SCALA

Hector Sanchez
sanchezh@.email.arizona.edu

Samantha Felzien
sfelzien@email.arizona.edu

April 7, 2020

ABSTRACT

1 Introduction

2 History

Scala was developed by Martin Oderksky in 2001 and first released in 2004. Scala was designed to fuse both functional and object oriented programming [3]. While Scala is not an extension of Java, it is interoperable and translates to Java bytecode [4]. The name is derived from the word scalable, meaning it can grow the demand of users. Scala is widely used today, as teams on Twitter, Apple Inc, The New York Times, Google, and Walmart Canada use it [3].

3 Control Structures

The Scala programming language is one that features all of the control structures we are familiar with from other more-common programming languages such as Java. Among these control structures we find the essential "if", "while", "for", and "case" structures. Contrary to the syntax found in languages such as Python but similar to Java, Scala requires the use of curly-braces to define the body and procedures within the control structure, such as the following:
while (true){

println ("this is true")
}
Much the same as Java, Scala also allows for the omission of these curly braces in certain structures if they are for

functional use [6]. We often think about this structure in Java when we have a condition that must be met and a one-line instruction to follow if that condition is met, such as the example below implies. As you might imagine however, not all control structures can fit this simplified syntax, such as for loops and while loops [6]. In the case of for-loops, the omission of the curly braces is dependent on the inclusion of a "yield" clause in order to avoid infinite loops.
if (boolVal)println("bool was true")

Unlike Java however, there are some control structure elements that are not present in Scala. One such example is the Java ternary conditions, in which the syntax is made easy by the use of colons and question marks to essentially represent an if/else statement in one line of code. Scala doesn't fall far behind in this concept however, but rather offers a less-elegant but still very functional approach. Syntax in Scala is as follows:
val b = if (x) "x" else "y"
We can note the similarities in structure to other programming languages, but with Scala taking a more literal approach

while also implementing the previously mentioned omission of the curly braces.
Despite this however, Scala does offer some very elegant and useful features in their control structure syntax, one

of which being “for-comprehension” [7]. This syntactical notation allows for the traversal of multiple List elements within the for-loop declaration while also allowing for other conditionals to dismiss certain elements, returning only the desired elements using only two lines of code or so. In essence, Scala for-loop declarations simplify the foreach functionality and allows it to be applied to multiple arguments [7]. The code below demonstrates this feature with a list of numbers ranging from 1 to 10, in which the for loop is responsible not only for iterating through the list, but also picking out numbers greater than 5 and appending those numbers to a very different list.

```
val s = for (n <- nums if (n > 5)) yield n
```

When compiled, list s will return all numbers greater than 5, saving us additional lines of code or at least make our code a tad easier to read. This example is of the most simplistic nature, and this feature can be a very outstanding tool when iterating through lists of objects in which individual values or attributes must be checked for each object, since Scala allows for class fields to be references within if statements. [7]

Finally, we arrive at case conditionals, which are also embedded with these short syntactical shortcuts that allow for one-line code segments useful in more complex case statements that could otherwise be quite messy. At it’s simplest form, the Scala case conditional is similar to those found in other languages, however Scala shines when creating more complex case-statements which may also require embedded conditionals or pattern matching. [6]

```
x match{

case a if (x.status == “bored”) => findSomethingToDo(x)
case a if (x.status == “tired”) => rest(x)
case _ => doNothing()
}
```

In the above code snippet we see the implementation of these if-conditionals within the case statements as well as the final pattern-matching case similar to that in Haskell. Such syntax is useful for creating elegant and easy to follow code that is often characteristic within the Scala programming language.

4 Data Types

5 Sub Programs

References

[1] Community.

[2] Documentation.

[3] History of scala - javatpoint.

[4] The scala programming language.

[5] Gavin Bisesi. Daenyth/taklib.

[6] Heather Miller and Tyler Nienhouse. Control structures.

[7] Seth Tisue and Heather Miller. For comprehensions.