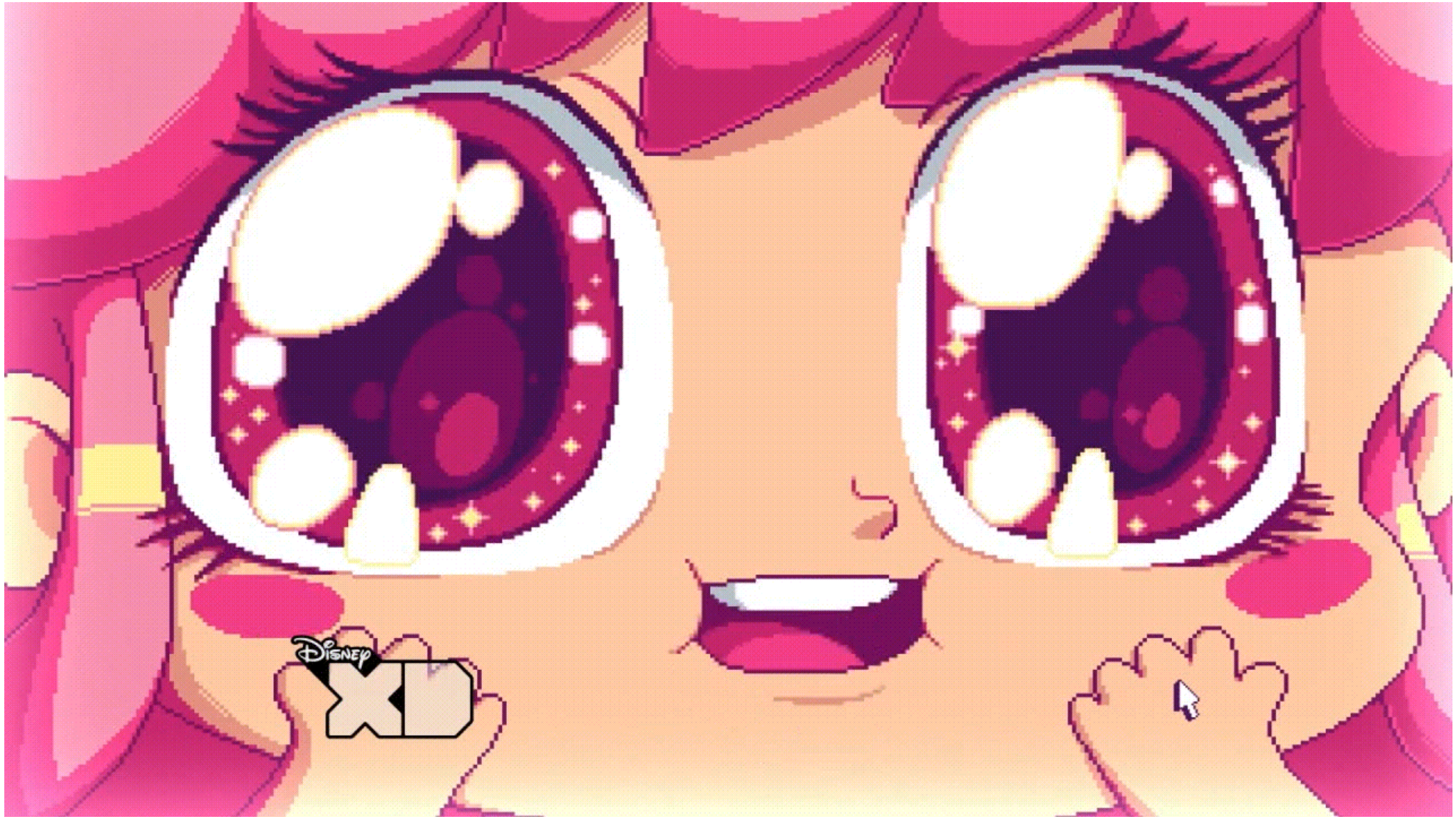
Antes de comenzar

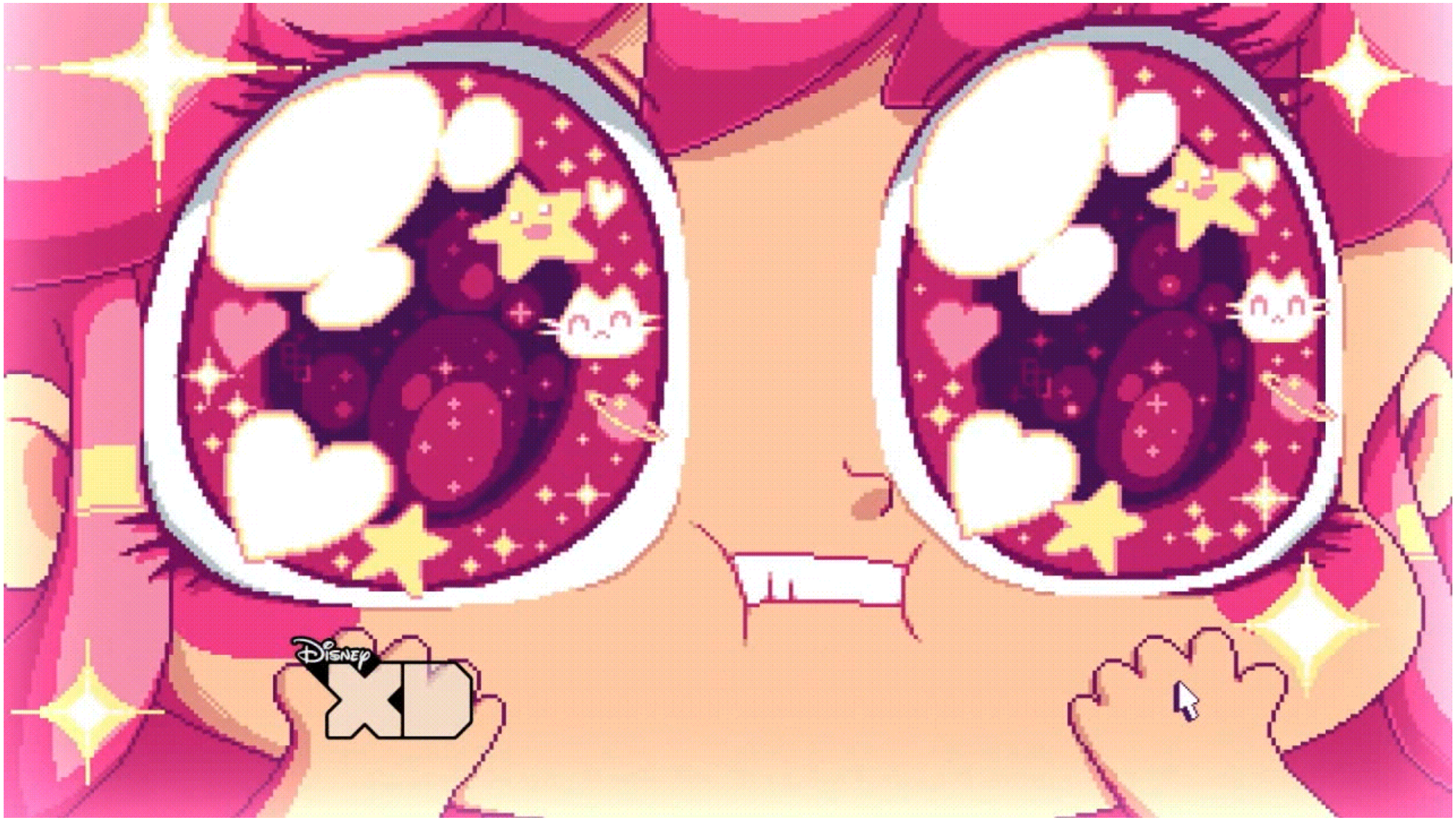
- *Object-oriented programming is an exceptionally bad idea which could only have originated in California* - **Edsger Dijkstra**
- Sometimes, the elegant implementation is just a function. Not a method. Not a class. Not a framework. Just a function - **John Carmack**
- *Though my tip though for the long term replacement of javac is Scala. I'm very impressed with it! I can honestly say if someone had shown me the Programming in Scala book by by Martin Odersky, Lex Spoon & Bill Venners back in 2003 I'd probably have never created Groovy* - **James Strachan**
- *Any sufficiently advanced technology is indistinguishable from magic* - **Arthur C. Clarke**
- *Don't Panic* - **The Hitchhiker's Guide to the Galaxy - Douglas Adams**




DISNEY
XD


Disney
XD


DISNEY
XD


DISNEY
XD

Inmutabilidad, valores y funciones

Inmutabilidad

- Todo valor en Scala es un **objeto**
  - Números son objetos
  - Funciones son objetos

- Se prima la **inmutabilidad (State is the root of all evil)**

- Existen variables formas de declarar variables/f
  - **var** mutable, se puede reasignar. Desaconsejada :(:(
  - **val** inmutable, evaluada sólo en la asignación
  - **def** inmutable, evaluada en cada llamada (usada para funciones normalmente)

- Se puede añadir el modificador **lazy** a un valor para postergar la evaluación

- En Scala solo es necesario definir el tipo de una variable si no puede ser inferido
  - val aux : Int = 14
  - val aux = 14 (*El compilador infiere que aux es un Int*)

Funciones

- Las funciones son objetos http://www.scala-lang.org/api/current/index.html#scala.Function2
- Buscamos la **pureza** y la **referencia transparencial**
- En Scala las funciones **siempre** devuelven un valor.
  - **Unit** define una función con *efectos de lado* (side effects)
  - El valor devuelto está en la **última línea** del cuerpo de una función
  - El valor de salida puede **inferirse**
- Las funciones tiene valor por defecto y se puede hacer referencia a los parámetros por su nombre

```
def printName(first:String, last:String = "Doe") = {
        println(first + " " + last)
        }

// Prints "John Smith"
printName(last = "Smith",first = "John")

// Prints "John Doe"
printName(first = "John")
```

Currying

- Toda función de N parámetros puede verse como una función de 1 parámetro que devuelve una funcion de N-1 parametros

```
def plus(n1: Int, n2: Int): Int = n1 + n2

def plus(n1: Int)(n2: Int): Int = n1 + n2

def plus: Int => Int => Int = n1 => n2 => n1 + n2
```

- Con esto podemos especificar un valor

```
def plus3 = plus(3)_

plus3(5) //returns 8!!
```

- Y obviamente pasar funciones como parámetros

```
def sumAndMultiply(n: Int)(m: Int)(f: Int => Int) : Int = f(n*m)

sumAndMultiply(2)(5)(plus3)
```

Polimorfismo y funciones anónimas

- Podemos tener funciones iguales con distintos parámetros

```
def concat(n1: String, n2: Int) = n1 + n2

def concat(n1: String, n2: String) = n1 + n2
```

- Se pueden crear funciones anónimas (lambda functions)

```
(x: Int) => x * x
val lambda: Int => Int = _ + 1
```

- _ es un placeholder. Indica una variable que no necesitamos su valor

Arity 0 & 1

- **Arity 0**: Las funciones con ningún parámetro puede definirse con o sin paréntesis. Y pueden llamarse con o si estos, pero si se definen sin paréntesis sólo podrán ser llamados sin estos (normalmente usados para funciones sin side effects).

```
def noSide = 12
def side() = println(34)

//noSide() Error
noSide

side()
side
```

- **Arity 1**: Los métodos pueden llamarse sin punto, ni parámetros. Esto facilita la creación de DSL:

```
List(1, 2, 3) :+ 24

List(1, 2, 3).:+(24)
```

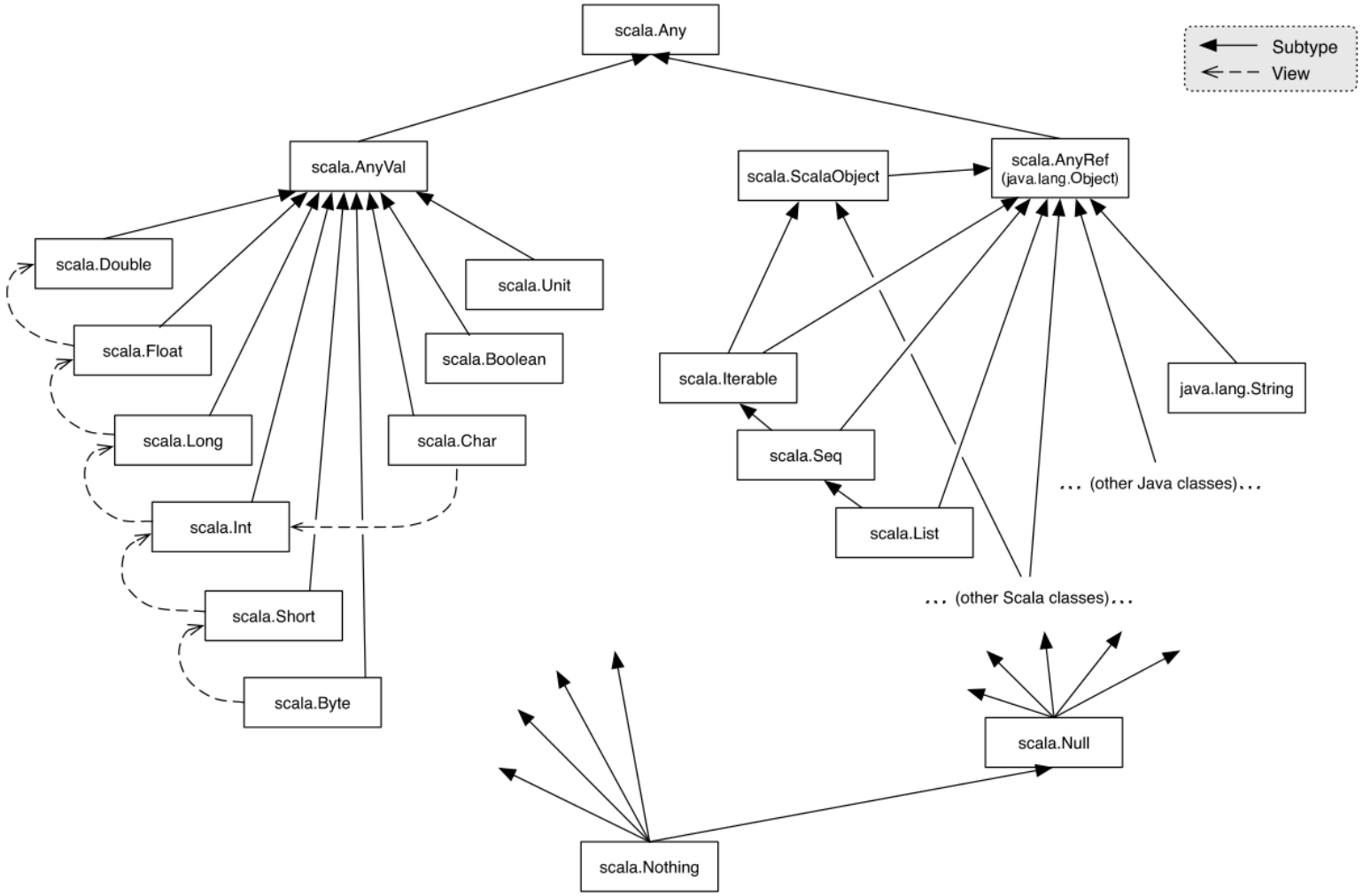
Recursión de cola y trampolines

- Scala permite crear funciones con *tail-recursion* y trampolines
  - **recursión de cola**: Funciones que finalizan con una llamada recursiva que no crea ninguna operación diferida . Se pueden optimizar con un proceso iterativo.

```
@tailrec
final def fibonacci(n: Long): Long = {
 if (n < 1) 1
 else fibonacci(n-1)
}
```

Tipado

Tipos



En Scala todo es un objeto.

- **Any**: Todo hereda de este tipo
- **AnyVal**: Para los tipos básicos de Java y Unit
- **Unit**: Define elementos con side effects
- **AnyRef**: Engloba lo que no es un AnyVal
- **Nothing**: Tipo inferior nada puede heredad de él

Varianza y herencia de tipos

En Scala los tipos pueden heredarse:

```
trait Persona

case class Futbolista(dinero: Long) extends Persona
case class Profesor(publicaciones: Int) extends Persona

def add2[T <: Persona](list: List[T])(element: T): List[T] = element :: list //Bound Superclass
def add2[T >: Persona](list: List[T])(element: T): List[T] = element :: list //Bound Subclass
def add2[T : Persona](list: List[T])(element: T): List[T] = element :: list //Bound Same class
```

Y se permite la varianza, contravarianza e invarianza

```
//Covariant
class Covariant[+A]
val cv1: Covariant[AnyRef] = new Covariant[String]
//val cv2: Covariant[String] = new Covariant[AnyRef] Error

//Contravariant
class ContraVariant[-A]
//val contra1: ContraVariant[AnyRef] = new ContraVariant[String] Error
val contra2: ContraVariant[String] = new ContraVariant[AnyRef]

//Invariant, Not related
class Invariant[A]
//val inva1: Invariant[AnyRef] = new Invariant[String] Error
//val inva2: Invariant[String] = new Invariant[AnyRef] Error
```

Estructuras de datos

Opciones 1

NULL, null, Nil, None, Unit, Nothing

Opciones 2

- **NULL** Tipo más bajo de los AnyRef (null es de tipo NULL)
- **Unit** Tipo que indica una función con efectos de lado
- **Nothing** Tipo más bajo en la jerarquía de tipos (métodos que no retornan...)

- **None** Se usa con las opciones para indicar que no existe un elemento
- **null** Es el null de Java. Se usa por retrocompatibilidad

- **Nil** Lista vacía

Opciones 3

I call it my billion-dollar mistake **Tony Hoare**

Scala intenta no trabajar con valores nulos. Se engloban en una opción

Option(posibleValorNulo)

Una opción puede ser:

- **Some[T]** Existe un valor T
- **None** No existe ningún valor

Listas, Vectores, Mapas, Tuplas, Streams

Todas las **estructuras de datos** en Scala son **inmutables** (existe collections.mutable, como Array :( )
Se pueden extraer los elementos mediante pattern matching

- **Listas** List(1, 2, 3, 4)
  - Eficientes salvo acceso aleatorio
  - La lista vacía es Nil
  - Concatenar elementos 1 :: List(1,2)
- **Vectores** Vector(1,2, 3,4)
  - Buen acceso
- **Tuplas** (1, 2, 3, 4)
  - Conjunto de N elementos heterogéneos
- **Mapas** Map(1->2)
  - Pueden verse como listas de tuplas de dos elementos
- **Stream** Stream(5, 7)
  - Listas potencialmente infinitas
  - Son totalmente lazy
  - No evalúan un elemento hasta no ser leídas
  - Eficientes

Existe colecciones **paralelizables**

Orientación a objetos

Orientación a objetos

- En scala se pueden definir clases de forma sencilla

```
class Person(age: Int, var  name: String, val surname: String) {
 val drive =  age >= 18
 //Everything is public by default
 def printName(): Unit = print(s"Name is $name")
 def sumAge(n: Int): Int = n + age
}
```

- También existen **Objects** que crean singleton
- Los objetos con el mismo nombre que una **clase** se llama **Companion Object**
- Se usan para muchas cosas, entre ellas poder crear extractores (unapply) o constructores(apply)

```
class Cube(size: Int){
 def area : Int = ???
}

object Cube{
 def apply(size: Int) = new Cube(size)
 def apply(size: Int, scale: Int) = new Cube(size*scale)
}
```

Case classes

- Si queremos usar las clases como contenedores de información inmutable podemos usar **case classes**
- Son clases con companion objects que
  - Todas sus variables son públicas e inmutables
  - Tienen extractores para pattern matching definidos
  - Ya está sobreescrito el método equals
  - Pueden ser serializadas
  - ...

```
case class Rectangle(a: Double, b : Double)
```

Traits

- Scala en vez de tener Interfaces tiene algo mucho más interesante: Traits
- Funcionan como una intefaz de Java pero permiten ser implementadas

```
trait Meow{
 val times: Int = 5
 def meow() = println("Meoow")
}
```

- Una clase u otro trait puede heredar de esta

```
class AnimalMeow(name: String) extends Meow //You can extends a trait
```

- Tambien podemos hacerlo en la instanciacion de un objeto (Creando un Mixin)

```
val animalSuperMeow = new Animal("Pig3") with Meow //Mixin!!
```

- Los traits además son tipos, lo que nos permite usarlos de forma muy interesantes

```
def onlyCats(cat: Animal with Meow) = ???
```

Herencia múltiple

- Scala no permite la herencia múltiple de Scala solo de traits
  - Una clase sólo permite puede heredar de una única clase
  - Una clase o trait pueden heredar de **múltiples traits**

- Para resolver el problema herencia múltiple Scala usa
  - El elemento más a la derecha prevale heredando todos los conflictos
  - Class linearization

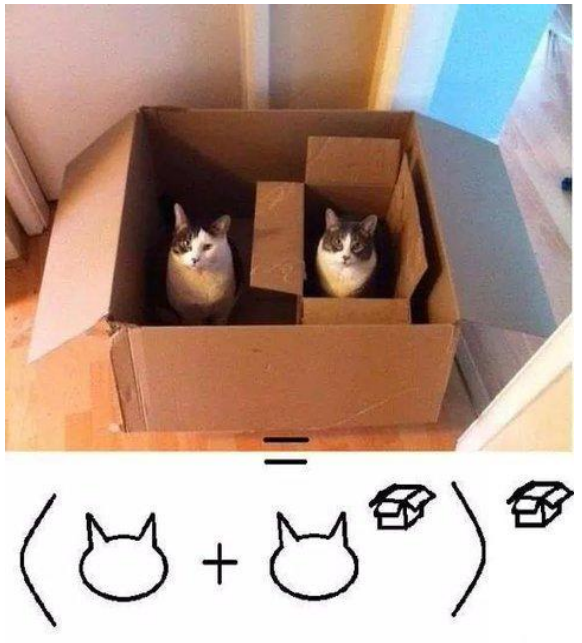
```
trait Talk{
 def hello = ""
}
trait TalkSpanish extends Talk{
 override def hello = "Hola"
 def two = "2"
}
trait TalkEnglish extends Talk{
 override def hello = "Hello"
 def jump = "jump"
}
case class Human(name: String) extends TalkEnglish with TalkSpanish

human.hello //Hola
```

Programación Funcional

Programación Funcional

La programación funcional se basa en

- ❏ Inmutabilidad (Stateless)
- ❏ Funciones de orden superior
- ❏ Recursividad (tail-recursion)
- ❏ Currying
- ❏ Closures
- ❏ Evaluación perezosa
- ❏ Sistemas de tipos (ADT)
- ❏ Funciones pura sin efectos de lado (Referencia transparencial)



Inmutabilidad

- Una operación sobre algo **inmutable** siempre devuelve una **nueva** instancia
- Algunas de las operaciones más habituales son:
  - map
  - filter
  - fold, foldLeft, reduce
  - flatten
  - flatMap (puede usarse **for** para componer varios flatMap)

```
def headsSum(list1: List[Int])(list2: List[Int])(list3: List[Int]) : Option[Int] = {
 for {
   h1 <- list1.headOption
   h2 <- list2.headOption
   h3 <- list3.headOption
 } yield h1 + h2 + h3
}
```

Pattern Matching

El pattern matching nos permite buscar patrones en nuestros datos

- Scala proporciona **extractores** (unapply)
- **Case classes** y estructuras de datos tienen creados sus propios extractores
- Se basa en la aplicación de funciones parciales

```
case class Human(name:String, age:Int)

val (jose, pepe, sara) = ( Human("Jose", 12), Human("Pepe", 13), Human("Sara", 21)
)
val aux = Option( (Human("Jose", 12), Human("Pepe", 13), Human("Sara", 21)) )

val valuePM = aux match {
 case Some((Human(name, edad), _ , _)) if edad < 20 => name
 case None => "Empty"
 case _ => "Others"
}
```

Gestión de Errores

- A diferencia con Java, no es obligatorio capturar las excepciones
- Se pueden gestionar
  - Try
  - Try monad
  - Either
  - Otros *contenedores*: Futuros...

Implícitos

La palabra reservada implicit nos permite

- Crear parámetros implícitos

```
class Fun(val fun: String)
def addFun(s: String)(implicit p: Fun) = p.fun + s
```

- Vistas

```
implicit def strToInt(x: String): Int = x.toInt
```

- Añadir métodos a clases (Pimp my library) - Monkey patching in local scope

```
object StringUtils{
 implicit class StringImprove(val s: String){
   def plusN(n: Int) : String = s"$s $n"
 }
}

import StringUtils._

"23" plusN 25
```

Implícitos

La palabra reservada implicit nos permite

- Crear parámetros implícitos

```
class Fun(val fun: String)
def addFun(s: String)(implicit p: Fun) = p.fun + s
```

- Vistas

```
implicit def strToInt(x: String): Int = x.toInt
```

- Añadir métodos a clases (Pimp my library) - Monkey patching in local scope

```
object StringUtils{
 implicit class StringImprove(val s: String){
   def plusN(n: Int) : String = s"$s $n"
 }
}

import StringUtils._

"23" plusN 25
```

Infinito y más allá

La cantidad de opciones que nos da la **programación funcional** es enorme

- Monads
- Funciones parciales
- ADT
- Tipos existenciales
- ….

Si queremos ir más allá

- **Shapeless** https://github.com/milessabin/shapeless
- **Scalaz** https://github.com/scalaz/scalaz

Gracias por venir

Nos vemos en el Meetup

https://github.com/pirita

ignacio.navarro.martin@gmail.com

@inavarromartin