CS 2340 Objects and Design - Scala

Scala Introduction

Christopher Simpkins
chris.simpkins@gatech.edu

Scala Overview

Scala is the future:

- truly object-oriented
- functional
- powerful static type system
- runs on JVM, seamlessly interoperates with Java (mostly ...)

Today we'll have a whirlwind tour of the Scala ecosystem and basics of Scala's command-line tools.

A Scala Ecosystem

To run Scala you need

- Java 6 or later – the more up to date the better – either from OpenJDK or Oracle.
- Scala
- SBT, the Scala Build Tool (technically SBT stands for Simple Build Tool, but it’s not simple, so I refuse to call it Simple Build Tool)

If you use Linux I recommend using your distro’s package manager.
For Mac users, I recommend Homebrew.

The Scala REPL

Like Python, invoking the `scala` runtime without arguments launches the REPL (Read-Eval-Print Loop):

```
$ scala
Welcome to Scala version 2.11.1 (Java HotSpot(TM) 64-Bit Server VM,
    Java 1.8.0_11).
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

REPLs are awesome, as you'll see.

Values and Variables

```
scala> val a = 1
a: Int = 1

scala> a = 2
<console>:8: error: reassignment to val
       a = 2
         ^

scala> var b: Double = 2
b: Double = 2.0

scala> b = 3.0
b: Double = 3.0
```

- Every value has a type which, when possible, is inferred in the absence of an explicit type annotation
- Types come after names, *name: Type*
- `val`s are immutable, `var`s are reassignable

Functions

Every function has a value

```
scala> def mean(nums: Double*) = nums.sum / nums.size
mean: (nums: Double*)Double

scala> mean(100, 95)
res24: Double = 97.5
```

Note that:

- The function body is a single expression, so no need for curly braces (but legal to include them)
- Every function has a type, which includes its argument list and result type (Scala uses term "result" type instead of "return "type).
- The type of the function is `(nums: Double*) Double` which means "a function taking a variable number of `Double` parameters and returning a `Double`"
- The result type is inferred.

More on Functions

Every block is an expression whose value is the last value in the block.

```
scala> def normalize(s: String): String = {
     |   val trimmed = s.trim()
     |   val upper = trimmed.toUpperCase
     |   upper
     | }
normalize: (s: String)String

scala> normalize("  aBcd \n")
res18: String = ABCD
```

A function that returns nothing returns something, in particular the single instance of the `Unit` type (Scala's equivalent of Java's `void`).

```
scala> def shout(s: String) {
     |   println(s.toUpperCase)
     | }
shout: (s: String)Unit

scala> shout("hello")
HELLO
```

Note that the lack of a = in the function definition.

Classes and Objects

Take a moment to appreciate the boilerplate-free beauty here:

```
scala> class Person(val name: String)
defined class Person

scala> val bob = new Person("Bob")
bob: Person = Person@7c3c453b

scala> bob.name
res19: String = Bob

scala> class GtStud(name: String, val gtname: String) extends
    Person(name)
defined class GtStud

scala> val alice = new GtStud("Alice", "alice1")
alice: GtStud = GtStud@13908f9c

scala> alice.name
res22: String = Alice

scala> alice.gtname
res23: String = alice1
```

Scala Programs and Libraries

Scala code can be delivered as a program or a library.

- Two kinds of scala programs: scripts and applications
  - A Scala script is a file containing Scala code, the last line of which is en executable expression or statement
  - A Scala application is an object, defined with Scala's `object` keyword, that has a method with the signature `def main(args: Array[String])` (name of parameter doesn't have to be args, but has to be of type `Array[String]`)
- A library is a `.jar` file containing a tree of Scala classes
  - Technically, a `.jar` file contains a tree of JVM (Java Virtual Machine) bytecode classes that could be compiled from any source language, such as Java, Scala, Jython, Groovy, Clojure, etc.

To compile and run Scala code you need JRE 1.6 or higher (1.7 or 1.8 recommended).

Running a Scala Script

Given a file `foo.scala` with the following content:

```
class Writer(val repetitions: Int = 1) {

  def say(text: String) {
    sayRepeatedly(text, repetitions)
  }
}

def sayRepeatedly(text: String, times: Int) {
  println(text*times)
}

val repetitions = if (args.length > 0) args(0).toInt else 1
val writer = new Writer(repetitions)
writer.say("foo.scala run as a Scala script.")
```

we can run it like this:

```
[chris@nijinsky ~/examples]
$ scala foo.scala
foo.scala run as a Scala script.
```

Scala Scripts as Shell Scripts

Add following lines to top of Scala script to turn into a Unix shell script:

```
#!/bin/sh
exec scala "$0" "$@"
!#
```

and run it like this:

```
[chris@nijinsky ~/examples]
$ chmod +x foo.scala
[chris@nijinsky ~/examples]
$ ./foo.scala
foo.scala run as a Scala script.
```

On Windows you can acheive the same effect by giving your file a .bat extension and putting this at the top:

```
::#!
@echo off
call scala %0 %*
goto :eof
::!#
```

Compiling Scala Code

Let's try compiling our `foo.scala file`:

```
[chris@nijinsky ~/examples]
$ scalac foo.scala
foo.scala:8: error: expected class or object definition
def sayRepeatedly(text: String, times: Int) {
^
foo.scala:12: error: expected class or object definition
val repetitions = if (args.length > 0) args(0).toInt else 1
^
foo.scala:13: error: expected class or object definition
val writer = new Writer(repetitions)
^
foo.scala:14: error: expected class or object definition
writer.say("foo.scala run as a Scala script.")
^
four errors found
```

We have to remove the executable expressions and statements and move the function sayRepeatedly inside a class or object. Compiling is for class and object definitions.

Scala Applications

Here's `foo2.scala`, which defines a Scala *application* that does the same thing as our script.

```
object Example {
  def main(args: Array[String]) {
    val repetitions = if (args.length > 0) args(0).toInt else 1
    val writer = new Writer(repetitions)
    writer.say("Example object run as a Scala application.")
  }
}
class Writer(val repetitions: Int = 1) {
  def sayRepeatedly(text: String, times: Int) {
    println(text*times)
  }
  def say(text: String) {
    sayRepeatedly(text, repetitions)
  }
}
```

Compiling and Running Scala Applications

Given our

```
[chris@nijinsky ~/examples]
$scalac foo2.scala
[chris@nijinsky ~/examples]
$ ls
Example$.class Example.class Writer$.class Writer.class
foo.scala foo2.scala
```

Now we can run this Scala application like this:

```
[chris@nijinsky ~/examples]
$ scala -cp . Example
Example object run as a Scala application.
```

SBT

The Scala Built tool, SBT, is a very powerful buld system. Invoke it in interactive mode with `sbt`. Here are a few useful commands:

```
$ sbt
> help      # Describe commands.
> tasks     # Show the most commonly-used, available tasks.
> tasks -V  # Show ALL the available tasks.
> compile   # Incrementally compile the code.
> test      # Incrementally compile the code and run the tests.
> clean     # Delete all build artifacts.
> ~test     # Run incr. compiles and tests whenever files are saved.
            # This works for any command prefixed by "~".
> console   # Start the Scala REPL with project classes loaded
> run       # Run one of the "main" routines in the project.
> show x    # Show the definition of variable "x".
> eclipse   # Generate Eclipse project files.
> gen-idea  # Generate IntelliJ IDEA project files.
> exit      # Quit the REPL (also control-d works).
```

SBT Build Definitions

The simplest way to define a build in SBT is with a `build.sbt` in the project's root directory, which typically looks soemthing like this:

```
name := "afabl"

version := "0.3"

scalaVersion := "2.11.1"

libraryDependencies += "org.scalatest" % "scalatest_2.10" % "2.1.0" %
    "test"
```

- SBT assumes the Maven standard directory layout; can configure different layout, but shouldn't
- Note the blank lines between each line in `build.sbt`; those are necessary
- SBT uses Ivy for dependency management, which uses Maven's repository system; you can use any Java library in any accessible Maven repo

Editor and IDE Support for Scala

All three major Java IDEs (IntelliJ, Ecplipse, and NetBeans) have Scala plugins, but Eclipse and IntelliJ are far more popular. Plugins support all the usual things:

- compile-as-you-go with error highlighting
- autocompletion
- finding symbols, definitions, etc.
- refactoring (probably the most compelling feature of IDEs)

plus a unique feature for Scala: worksheets (Eclipse, IntelliJ).

From an SBT project you can create an Eclipse or IntelliJ project with

```
sbt eclipse
```

or

```
sbt gen-idea
```

and re-run whenever you change the SBT build definition.

Where to Go from Here

Lots more to learn. I recommend using one or both of the first two of thse books for self study:

- Programming in Scala, 2ed - "the" Scala book, written by Martin himself (and GT PhD grad Lex Spoon). Focuses on language (no discussion of tools like SBT) and is a bit outdated (written for Scala 2.8 so, e.g., covers old Actors library instead of now standard Akka), but is an excellent book
- Programming Scala, 2ed by Wampler and Payne. Pre-publish release as an ebook. If you only get one Scala book, get this one. Covers the language and tools, such as SBT.
- Learning Scala by Schwatz. A gentle introduction. Also an early release ebook. I'm am deferring judgment on its quality.
- Scala Cookbook by Alexander. Your standard O'Reilly cookbook, with much more tutorial-style coverage of tools and tasks like web services and database access, as well as common general programming tasks.