Programación Funcional en Scala

– Tema 1 –
Introducción a Scala

Jesús López González
jesus.lopez@hablapps.com

Programación Funcional en Scala
Habla Computing

Cursos ETSII-URJC 2015

Índice

Instalación del Entorno

Herramientas

- Editor de Texto Plano
  - EMACS (scala-mode2)
  - Sublime-Text
  - Gedit
  - Vim
  - Otros...
- Simple Build Tool (SBT)

Instalación del Entorno: SBT

¿Qué es SBT?

- Herramienta de Construcción para proyectos Scala (y Java).
- Creada por Mark Harrah
- Permite:
  - Definir y describir proyectos (mediante un *Domain-Specific-Language* embebido en Scala)
  - Compilar, Ejecutar y Probar un proyecto (modo continuo)
  - Descargar/Publicar dependencias
  - Integración con la REPL (*Read-Evaluate-Print-Loop*) de Scala

Instalación del Entorno: SBT

Instalación de SBT

- Descargar el *zip* de la página oficial: http://www.scala-sbt.org/download.html
- Descomprimir y añadir la ruta *$DIRECTORIO_SBT/bin* al *PATH* del sistema.
- Ejecutar *sbt(.bat)*
- Si todo ha ido bien, entraremos en la *shell* de SBT, que reconoceremos por el prompt "> " (La primera ejecución puede llevar varios minutos)

Índice

Primer Programa: Hola Mundo

Configuración del Proyecto

1. Creamos un directorio *hola-mundo* para albergar el proyecto.
2. Editamos el fichero *hola-mundo/build.sbt*, en el que pegaremos el código que aparece en la diapositiva 8. Prestar especial atención a las líneas en blanco existentes entre las propiedades, ya que son necesarias para el correcto funcionamiento de SBT.
3. Desde una terminal, accedemos al directorio del proyecto mediante *cd hola-mundo*.
4. Ejecutamos *sbt* y, si todo ha ido bien, veremos el prompt "> ".

Primer Programa: Hola Mundo

Listing 1: Contenido del fichero build.sbt

```
name := "hola-mundo"

version := "1.0"

organization := "es.urjc.etsii"

scalaVersion in ThisBuild := "2.11.5"

scalacOptions ++= Seq("-feature")
```

Primer Programa: Hola Mundo

Edición del fichero HolaMundo.scala

1. Editamos el fichero *hola-mundo/HolaMundo.scala*, en el que pegaremos el código que aparece en la diapositiva 9.

Listing 2: Contenido del fichero HolaMundo.scala

```
object HolaMundo extends App {
  println("¡Hola Mundo!")
}
```

Primer Programa: Hola Mundo

Compilación y Ejecución del Programa

1. Desde la shell de SBT ejecutaremos la orden *compile* (escribiendo el comando "compile" a continuación del prompt y pulsando ENTER). SBT sabe localizar el nuevo fichero fuente porque se encuentra en la raíz del proyecto. Más adelante, trataremos otras localizaciones donde también podemos depositar nuestros ficheros ".scala".
2. Si todo ha ido bien, obtendremos una traza cuya última línea comienza con el texto en tonalidad verde *"[success]"* y que ofrece unas breves estadísticas sobre el tiempo que llevó ejecutar la tarea. Si por el contrario, algo ha ido mal, la última línea mostrará un *"[error]"* en un llamativo color rojo.
3. Una vez compilado el código, podremos ejecutarlo mediante la orden SBT *run*, donde apreciaremos que el código *"¡Hola Mundo!"* se imprime por la consola.

Primer Programa: Hola Mundo

Algunas consideraciones sobre las tareas de SBT

- La tarea *run* depende de la tarea *compile*. SBT es inteligente y sabe identificar estas dependencias. Si modificamos un fichero y ejecutamos *run*, SBT será capaz de detectar que los fuentes han cambiado y procederá a compilar el código antes de ejecutarlo.
- Si escribimos el carácter virgulilla *'~'* antes de una tarea, por ejemplo *"~compile"*, SBT entrará en modo de ejecución continuo. Esto quiere decir que cada vez que se detecte que los fuentes se han modificado, automáticamente se pasará a ejecutar la tarea indicada. Para salir de este modo (y tal y como explica el propio SBT al pasar a esta modalidad) basta con pulsar ENTER.

Índice

Segundo Programa: Curso de Universidad

Configuración del Proyecto

1. Seguiremos los mismos pasos de configuración que se utilizaron en la diapositiva 7, reemplazando las referencias a *hola-mundo* por *gestor-alumnos*.
2. Este nuevo proyecto también contará con un test. Por ello, será necesario indicar que existe una dependencia con ScalaTest, añadiendo en el fichero *gestor-alumnos/build.sbt* la propiedad que aparece en esta misma diapositiva[a].

[a] Cuando modifiquemos el fichero *build.sbt* será necesario reiniciar SBT o ejecutar la orden *reload* para cargar los nuevos cambios.

Listing 3: Dependencia a ScalaTest (build.sbt)

```
libraryDependencies +=
  "org.scalatest" % "scalatest_2.11" % "2.2.4" % "test"
```

Segundo Programa: Curso de Universidad

Estructura del Proyecto

- SBT permite mantener separados el código principal y el código de pruebas. Por defecto, los fuentes principales deben situarse en el directorio *$RAIZ_PROYECTO/src/main/scala* mientras que los fuentes asociados a las pruebas se mantendrán en la ruta *$RAIZ_PROYECTO/src/test/scala*.
- La tarea SBT *test:compile* compila los fuentes correspondientes al directorio de pruebas. Por su parte, la tarea *test* ejecutará todos los tests que se encuentran en dicha ruta. Como es evidente, la tarea *test* depende de la ejecución de la tarea *test:compile*.
- Scala, al igual que Java, también permite la organización del código en paquetes. Aunque no es obligatorio, es muy recomendable que cada paquete cuente con su propio directorio. En este proyecto trabajaremos bajo el paquete *es.urjc.etsii*, es decir, nuestros ficheros Scala principales estarán en *gestor-alumnos/src/main/scala/es/urjc/etsii*.

Segundo Programa: Curso de Universidad

Nuestra primera clase: Alumno

1. Crearemos el fichero *gestor-alumnos/src/main/scala/es/urjc/etsii/Alumno.scala*, donde copiaremos el código que se encuentra en esta diapositiva.
2. La primera línea del snippet sirve para definir en qué paquete debe quedar recogido el contenido de las declaraciones de este fichero, en este caso, la clase *Alumno*.
3. La definición de la clase incluye el constructor, la definición de los atributos y la definición de los "getters".

Listing 4: Contenido del fichero Alumno.scala

```
package es.urjc.etsii

class Alumno(val nombre: String, val apellidos: String)
```

La REPL de Scala

- Si ejecutamos la tarea *console* desde la shell de SBT, accedemos a la REPL de Scala, con los fuentes del proyecto cargados.
- Esta herramienta es vital en el flujo de trabajo de un proyecto Scala, ya que nos permite experimentar con nuestro código de forma inmediata.

Listing 5: REPL de Scala

```
> console
scala> import es.urjc.etsii._
import es.urjc.etsii._

scala> new Alumno("Jordi", "Hurtado")
res0: es.urjc.etsii.Alumno = jordi_hurtado

scala> res0.nombre
res1: String = Jordi
```

Segundo Programa: Curso de Universidad

La clase *Curso*

- Los "valores de parámetro por defecto" nos permiten establecer el valor por defecto de un atributo. Si el programador no proporciona un valor en la instanciación, se utilizarán estos mismos.
- El tipo *Option[A]* nos permite tener un valor de tipo *A* que puede estar definido o no. En este contexto, la descripción del curso puede ser opcional.

Listing 6: Contenido del fichero Curso.scala

```
package es.urjc.etsii

class Curso(
  val nombre: String,
  val limiteAlumnos: Int = 30,
  val descripcion: Option[String] = None)
```

Segundo Programa: Curso de Universidad

Listing 7: Jugando con Curso en la REPL

```
scala> new Curso("Prog Func Scala", 40, None)
res0: es.urjc.etsii.Curso = es.urjc.etsii.Curso@4c5742d5

scala> new Curso("Prog Func Scala", 25)
res1: es.urjc.etsii.Curso = es.urjc.etsii.Curso@4ca65f7a

scala> new Curso("Prog Func Scala")
res2: es.urjc.etsii.Curso = es.urjc.etsii.Curso@3d29753c

scala> new Curso("Prog Func Scala", descripcion=Some("Piensa en
    Funcional"))
res3: es.urjc.etsii.Curso = es.urjc.etsii.Curso@8b62a98
```

Segundo Programa: Curso de Universidad

La clase Gestion

- Un método se define bajo el formato
  def nombre_metodo(arg1: TArg1, arg2: TArg2...): TRet = ...
- NO existe un token *return* para devolver el valor de la función. En su lugar, la función devolverá el valor resultado de la última instrucción existente dentro del bloque[a].
- Se puede apreciar cómo el tipo que devuelve el método también es opcional, lo que nos permite marcar situaciones de error, mediante un valor no definido (*None*).

---

[a] De hecho, el patrón es más genérico. Una expresión-bloque devuelve el valor de salida de la última instrucción que se encuentra dentro del bloque.

Listing 8: Contenido del fichero Gestion.scala

```
package es.urjc.etsii

class Gestion(val relacion: Map[Curso, List[Alumno]] = Map()) {
  def inscribirAlumno(al: Alumno, cr: Curso): Option[Gestion] = {
    val alumnos = relacion(cr)
    if ((al.size >= cr.limiteAlumnos) || alumnos.contains(al))
      None
    else
      Some(new Gestion(relacion + (cr -> (al :: alumnos))))
  }
}
```

Segundo Programa: Curso de Universidad

Listing 9: Invocando un método en la REPL

```
scala> val al = new Alumno("Jordi", "Hurtado")
al: es.urjc.etsii.Alumno = jordi_hurtado

scala> val cr = new Curso("Programacion Funcional en Scala")
cr: es.urjc.etsii.Curso = es.urjc.etsii.Curso@6fc54b7c

scala> val gs = new Gestion()
gs: es.urjc.etsii.Gestion = es.urjc.etsii.Gestion@15e21f69

scala> gs.inscribirAlumno(al, cr)
res0: Option[es.urjc.etsii.Gestion] =
    Some(es.urjc.etsii.Gestion@13a866c6)
```

Segundo Programa: Curso de Universidad

Probando el método Gestion.inscribirAlumno

1. Crearemos un fichero GestionTest.scala el el directorio *gestor-alumnos/src/test/scala/es/urjc/etsii/test*
2. Como ya mencionamos anteriormente, el hecho de depositar el fichero en esa ruta nos permite la integración con la tareas de SBT.
3. Una vez efectuada la implementación del caso de prueba (ver el listado en la diapositiva 23), pasamos a ejecutarlo mediante la tarea *test* de SBT.
4. Al finalizar la ejecución de la tarea veremos una relación con los casos de prueba correctos y erróneos.

Listing 10: Probando el método inscribirAlumno

```
package es.urjc.etsii.test

import scalatest._
import es.urjc.etsii._

class GestionTest extends FlatSpec with Matchers {
  "El gestor" should "permitir a un alumno inscribirse" in {
    val gtn = new Gestion(Map(curso -> List(maria, jose)))
    val optGtn = gtn.inscribirAlumno(ana, curso)
    val exp = Map(curso -> List(ana, maria, jose))
    optGtn.get.relacion should be (exp)
  }
  it should "rechazar a un alumno si ya está inscrito" in {
    val gtn = new Gestion(Map(curso -> List(maria, jose)))
    gtn.inscribirAlumno(maria, curso) should be (None)
  }
}
```

Segundo Programa: Curso de Universidad

Probando el método Gestion.inscribirAlumno

1. Crearemos un fichero GestionTest.scala el el directorio *gestor-alumnos/src/test/scala/es/urjc/etsii/test*
2. Como ya mencionamos anteriormente, el hecho de depositar el fichero en esa ruta nos permite la integración con la tareas de SBT.
3. Una vez efectuada la implementación del caso de prueba (ver el listado en la diapositiva 23), pasamos a ejecutarlo mediante la tarea *test* de SBT.
4. Al finalizar la ejecución de la tarea veremos una relación con los casos de prueba correctos y erróneos.

Índice

Takeaways

Conocemos las tareas básicas de SBT: *compile*, *run*, *test*... y somos capaces de configurar un nuevo proyecto.

Sabemos trastear con la REPL para realizar pequeños experimentos con nuestro proyecto.

Podemos desarrollar una especificación de prueba usando el framework ScalaTest.

Hemos tenido nuestra primera toma de contacto con la sintaxis de Scala. *¡No es necesario que entendamos toda la sintaxis vista en la sesión!*