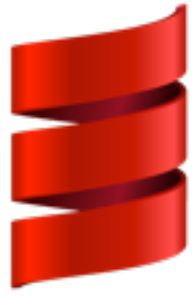
Introducción a Scala



Daniel Higuero (daniel.higuero@gmail.com)

@dhiguero

CIFF Centro Internacional de Formación Financiera
Universidad de Alcalá

NOVELTI

Agenda

- Introducción
- Instalación
- Conceptos básicos
- Ejercicios

¿Qué es Scala  ?

Scala

- Lenguaje de alto nivel sobre la JVM

JVM

MyClass.java

⇩ `javac`

MyClass.class

⇩ `java`

JVM Implementation

JVM

MyClass.scala

⇩ `scalac`

MyClass.class

⇩ `scala`

JVM Implementation

Características

- Lenguaje de alto nivel sobre la JVM
- Estáticamente tipado
- Compatibilidad bidireccional con librerías en Java
- Conversión no directa con:
  - Tipos nativos
  - Varargs
  - Collections
  - Parte funcional
  - Java 8 streams

Compatibilidad con Java

```
import java.time.LocalDateTime
...
def getNextDay() : LocalDateTime = {
  val currentTime : LocalDateTime = LocalDateTime.now()
  val nextDate = currentTime plusDays 1
  nextDate
}
```

¿Qué frameworks conoces en Scala?

Otros framework de Scala

Typesafe

akka

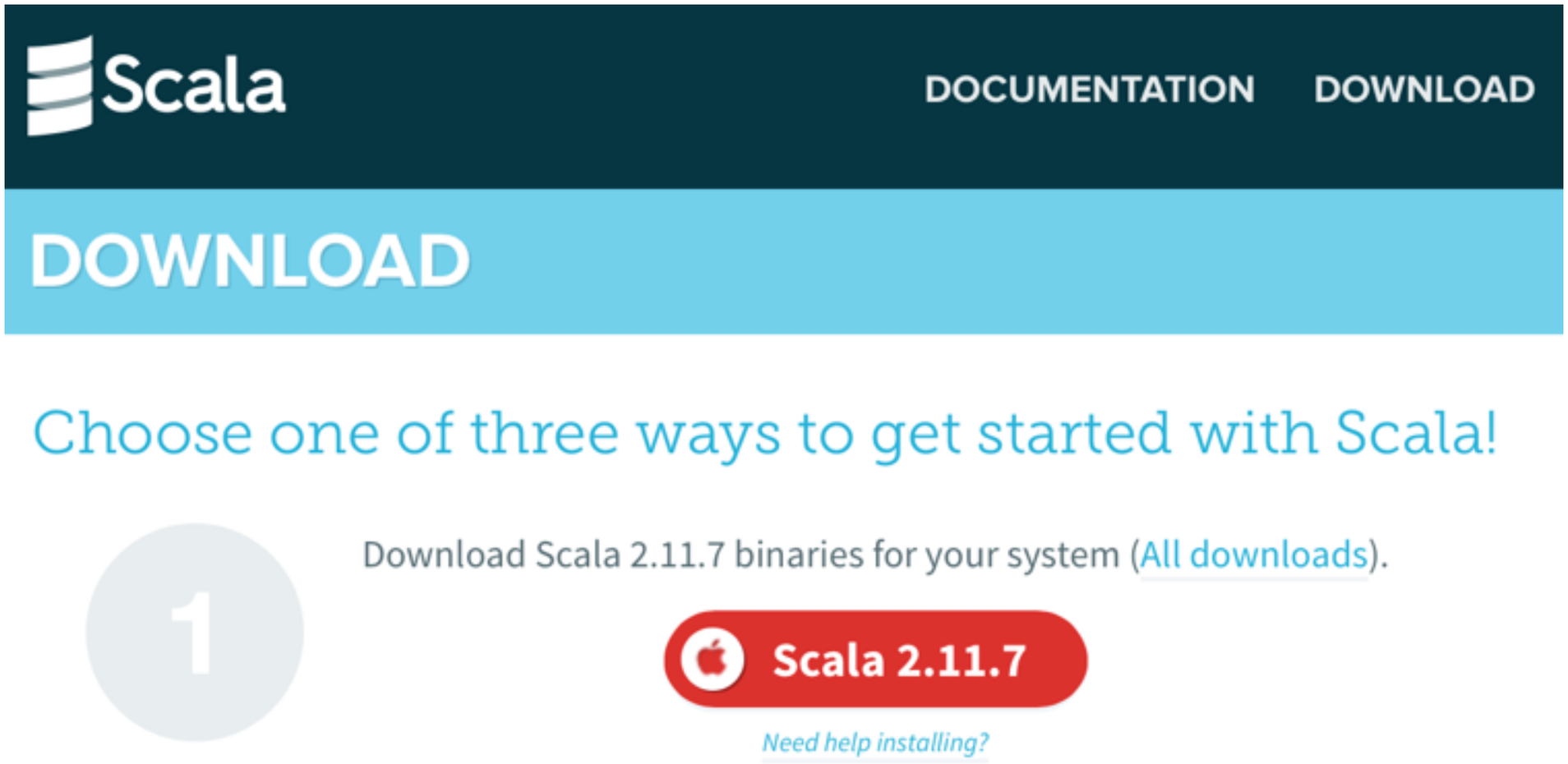
play

Spark

Instalación

Instalación

http://www.scala-lang.org/download/



Instalación

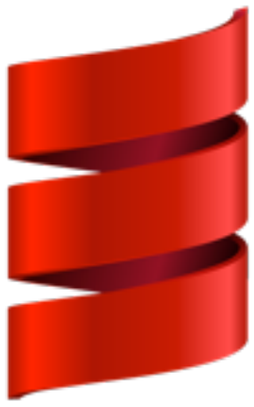
```
$ tar xvzf scala-2.11.7.tgz
$ cd scala-2.11.7
$ ./bin/scala
Welcome to Scala version 2.11.7 (Java HotSpot(TM) 64-
Bit Server VM, Java 1.8.0_60).
Type in expressions to have them evaluated.
Type :help for more information.

scala> print("Hello world!")
Hello world!
```

Basics

No ;

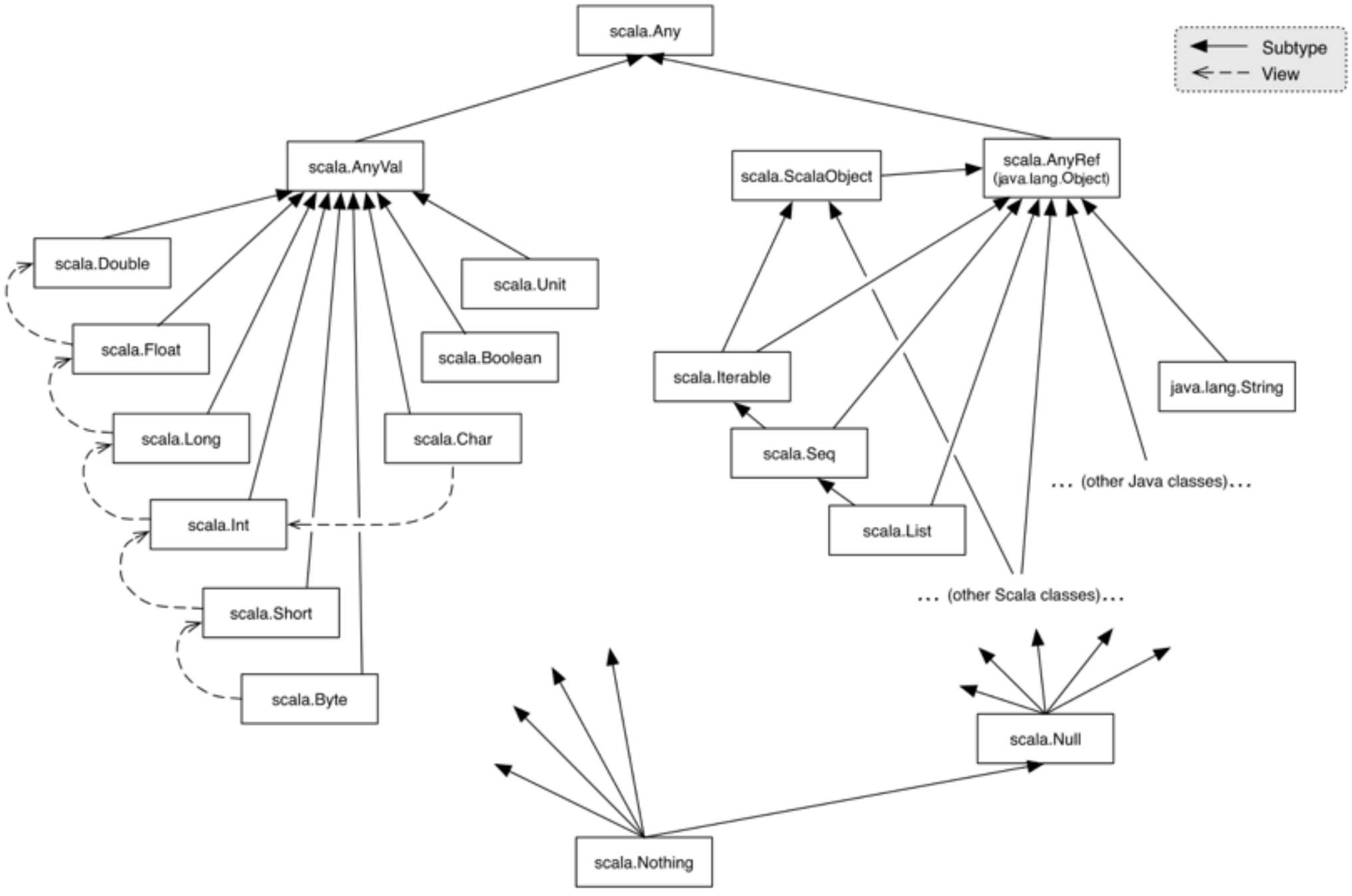
No NULL

Inmutable



Classes

Type hierarchy



Hello World

```
object HelloWorld {
  def main(args: Array[String]) {
    println("Hello world from main!")
  }
}
```

```
$ scalac HelloWorld.scala
$ scala HelloWorld
```

```
object HelloWorld extends App {
  println("Hello world from implicit main!")
}
```

Classes & Companion objects

```scala
object MyClass {
  val DefaultValue : Int = 3
  def main(args: Array[String]) : Unit = {
    val mc = new MyClass(1)
    println(mc.myMethod)
  }
}
class MyClass(value: Int){
  def myMethod() : Int = {
    MyClass.DefaultValue + this.value
  }
}
```

```scala
public class MyClass {
  public static void main(java.lang.String[])
  public static int staticAttribute();
  public int myMethod();
  public MyClass(int);
}
```

Declaración de variables

Declaración de variables

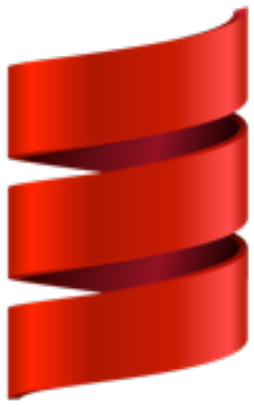
- Declaración de variables inmutables con val
- Declaración de variables mutables con var

```
var inmutableVariable : String = “InitialValue"

val mutableVariable : String = "InmutableValue"
```

Options

Always have options

```
scala> var c : Option[MyClass] = None
c: Option[MyClass] = None

scala> c.isDefined
res17: Boolean = false

scala> c = Some(new MyClass)
c: Option[MyClass] = Some(MyClass())

scala> c.isDefined
res18: Boolean = true
```



Funciones

Métodos y funciones

```
def functionName ([list of parameters]) : [return type] = {
   function body
   [return expr]
}
```

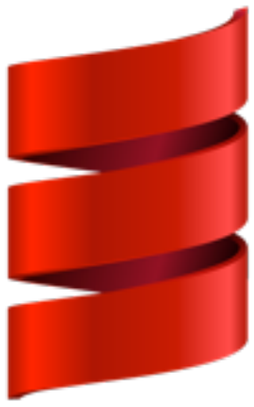
```
override def drop(n: Int): List[A] = {
  var these = this
  var count = n
  while (!these.isEmpty && count > 0) {
    these = these.tail
    count -= 1
  }
  these
}
```

Métodos y funciones

```
def print(x: Any) = Console.print(x)
def println() = Console.println()
def println(x: Any) = Console.println(x)
def printf(text: String, xs: Any*) = Console.print(
  text.format(xs: _*))
```

Funciones como parámetro

```
final override def foreach[U](f: A => U) : Unit = {
  var these = this
  while (!these.isEmpty) {
    f(these.head)
    these = these.tail
  }
}
```



Case classes

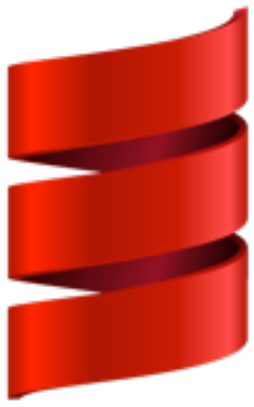
Case class

- Clase estándar
- Exporta automáticamente los parámetros del constructor
- Permite pattern matching

```
abstract class Expr
case class Var(name: String) extends Expr
case class Number(num: Double) extends Expr
case class UnOp(operator: String, arg: Expr) extends Expr
case class BinOp(operator: String,
  left: Expr, right: Expr) extends Expr
```

Pattern matching

```
def simplifyTop(expr: Expr): Expr = expr match {
  case UnOp("-", UnOp("-", e))  => e   // Double negation
  case BinOp("+", e, Number(0)) => e   // Adding zero
  case BinOp("*", e, Number(1)) => e   // Multiplying by one
  case _ => expr
}
```



Traits

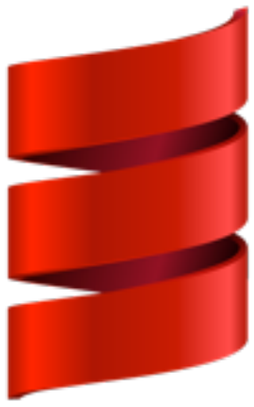
Traits

```
trait Similarity {
 def isSimilar(x: Any): Boolean
 def isNotSimilar(x: Any): Boolean = !isSimilar(x)
}
```

```
class Point(xc: Int, yc: Int) extends Similarity {
 var x: Int = xc
 var y: Int = yc
 def isSimilar(obj: Any) =
   obj.isInstanceOf[Point] &&
   obj.asInstanceOf[Point].x == x
}
```

Traits

```
object StringIteratorTest {
  def main(args: Array[String]) {
    class Iter extends StringIterator(args(0))
      with RichIterator
    val iter = new Iter
    iter foreach println
  }
}
```



Generics

Generics

```
1 class Stack[T] {
2   var elems: List[T] = Nil
3   def push(x: T) { elems = x :: elems }
4   def top: T = elems.head
5   def pop() { elems = elems.tail }
6 }
```

For

For

- Diferencias semánticas con Java

```
for(x <- c1; y <- c2; z <-c3) {...}
```

```
c1.foreach(x => c2.foreach(y => c3.foreach(z => {...})))
```

```
for(x <- c1; y <- c2; z <- c3) yield {...}
```

```
c1.flatMap(x => c2.flatMap(y => c3.map(z => {...})))
```

For-yield

```
object ComprehensionTest1 extends Application {
  def even(from: Int, to: Int): List[Int] =
    for (i <- List.range(from, to) if i % 2 == 0) yield i
  Console.println(even(0, 20))
}
```

Try-catch

Try-catch

```
try {
    // ...
  } catch {
    case ioe: IOException => ...
    case e: Exception => ...
  } finally {
...
}
```

Try

```
def divide: Try[Int] = {
  val dividend = Try(Console.readLine("Int dividend:\n").toInt)
  val divisor = Try(Console.readLine("Int divisor:\n").toInt)
  val problem = dividend.flatMap(x => divisor.map(y => x/y))
  problem match {
    case Success(v) =>
      println(“Result: " + dividend.get + "/"
        + divisor.get +" is: " + v)
      Success(v)
    case Failure(e) =>
      println(“Division error. Try again!")
      println("Info from the exception: " + e.getMessage)
      divide
  }
}
```

Programación funcional

Map

```
final def flatMap[B](f: (A) => GenTraversableOnce[B]): List[B]

final def map[B](f: (A) => B): List[B]
```

```
scala> val myList = List(1,2,3,4,5,6)
myList: List[Int] = List(1, 2, 3, 4, 5, 6)

scala> myList.map(value => value * 2)
res1: List[Int] = List(2, 4, 6, 8, 10, 12)
```

Reduce

```
def reduce[A1 >: A](op: (A1, A1) => A1): A1
```

```
scala> val myList = List(1,2,3,4,5,6)
myList: List[Int] = List(1, 2, 3, 4, 5, 6)

scala> myList.reduce((a,b) => a + b)
res3: Int = 21
```

Filter

```
def filter(p: (A) => Boolean): List[A]
```

```
scala> val myList = List(1,2,3,4,5,6)
myList: List[Int] = List(1, 2, 3, 4, 5, 6)

scala> myList.filter(_ > 3)
res2: List[Int] = List(4, 5, 6)
```

Fold

```
def fold[A1 >: A](z: A1)(op: (A1, A1) => A1): A1
```

```
scala> myList.fold(0)((acc, cur) => {acc + cur})
res16: Int = 21
```

FoldLeft

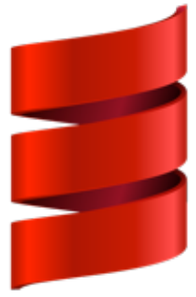
```
def foldLeft[B](z: B)(f: (B, A) => B): B
```

```
scala> myList.foldLeft(Map.empty[Int, Int])(
(acc, cur) => {acc + (cur -> cur *2)}
)
res5: scala.collection.immutable.Map[Int,Int] =
 Map(5 -> 10, 1 -> 2, 6 -> 12,
 2 -> 4, 3 -> 6, 4 -> 8)
```

Otros elementos

- Implicits
- Modos de herencia
- Rendimiento de las colecciones
- Sbt vs Maven
- Evolución al futuro

Introducción a Scala



Daniel Higuero (daniel.higuero@gmail.com)

@dhiguero

CIFF Centro Internacional de Formación Financiera
Universidad de Alcalá

NOVELTI