Nuevas Tecnologías de la Programación

Tema 3: valores, variables y tipos

Curso 2017-18

Índice

Uso de REPL

Esta herramienta permite evaluar y ejecutar código **Scala** línea a línea, con información de ayuda tras cada línea. Todas las variables están disponibles durante la sesión de trabajo.

Dispone de un sistema de ayuda, que se inicia mediante el comando **:help**

```
scala> :help
All commands can be abbreviated, e.g., :he instead of :help.
:edit <id>|<line>  edit history
:help [command]    print this summary or command-specific help
:history [num]     show the history (optional num is commands to show)
.....................................................
```

Uso de REPL

Para practicar con esta herramienta comenzamos ejecutando el típico **Hola, mundo**. Para ello basta con teclear la orden correspondiente tras el cursor de **scala**:

```
scala> println("Hola, mundo")
Hola, mundo
```

Se aprecia que debajo de la sentencia se muestra el resultado producido. Se observa que la función **println** es accesible de forma directa.

Uso de REPL

La herramienta **REPL** permite el uso de las flechas arriba y abajo, para poder recuperar los comandos previamente tecleados. La historia de cada sesión se mantiene entre sesiones.

Se prueban ahora algunas operaciones aritméticas sencillas:

```
scala> 37*123
res0: Int = 4551
```

Si la expresión evaluada genera un valor, entonces se crea una variable que almacena el resultado producido (son variables que no pueden modificar su valor), siempre con nombre formado por el prefijo **res**; el sufijo va actualizándose a medida que se ejecutan más comandos.

Uso de REPL

Los resultados de operaciones pevias pueden usarse en operaciones posteriores:

```
scala> res0*1345
res1: Int = 6121095
```

Como se aprecia, el nuevo resultado se almacena en la variable **res1**.

Índice

Conceptos básicos

Conviene definir los conceptos básicos antes de continuar con la exposición del tema:

- literal: dato que aparece directamente en el código fuente, como 5, el carácter 'a' o la cadena "Hola, Pepe"
- valor: unidad de almacenamiento para un tipo determinado e inmutable (constante). Es decir, una vez asignado el dato no será posible cambiar su valor
- variable: unidad de almacenamiento para un tipo determinado, pero mutable. De esta forma, es posible reasignar su valor tantas veces como se desee

Conceptos básicos

- tipo: clase de dato con el que se trabaja; definición o clasificación de los datos. Todos los datos manejados en **Scala** tienen un tipo asociado; además todos los tipos de definen como clases, ofreciendo métodos que trabajan sobre ellos. Es decir, todo es un objeto

```
scala> 3.getClass().getName()
res1: String = int
```

Al pulsar el tabulador tras haber escrito el objeto (3) y el punto muestra todos los posibles métodos a usar sobre él.

Conceptos básicos

La diferencia entre valores y variables se pone de manifiesto de forma explícita en el código mediante las palabras reservadas **val** y **var**. Veamos algunos ejemplos:

```
scala> val x : Int = 5
x: Int = 5
```

Esta sentencia crea un **valor**, al que se le asigna el literal 5. Al ser **val** no es posible modificar su valor:

```
scala> x=37
<console>:8: error: reassignment to val
       x=37
         ^
```

Conceptos básicos

Una vez declarado el valor x, podemos usarlo en cualquier expresión o sentencia:

```
scala> x
res2: Int = 5

scala> x*23
res3: Int = 115

scala> x/3
res4: Int = 1
```

Conceptos básicos

El uso de variables precisa la declaración mediante var:

```
scala> x
scala> var y : Double = 18.7
y: Double = 18.7
```

Y pueden usarse sin ninguna limitación:

```
scala> y=y*x/37
y: Double = 2.527027027027027
```

Índice

Valores

Como hemos indicado, se trata de unidades de almacenamiento para un tipo específico de datos, que no pueden cambiar su valor. Al basarse la programación funcional en el principio de estado inmutable, deberíamos usar valores en lugar de variables.

La sintaxis de declaración de valores es:

```
val <identificador> [ : <tipo> ] = <dato>
```

En muchos casos no será preciso indicar el tipo de dato, ya que **Scala** puede inferirlo mediante el análisis del código.

Valores

Veamos algunos ejemplos:

```
scala> val x : Int = 40
x: Int = 40

scala> val importante : String = "val - similar a constante"
importante: String = val - similar a constante

scala> val z=3.45
z: Double = 3.45

scala> val caractera = 'a'
caractera: Char = a
```

Valores

La inferencia de tipos ayuda a escribir menos código, pero incluir el tipo puede ayudar a que el código sea más legible. Y, obviamente, debe coincidir con el tipo del literal usado en la asignación:

```
scala> val x : Int = 13.8
<console>:7: error: type mismatch;
 found   : Double(13.8)
 required: Int
       val x : Int = 13.8
                     ^
```

Valores

Hay que aclarar el concepto de inmutabilidad, para evitar confusiones. Como ejemplo, consideramos la colección de tipo **Array**, que se define como mutable. En este caso, el código mostrado a continuación no presenta problema alguno:

```
scala> val array : Array[Int] = new Array(10)
array: Array[Int] = Array(0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

scala> array(0) = 23

scala> array(1) = 4
```

Es decir, no hay problema a la hora de almacenar valores nuevos en la colección.

Valores

Lo que sí resultaría problemático es intentar crear un nueva colección sobre la misma variable:

```
scala> array = new Array(5)
<console>:8: error: reassignment to val
       array = new Array(5)
             ^
```

De esta forma, igual que las constantes, los valores deben inicializarse en el momento de su declaración.

Valores

La palabra reservada **val** también puede aparecer asociada a los datos miembro de una clase.

```
scala> class NumeroComplejo(val x:Double, val y:Double)
defined class NumeroComplejo

scala> val n1=new NumeroComplejo(1.3, 2.5)
n1: NumeroComplejo = NumeroComplejo@12028586

scala> n1.x
res7: Double = 1.3

scala> n1.y
res8: Double = 2.5
```

Valores

El uso de **val** con los datos miembro hace que su valor no pueda modificarse.

```
scala> n1.x=23.5
<console>:9: error: reassignment to val
       n1.x=23.5
           ^
```

Tampoco sería posible usar n1 para referenciar otro nuevo objeto (al ser también de tipo val).

Índice

Variables

Se trata ahora de unidades de almacenamiento mutables, asociadas a un determinado tipo. Coincide con el concepto de variable de cualquier lenguaje de programación.

En **Scala** se intenta favorecer el uso de valores frente a variables, como forma de adhesión a los principios básicos de programación funcional.

Valores

Para la declaración de variables se usa la palabra reservada **var**. La sintaxis de declaración es la siguiente:

```
var <identificador> [ : <tipo> ] = <dato>
```

Al igual que en el caso de uso de val puede omitirse el tipo de dato. Y al tratarse de variables normales, podemos modificar su valor tantas veces como sea necesario.

```
scala> var radio = 12.3
radio: Double = 12.3

scala> radio = radio*2
radio: Double = 24.6
```

Valores

No es posible asignar un literal que no se corresponda con el tipo con el que se hizo la creación de la variable:

```
scala> radio="a"
<console>:8: error: type mismatch;
 found   : String("a")
 required: Double
       radio="a"
             ^
```

Sí existe la conversión implícita de tipos compatibles (cuando no haya pérdida de información). También hay que tener en cuenta que en **Scala** no hay tipos de datos primitivos: se sigue el principio de orientación a objetos de forma completa.

Índice

Identificadores

En **Scala** los identificadores se forman de acuerdo a las siguientes reglas (algunas de ellas):

- una letra seguida de 0 o más letras y dígitos
- una letra seguida de 0 o mas letras o dígitos, un subrayado y después uno o más caracteres (letras, números o caracteres de operadores)
- uno o más caracteres de operadores
- uno o más de cualquier tipo de carácter, excepto la comilla, siempre y cuando estén encerrados entre comillas.

Identificadores

Ejemplos:

```
scala> val pi=3.1415
pi: Double = 3.1415

scala> val $=3.1415
$: Double = 3.1415

scala> val a_z="caracteres minuscula"
a_z: String = caracteres minuscula
```

Identificadores

Ejemplos:

```
scala> val 3pi=3*pi
<console>:1: error: Invalid literal number
       val 3pi=3*pi
           ^

scala> val x.y=37
<console>:8: error: value y is not a member of Int
       val x.y=37
             ^
```

NOTA: carácter PI (alt+p) válido; también el uso de punto entre x e y, si se usa comilla invertida (tecla a la derecha de p).

Identificadores

En **Scala** se usa el mismo estilo que en Java a la hora de usar los identificadores:

- se reserva mayúscula para clases y constantes
- al juntar palabras, la segunda se une a la primera, poniendo en mayúscula su primera letra: *buscarMinimo*
- nombres de variables, datos y métodos miembro, argumentos, etc... siempre en minúscula

Índice

Tipos

En **Scala** hay tipos numéricos y no numéricos. No ha datos de tipo primitivo. Los tipos numéricos básicos son:

- Byte: valores posibles -127 a 128
- Short: de -32768 a 32767
- Int: de -2^{31} hasta $2^{31} - 1$
- Long: de -2^{63} a $2^{63} - 1$
- Float: de $-\infty$ a ∞
- Double: de $-\infty$ a ∞

Tipos

Hay conversión automática de ensanchamiento: de tipos con menos capacidad a otros con más capacidad.

```
scala> val x: Short = 7
x: Short = 7

scala> val b : Byte = 10
b: Byte = 10

scala> val s : Short = b
s: Short = 10

scala> val d : Double = s
d: Double = 10.0
```

Tipos

La conversión de estrechamiento no puede hacerse de forma directa:

```
scala> val l : Long = 20
l: Long = 20

scala> val i : Int = l
<console>:8: error: type mismatch;
 found   : Long
 required: Int
       val i : Int = l
                     ^
```

Pero sí es posible la conversión explícita:

```
scala> val i : Int = l.toInt
i: Int = 20
```

Tipos

La asignación puede hacerse indicando el tipo de datos deseado:

```
scala> val entero=5
entero: Int = 5

scala> val hexadecimal=0xffff00
hexadecimal: Int = 16776960

scala> val enteroLargo=100l
enteroLargo: Long = 100
```

Índice

Cadenas de caracteres

Scala se basa en el uso de la clase **String** de Java:

```
scala> val saludo="Hola, Pepe"
saludo: String = Hola, Pepe

scala> val despedida="Hasta luego, \n Lucas"
despedida: String =
Hasta luego,
 Lucas
```

Cadenas de caracteres

Es posible el uso de operadores aritméticos con cadenas de caracteres:

```
scala> val saludo="Hola, "+"Pepe"
saludo: String = Hola, Pepe

scala> val iguales=(saludo == "Hola, Pepe")
iguales: Boolean = true

scala> val tarareo="ta"*5
tarareo: String = tatatatata
```

Cadenas de caracteres

Podemos manejar cadenas con varias líneas, que se especifican usando la triple comilla:

```
scala> val comentario = """Esto es un comentario dividido
     | entre varias lineas, hasta la triple comilla
     | cerrando"""
comentario: String =
Esto es un comentario dividido
entre varias lineas, hasta la triple comilla
cerrando
```

Cadenas de caracteres

Un mecanismo disponible para las cadenas es la interpolación, usando el carácter **s** justo antes de la primera comilla de la cadena. Gracias a la interpolación es posible usar el carácter $ para referenciar datos externos a la cadena:

```
scala> val aproximacionPi=355/113f
aproximacionPi: Float = 3.141593

scala> println(s"Valor de pi aprox: $aproximacionPi")
Valor de pi aprox: 3.141593
```

Cadenas de caracteres

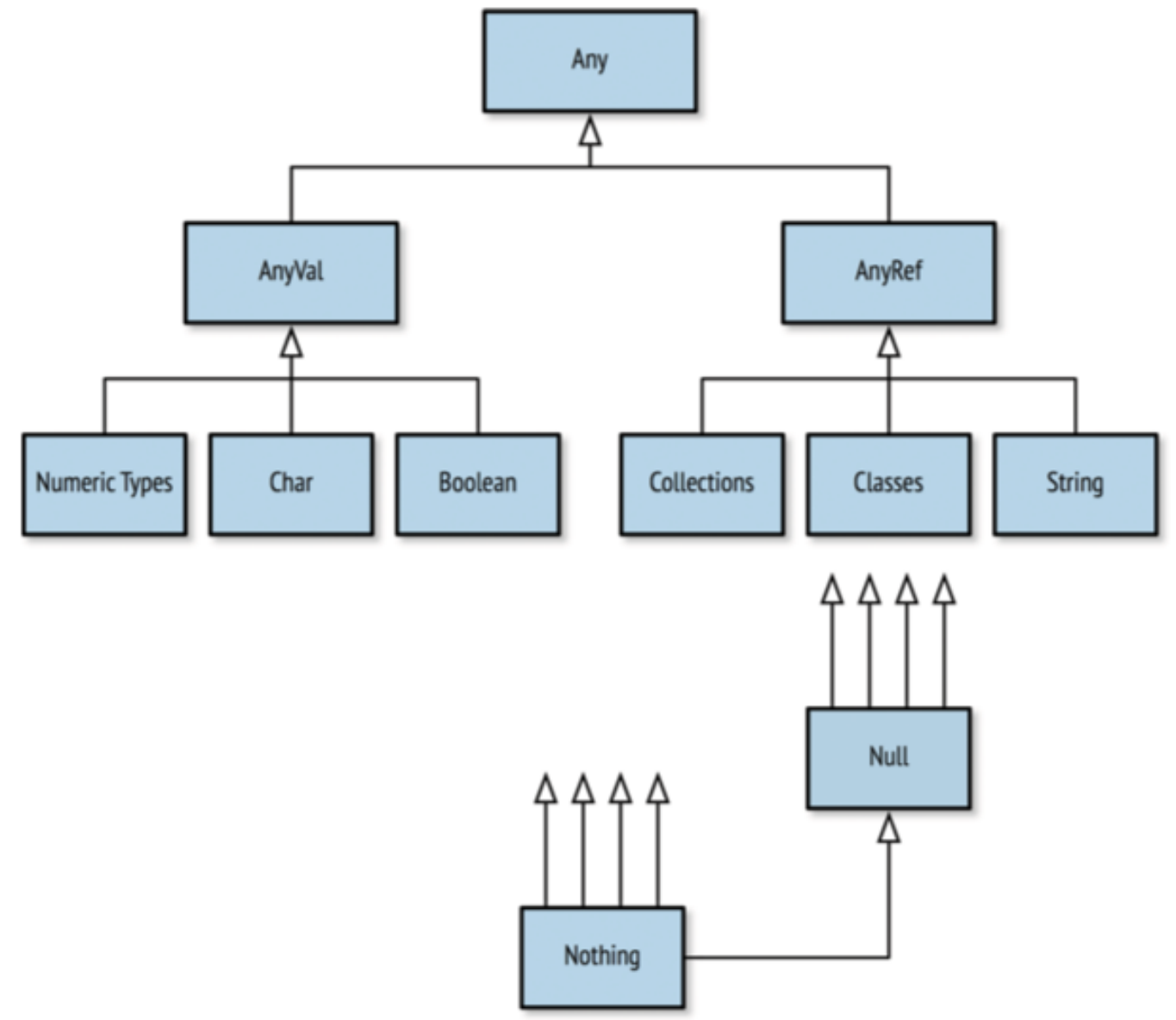
También es posible usar la salida con formato mediante la sentencia **printf**:

```
scala> val aproximacionPi=355/113f
aproximacionPi: Float = 3.141593

scala> println(f"Aproximacion de pi: $aproximacionPi%.5f")
Aproximacion de pi: 3,14159
```

Índice

Repaso de tipos básicos de Scala



Repaso de tipos básicos de Scala

Las clase Any, AnyVal y AnyRef son las clases básicas de la jerarquía de clases de **Scala**. A señalar:

- los tipos que heredan de AnyVal se conocen como tipos numéricos (incluyen los tipos numéricos más Char, Boolean y Unit)
- los tipos que heredan de AnyRef se conocen como tipos de referencia
- en la parte inferior aparecen los tipos Nothing y Null. El primero de ellos es subtipo de cualquier otro tipo y existe para proporcionar un tipo de devolución compatible para todas las posibles operaciones

Repaso de tipos básicos de Scala

- el tipo Null es subtipo de cualquier clase que derive de AnyRef y equivale a **null** de C++ o Java. Scala prefiere el uso de tipos reales sobre el uso de palabras reservadas
- Char es un tipo numérico pensado para almacenar un único carácter

```
scala> val c='A'
c: Char = A

scala> val i : Int = c
i: Int = 65

scala> val t : Char = 65
t: Char = A
```

Repaso de tipos básicos de Scala

- el tipo Boolean está limitado a los valores true y false
- los operadores relacionales pueden usarse de dos formas: con doble escritura del operador ($$ y ||) o con único carácter ($ y |). En el primer caso se aplican de forma lazy, mientras sea necesario. En el segundo caso, se evalúan de forma completa
- el tipo Unit representa ausencia de datos. Sería el equivalente a void en C++ o Java. Se representa con un par de paréntesis:

```
scala> val nada = ()
nada: Unit = ()
```

Repaso de tipos básicos de Scala

Al tratarse siempre con objetos, hay una serie de métodos genéricos en todas las clases que conviene conocer:

- asInstanceOf[<tipo>]

```
scala> val t : Char = 65
t: Char = A

t.asInstanceOf[Int]
res0: Int = 65
```

Repaso de tipos básicos de Scala

- getClass

```
scala> t.getClass
res2: Class[Char] = char
```

- isInstanceOf

```
scala> (5.0).isInstanceOf[Float]
res3: Boolean = false

scala> (5.0).isInstanceOf[Char]
res4: Boolean = false
```

Repaso de tipos básicos de Scala

- hashCode

```
scala> "A".hashCode
res6: Int = 65

scala> "a".hashCode
res7: Int = 97
```

- to<tipo>

```
scala> 20.toByte
res8: Byte = 20

scala> 47.toFloat
res9: Float = 47.0
```

Repaso de tipos básicos de Scala

- toString

```
scala> (5.0/7.0).toString
res10: String = 0.7142857142857143
```

Todos los tipos vistos hasta ahora se denominan escalares, ya que sólo contendrán un valor.

Índice

Tuplas

Contenedor ordenado de dos o más valores, que pueden tener tipos diferentes.

```
scala> val ejemplo1=(5, "sol", false, 9.43)
ejemplo1: (Int, String, Boolean, Double) = (5,sol,false,9.43)
```

Los elementos individuales pueden recuperarse a partir del nombre de la tupla, con índice basado en 1:

```
scala> println("Componente 2 de tupla: "+ejemplo1._2)
Componente 2 de tupla: sol
```

Tuplas

También pueden crearse tuplas mediante el operador ->. Se trata de una forma abreviada de representación de pares **clave - valor**. Esto sólo sirve para tuplas de dos componentes.

```
scala> val rojo="rojo" -> "0xff000000"
rojo: (String, String) = (rojo,0xff000000)

scala> val rojoInvertido = rojo._2 -> rojo._1
rojoInvertido: (String, String) = (0xff000000,rojo)
```