UNIVERSIDAD COMPLUTENSE DE MADRID

ntic master
revolucionamos la comunicación

Manual de referencia de Scala

Paquetes - Packages

```
// wildcard to import everything from the collection library
import scala.collection._

// specific import for the Vector class
import scala.collection.Vector

//  import multiple classes
import scala.collection.{Vector, Sequence}

//declare a package
package pkgname
```

Operadores

```
// infix notation where op can be +, -, *, /, %
x op y is x.op(y)
// postfix notation
x op is x.op( )
// compares two objects (calls equals method)
x == y
// There is no ++, -- in Scala
```

Symbols

```
;       // optional end of line
->      // returns a two element tuple for a key, value pair
<-      // assign to in a for comprehension
=>      // used in function literals to separate arguments from the
        // function body

::      // cons operator
//      // single-line comment
/*...*/ // multiline comment
```

Operadores relacionales

```
|| // or
&& // and
!  // not
```

Comparadores

```
==    // equals
<     // less than
>     // greater than
<=    // less than or equal to
>=    // greater than or equal to
```

Expresiones Lambda

```
// anonymous function to square x
(x:Int) => x * x

// anonymous function using bound infix method, multiplies 1,2,3,4,5
// by 2
(1 to 5).map(2 * _)

val x = (1 to 5).map {
      2 * _       // multiplies each value by 2
      println(x)  // print x
      x           // returns x (Vector (2, 4, 6, 8, 10)
}

// only returns even numbers; creates vector (2, 4, 6, 8, 10)
(1 to 10) filter { _ % 2==0}

// multiplies all even values by 2; creates vector (4, 8, 12, 16,
// 20)
(1 to 10) filter { _ % 2==0} map { _ * 2}
```

Variables

```
// creates a mutable variable
var

// creates a mutable integer variable
var myVar:Int

// creates an immutable variable
val

// creates an immutable String variable ( or val myVal = “Monday”)
val myVal:String
```

Funciones

```
// define function f, with parameter x, an integer; no return type
// specified
def f(x:Int) = {...}

// define function times3 that evaluates parameter x multiplied by 3
def times3(x:Int) = 3 * x

// anonymous function call
val f = (x:Int) => 3 * x

//function returns unit since it has no = sign; prints Hello world x
// times
def message(x:Int){
      for(i<-(1 to x)) println(“Hello World”)
}

//use a default value for intro
def message(x:String, intro:String =”Dear”) {
      println(intro + “,” + x)
}

// call by value
def f(x: R)

// call by name (reference)
def f(x: => R)
```

```
//return type required for recursive functions
def sum(xs:Int*):Int = { // * indicates variable number of args
      var r = 0
      for(x <- xs) r += x
      r
}

//same results as above but with functional style
def sum(xs:Int*):Int = if(xs.length == 0) 0 else xs.head +
sum(xs.tail : _*)
```

Estructura de datos

```
// tuple literal
(1,2,3)

// tuple unpacking via pattern matching
var(a,b,c) = (1,2,3)

// creates an immutable list called xs
var xs = List(1,2,3)

// access the element at location zero, indexing
xs(0)

// adds 4 to the front of the list creating List(4,3,2,1)
4::List(3,2,1)

// range of numbers from 1 to 10 inclusive
1 to 10

// range of numbers from 1 to 9, excludes upper bound
1 until 10

// creates a List of values excluding the upper bounds
val list = List.range(1,11)
```

Sentencias de decisión

```
If(expr that evaluates to true/false) println("true")
else println("false")
```

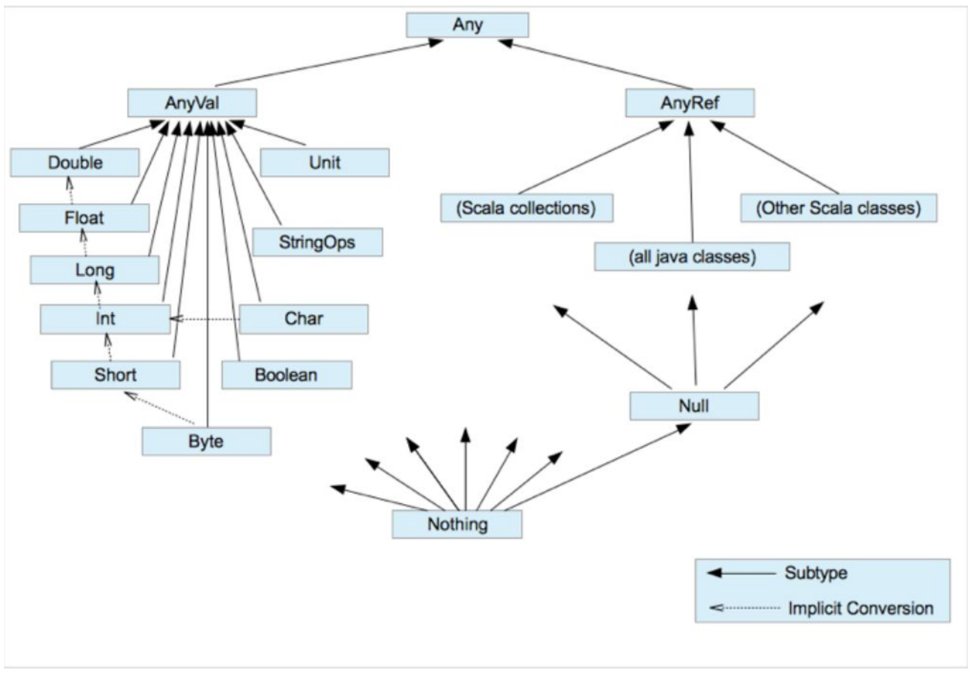
Bucles

```
// execute a body of code while the expr is true
while(expr) {...}

// execute a body of code at least once, continue while expr is true
do{...} while(expr)

// print all values of x from the List called myList
for(x <- myList) println(x)

// for comprehension
for(x <- myList if x%2 == 0) yield x*10
for(x <- 1 to 10) {...}
```

Pattern Matching

```
// assign value for x after evaluates the correct matching of r
val x = r match {
      case '0' => ... //match a value
      //add a guard to the match criteria
      case ch if someProperty(ch) => ...
      case e: Employee => … //match runtime type
      case (x,y) => … //destructures pairs
      case Some(v) => … //case classes have extractors
      //infix notation for extractors yielding a pair
      case 0 :: tail => ...
      case _ => … //default case
}
```

Tipos de Datos

Byte Short Int Long Float Double Boolean String Char Unit Null Nothing Any AnyRef	

Any
AnyVal
AnyRef
Double
Unit
Float
StringOps
Long
Int
Char
Short
Boolean
Byte
(Scala collections)
(Other Scala classes)
(all java classes)
Null
Nothing
Subtype
Implicit Conversion