A Scala Tutorial for Java programmers

Version 1.3
January 16, 2014

Michel Schinz, Philipp Haller

PROGRAMMING METHODS LABORATORY
EPFL
SWITZERLAND

1 Introduction

This document gives a quick introduction to the Scala language and compiler. It is intended for people who already have some programming experience and want an overview of what they can do with Scala. A basic knowledge of object-oriented programming, especially in Java, is assumed.

2 A first example

As a first example, we will use the standard *Hello world* program. It is not very fascinating but makes it easy to demonstrate the use of the Scala tools without knowing too much about the language. Here is how it looks:

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

The structure of this program should be familiar to Java programmers: it consists of one method called `main` which takes the command line arguments, an array of strings, as parameter; the body of this method consists of a single call to the predefined method `println` with the friendly greeting as argument. The `main` method does not return a value (it is a procedure method). Therefore, it is not necessary to declare a return type.

What is less familiar to Java programmers is the **`object`** declaration containing the `main` method. Such a declaration introduces what is commonly known as a *singleton object*, that is a class with a single instance. The declaration above thus declares both a class called `HelloWorld` and an instance of that class, also called `HelloWorld`. This instance is created on demand, the first time it is used.

The astute reader might have noticed that the `main` method is not declared as `static` here. This is because static members (methods or fields) do not exist in Scala. Rather than defining static members, the Scala programmer declares these members in singleton objects.

2.1 Compiling the example

To compile the example, we use `scalac`, the Scala compiler. `scalac` works like most compilers: it takes a source file as argument, maybe some options, and produces one or several object files. The object files it produces are standard Java class files.

If we save the above program in a file called `HelloWorld.scala`, we can compile it by issuing the following command (the greater-than sign '>' represents the shell prompt and should not be typed):

```
> scalac HelloWorld.scala
```

This will generate a few class files in the current directory. One of them will be called `HelloWorld.class`, and contains a class which can be directly executed using the `scala` command, as the following section shows.

2.2 Running the example

Once compiled, a Scala program can be run using the `scala` command. Its usage is very similar to the `java` command used to run Java programs, and accepts the same options. The above example can be executed using the following command, which produces the expected output:

```
> scala -classpath . HelloWorld

Hello, world!
```

3 Interaction with Java

One of Scala's strengths is that it makes it very easy to interact with Java code. All classes from the `java.lang` package are imported by default, while others need to be imported explicitly.

Let's look at an example that demonstrates this. We want to obtain and format the current date according to the conventions used in a specific country, say France[1].

Java's class libraries define powerful utility classes, such as `Date` and `DateFormat`. Since Scala interoperates seemlessly with Java, there is no need to implement equivalent classes in the Scala class library–we can simply import the classes of the corresponding Java packages:

```
import java.util.{Date, Locale}
import java.text.DateFormat
import java.text.DateFormat._

object FrenchDate {
  def main(args: Array[String]) {
    val now = new Date
    val df = getDateInstance(LONG, Locale.FRANCE)
    println(df format now)
  }
}
```

[1] Other regions such as the french speaking part of Switzerland use the same conventions.