Scala Style Guide

Spring 2020

Contents

1 Introduction

This document is a Scala style guide. The official Scala style guide can be found here (click on the word "here"—it's a link!). Much of our style guide is taken directly from the official one.

Scala style is similar in many ways to Java style. This guide focuses on the style conventions unique to Scala, so for any cases not covered here, please refer back to our Java style guide.

As always in CS 17/18, your assignments will be graded on style as well as functionality, so this guide should be a valuable tool for you while coding.

2 Naming

The Scala naming conventions are similar to those of Java. But one difference is that while Java insists on type parameters (i.e., generics) with single-character upper-case names, more descriptive names are often used in Scala (and when less descriptive names are used, they are not restricted to the usual characters like `E`, `T`, etc.). Here are some examples:

```scala
class List[A] {
  ...
}

class Hashmap[Key, Value] {
  ...
}

def task[A, B, C](arg1: A, arg2: B): C = ...
```

Another difference is that, it is standard practice for variable names to be short in purely functional methods. For example:

```scala
def sub(a: Int, b: Int): Int = a - b
```

Short names like these would be bad practice in Java, but in Scala, these names are good, because purely functional methods can be short and sweet—one expression comprised of just a few variables. So short names in Scala end up improving readability.

3 Formatting

Spacing

As usual in CS 17/18, the space bar is our friend.

Always insert spaces around infix operators, like this:

```
// Good style
x = x + 18

// Bad style
x=x-16
```

Always insert spaces around curly braces as well, but never insert a space between parentheses (left or right) and the code they delimit, or after a method's name. For example:

```
// Bad style: the opening brace needs space (to breathe)!
def bad(end: Int){
  for (i <- 0 until end) {
    println(i)
  }
 }
```

```
// Bad style: you don't want space after the method's name
// or around the parentheses!
def alsoBad ( end: Int ) {
  for (i <- 0 until end) {
    println(i)
  }
}
```

```
// Good style
def good(end: Int) {
  for (i <- 0 until end) {
    println(i)
  }
}
```

Indentation

Whenever an indent is required, it should be two spaces.

Sometimes a single expression will be too long to fit on one line. When this happens, you should try to break the expression up into multiple shorter expressions to avoid line wrapping.

When this is impossible, or severely detracts from readability, you can break the expression up into multiple lines, and insert two spaces at the beginning of each new line.

Here is an example where this is definitely not impossible, but it illustrates something else: you must end each sub-statement with a still open left parenthesis or an infix operator! Otherwise, Scala's compiler will infer that the statement has ended, and all hell could break loose.

```
// Right
val answer = 1 + 2 + 3 + 4 + 5 + 6 +
  7 + 8 + 9 + 10 + 11 + 12 + 13 +
  14 + 15 + 16 + 17 + 18 + 19 + 20

// Also right
val answer = (1 + 2 + 3 + 4 + 5 + 6
  + 7 + 8 + 9 + 10 + 11 + 12 + 13
  + 14 + 15 + 16 + 17 + 18 + 19 + 20)

// Wrong, on two counts
val answer = 1 + 2 + 3 + 4 + 5 + 6
  + (7 + 8 + 9 + 10 + 11 + 12 + 13)
  + 14 + 15 + 16 + 17 + 18 + 19 + 20
```

If you need to call a method that takes in so many arguments that the call won't fit on one line, you should put each argument on a line by itself, indented two spaces from the current indent level.

```
myFunction(
  aVeryLongVariableName,
  anotherEvenLongerVariableName,
  "Scala is my favorite programming language",
  "these are a few of my favorite things:")
```

Likewise, to return the value of such a call to a variable, you should do so as follows:

```
val myResult =
  myFunction(
    aVeryLongVariableName,
    anotherEvenLongerVariableName,
    "Scala is my favorite programming language",
    "these are a few of my favorite things:")
```

4 Class Declarations

All constructors should be declared on one line, unless that line gets too long. When the line gets too long, you should put each argument to the constructor on its own line, with each argument indented *four* spaces (not two!). If the class extends anything, you should also put everything it extends on a line of its own; however, those lines should just be indented two spaces.

For example:

```
class Student(name: String, semesterLevel: Int) {
  ...
}
```

```
class Student(
    name: String,
    semesterLevel: Int,
    birthday: String,
    concentration: String,
    livesOnCampus: Boolean,
    Courses: List[String])
  extends Entity
  with Serializable {
  ...
}
```

Note: As in Java, the opening curly brace is always on the same line as the end of the corresponding declaration, be it a class or a method declaration.

5 Functional Paradigms

Here is the pattern matching style in Scala:

```
// Good style
def sum(ls: List[Int]): Int = ls match {
  case hd :: tail => hd + sum(tail)
  case Nil => 0
}

// Bad style
def sum(ls: List[Int]): Int = {
  ls match {
    case hd :: tail => hd + sum(tail)
    case Nil => 0
  }
}
```

Observe that the `match` keyword appears on the same line as the method's declaration.

The style is similar for anonymous functions: the arrow should appear on the same line as the function's declaration.

```
// Good style
def mySumFunction = { (ls: List[String]) =>
  ls.map(_.toInt).foldLeft(0)(_ + _)
}
```

```
// Bad style
def mySumFunction = {
  (ls: List[String]) =>
  ls.map(_.toInt).foldLeft(0)(_ + _)
}
```

6 Comments

As always when programming, it is important to provide documentation. When commenting your Scala programs in CS 18, you should follow the guidelines described in the CS 18 Java style guide. By so doing, you will actually be writing Scaladoc, not Javadoc, but this distinction should not really concern you. There is just one important difference to note: in Scaladoc, the left-hand margin of asterisks is in the third column, not the second, as is the convention in Java. For example:

```
/**
 * This is a Java-style asterisk format.
 */

/**
  * This is a Scala-style asterisk format.
  */
```

Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS18 document by filling out the anonymous feedback form: `https://cs.brown.edu/courses/cs018/feedback`.