Nick Stone

CSCI 300

Programming Languages

Dr. Johnson

Draft Case Scala

Overview

Scala is a term used for scalable language, thus meaning that Scala's purpose and design was to create a language that would grow with you. For some programming languages we see that some of the weaknesses of the language are that the application of these languages cannot support a large number of users. One such language Is Ruby and the on rails framework. While it allows for quick development time it does not handle thousands of users at once contacting a website. To counteract this, Scala had hoped to take the best aspects of Java and try and shorten Java commands but also allow Scala to support Java commands. Furthermore, by allowing both object-oriented programming and functional programming tactics you could build and develop a website, and have it scale with the number of users you have. This allowed for quick reusable code that would be able to scale for an application of any size. Some popular websites that used Scala were Reddit, Twitter and LinkedIn.

Scala was created by Martin Odersky in 2004 in hopes that Scala could be used to augment Java. The goal was for Scala to allow programmers to write code that would follow either functional programming practices or object-oriented practices. Scala as you will see later

in this document, was intended to be less verbose than Java and from this allows for programs to be written quicker. The purpose however was so that Scala could be interoperable with Java, which means that Scala can execute Java code. From this we also see that Java and Scala will have a lot of similarities, some of this being pass by value, static scoping, automatic garbage collection and the same primitive types.

Scope

Scala is a high-level programming language that incorporates both function and object-oriented programming. Scala operates on the Java Virtual Machine and shares a lot of similarities with Java including the ability to run and execute Java commands and functions from Scala without a need to add any extra steps. Scala also has the ability to execute JavaScript code without any extra steps as well. According to the Scala documentation reference, everything in Scala is stored as an object except for primitive data types that are instantiated in functions. These latter variables are stored on the stack just as in Java however Scala has eliminated the need to type new and assigns everything as an object. This feature can be problematic for some languages, but Scala is able to use Java automated garbage collection so with no need to create a delete or a deconstructor Scala allows for easy code writability and reliability. (Declarations, 2)

Since almost everything is instantiated as an object, the objects are stored on the heap. Scala is a programming language that also has an order or precedence for binding of variables. From this we see Scala has shadow scoping or the idea that If the same variable is assigned to two different order of precedence the variable can be instantiated in two different scopes.

(Documentation chapter 2 Scala Documentation) These variables are also instantiated at runtime and thus the storage binding is done at run time. Scope in Scala is very similar to Java in terms of scope. The idea of scope is static, and the program will look for a local variable, then a parameter variable and then finally looking for a global variable.

Data Types

While the scoping system of Scala was very similar to Java the type system in Scala is very different than that of Java and has the ability to create several types of data types. Several of these types of data structures and data types were new to me and required me to read about them in depth multiple times before I felt comfortable in understanding how the type system not only worked but how I could describe it or explain it to someone. When invoking the **val** command the value that is assigned to that variable cannot change. However, when **var** command is invoked, the variable can change. Furthermore, Scala supports inferred types so that need we do not mean to assign a type to a value.

Scala supports all of the same primitive types as Java and allows for more types to be created. Scala is a compiled language just like Java. Readability is actually hurt by this inferred typing and use of non-verbose Java code because it requires one to have to read through and also infer what variables are which type. Scala is strongly typed. The compiler will not compile the error and throw an error if a value or variable is passed that is not compatible with the parameter type in the function or if an int is passed when the function call requires a string.

One of the unique functions in Scala is that one can declare any number of variables within a variable allowing the creation of a list of arrays without instantiating an array in Java.

Everything is an object including primitive data types. Any type can be assigned to be null, so strings can be null as well as can be Boolean or any other primitive data type. Furthermore, we can also assign the annotation types as well as tuple types to be Null. Annotation types. We can add an annotation to a variable. For example, we can add a string to describe what a variable is doing or what a variable is.

This is very important in Scala since almost everything is an object, we can annotate what parameters are required. Thus, this allows one to create types with the ability to access specific locations of the variable within the type. The type determination is static thus it is determined by the compiler in order to allow the compiler to optimize the code. The type system in Scala allows for polymorphism to occur between many of the types such as between integers as well as doubles. This is done since every data type is an object and, they all start by inheriting the same class.

Tutorial

Functional Programming:

When understanding and programming in language that allows multiple paradigms, we have to understand how functional programming and object-oriented programming are different. Since functional programming does not allow for objects then we have to come up with a methodical way to sort and manipulate the data. The following programs are two simple functional programs these being absolute value and computing factorial.

```
def absoluteValue(int x){
      if(x < 0){
        return x * -1;
```

```
    }
    else{
      return x;


    }
    def factorial(int x){
      if(x == 1){

        return 1;
      }
      else{

        return x * factorial(x-1);
      }

    }
```

We can see from this code example that the syntax of Scala is very similar to Java, in the form of not only scoping but also the brackets around if and else statements. In the given scenario I decided to add int to x in order to allow less confusion when describing the creation of this topic. Many of these lines of code are similar to Java and we see when building these functions, we do not utilize any functions and compute numbers. We can see the second function is recursion and follows the same flow as it would in Java.

```
def product1(a: Int, b: Int) :  Int = {

     a - b;
 }

def product(a:Int,b:Int) :Int = {a-b}


 def main(args: Array[String]): Unit = {

   var d = product(30,20);
   println(d);
   var h = product1(10,20);
   println(h);

 }
```

Sorting (Selection Sort)

The next algorithm seen is the selection sort function and we can see that this is very different than the functional programs that I used. We have implemented selection sort in Scala.

```
var Array2 = Array(54,6,734,48);
    var temp = 0;

    for(i <- 0 to Array2.length-1){

      for( j <- 0 to i){

        if(Array2(i) < Array2(j)){
          println("Swap");
          println("-----------------------------------------");
          temp = Array2(i);

          Array2(i) = Array2(j);

          Array2(j) = temp;
          for(g<- 0 to Array2.length-1){

              println(Array2(g));

          }
          println("-----------------------------------------");
        }

      }

    }
```

We can see when we create the array the syntax is slightly different from that of Java. Then with the creation of the for loops these are very different, and we can create and instantiate a for loop by using the for keyword. Then we create the index variable or the value that will be used as our index IN the scenarios given j I and g are the incremental variables. Then we use an arrow to assign a value to that variable and the end size of the variable. For the code given, the variable I is used until the end of the list and we use j until i. When printing and traversing the array, we access data types by using parentheses to access a specific data point in the array.

Linear Search

The next code snippet that I have implemented is a normal linear search. This is used as a function and once again assigned x to be an int a we set that to be a var targ. We could have used a val since the val key word does not allow the variable that is constrained to change.

```
def findtarg(int x){
 var targ = x;
 var found = false;

for(lk<- 0 to Array2.length-1){

  if(targ == Array2(lk)){
    found = true;


  }



}

println(found);



}
```

We instantiate a function by declaring a def, then the name of the function and the variable that we take in. In this scenario we are just looking to see if the integer x is found within the array that I have instantiated as a global variable. We can see that this method call starts by assigning the variable lk to 0. Then we search from the 0 position in the array until the very end, this is to ensure we have searched the entire list for the target value. If the target value has been found we then set a variable found to be true, else the variable will remain false and we print the variable.

The following function is a functional programming way to search through an array. This highlights Scala's functionality because we get to utilize Scala's functional style and we can write a recursive function as follows.

```
def search(a:Array[Int], targ: Int,count:Int){
  var p = a;
  var t = targ;
  var h = count;

  if(h > p.length-1){
    print("Not Found");
    return false;
  }
  else if (p(count) == t){
    print(t);

    return true;

  }
  else {
    h = h + 1;
     search(p,t,h);

  }


}
```

The function takes in three parameters and goes through the logical flow statements. If the bottom statement is reached the recursive call recalls itself with the new value of h and recalls the functions.

Comparing Scala to Java

Over View of what's being compared

The following code segments that are listed here after are the same implementation of a percent error calculator in both Java and Scala. We can see from the differences in the code not

only the syntactic differences but also the logical flow and overall structure of Scala being designed to handle functional programming more efficiently than that of Java.

Java:

```
	public static void main(String[] args) {


		System.out.println(error(110,100));

	}

	public static double absolute(double d) {


		if(d < 0) {
			d = d * -1;
			return d;
		}

		return d;
	}
	public static double error(double theory, double actual) {

		double temp = 0;

		temp = theory - actual;
		temp = absolute(temp);
		double temp1 = temp / theory;

		temp1 = temp1 * 100;

		return temp1;

	}



	}
```

We see from the following code segment that the operation and function at hand has a lot of logic breaks within the program. The use of this program highlights an issue in programming functional programming in Java that Scala eliminates. We can see just from the five logical return statements all return the same value and only calls one other function when within the function. If the function were to call many other functions and each function returned

something unique the code segment could be bulky and hard to follow. However, when looking at the Scala code given that is a solution to the same problem, we see no return types. This is due to the fact that Scala will always return the last segment of code executed. This allows the use of return statements and other function calls to be minimized and the use of more abstraction. Now multiple function calls can be called by the same function without needing more logical flow structures and more jumping.

Scala:

```
 def abs(a:Double):Double =  {
      var b = a;
      if(a < 0){
        b = a * -1;
      }
      b;
    }

 def error(theoretical:Double,Actual:Double):Double = {
      var temp = 0.0;
      var theo = theoretical;
      var act = Actual;
      temp = theo - act;
      temp =  abs(temp);

      var temp1 = temp / theo;

       var temp2 = temp1 * 100;
       temp2;

   }


  def main(args: Array[String]): Unit = {

    var d = product(30,20);
    println(d);
    var h = product1(10,20);
    println(h);
    var g = abs(-10);
    println(g);
    var p = error(110,100);
    println(p);

  }
}
```

Syntatic Sugar:

The following code execution highlights some of the functionality of Scala and the use of syntax and other various attributes that make Scala so powerful. When instantiating objects the Scala programming language is not verbose, which means that an object can be built and used in about ten to fifty lines of code instead of Java code that takes close to double that.

```
class lets(var c: Int, var y: Int) {
   var x: Int = c
   var v: Int = y
   var k: Int = 0;

   def math(){
     k = x + v;
     print(k);
   }
}
```

The code below highlights the implementation of the object **lets** that we have built above. We can see the instantiation of an array of the lets object is does not require any declaration. Furthermore, we can see that use and functionality of Arrays is more powerful in the sense that when we call functions such as print Array, all of the individual data in the array will be printed to the screen. From this we can also declare objects as other types such as the var and set the value to null. Normally all of these variables would be casted to Null data types, but we declare them to be Strings. Now the array holds the values of String.

```
  var a,r,t,y,b = null:String;

    var list = Array(a,r,t,y,b);

     var list2 =(a,r,t,y,b);

    println(list(0));
    println(list2);

    var testingobjects = new lets(10,10);
```

```
var testingobjects2 = new lets(10,10);

var list3 =Array(testingobjects);
println(list3);

val list4 = List(testingobjects,testingobjects2);

print(list4(1).math);
print(list4(0).math);
```

Everything is an object

When observing the following code block there are a few very important differences that highlight the significance of everything being stored as an object to the programmer. The numbers that are in the function those being 1,2,3 and 5 are all stored as objects with no need to be called or instantiated.

```
def objects():Int ={

  1 + 2 + 3 ;

  5;
}
```

The following block of code runs and compiles and runs just like a normal Scala function. This function will still return the last line of code, when calling this function, the number 5 will be returned. However, you can change the return data type to a different data type and this function will still return the data type you changed it too. With this change you can now write functions with numbers and computations occurring without any casting and the operations still completing like normal.

Readability

Scala, in terms of readability is a very simplified version of Java. However, the function calls, that are pre-built and that are implemented by Scala data structures, are not intuitive and

can be confusing. With the ability of Scala to be both function and object-oriented programming styles there has been some issues arising when a programmer will incorporate both paradigms into a program. The bonuses of having the ability to do this is greatly speeding up the development time. With the ability to use both the programmer is able to cut several corners and save time. This negatively impacts the readability of Scala because it now forces the programmer to be able to not only understand both paradigms but implies that the reader will be able to seamlessly follow quick development time practices. This combination can produce code that is very difficult to read and understand however when a programmer sticks to one paradigm, that is when the program is all object oriented or all functional, Scala is much easier and quicker to read than an equivalent Java program because it takes many more lines of Java code to execute a Scala program. Furthermore, the ability to write Java programs and use Java libraries is a big bonus to the readability of Scala.

Writability

Scala is not only more simplistic than Java, but the duality of paradigms allows Scala code to be written by a programmer who can write functional or object-oriented libraries and applications. This means that the language does not force a programming style on the programmer as with other languages. For example, PHP allows no objects and thus forcing all PHP applications to be written in functional programming styles. This allows for very easy writability because now a programmer only has to learn the data structures and the syntactic sugar of the programming language which is already much more simplistic than the syntax of Java. The inferred casting is also a major advantage that Scala gets, now Scala can read in a variable as an object and we can pass these objects along to other parts of the application or

we can use only primitive data types, but all of this is inferred so a function can handle multiple data types. This allows for searching to be done over multiple data types instead of a unique search per data types.

Reusability

Since Scala can be written in both functional and object-oriented styles of programming a programmer can easily reuse part of a program in other applications, which is a major reason that Scala can lead to a quicker development time than an application written in C++ or C. However, when a given function needs adjustments, the reusability of the code can become more difficult. Since everything is an object in Scala, we will see that making changes is not always the simplest, especially if you are trying to change an object-oriented programming function into a functional one. The best practice for this is to use the object-oriented code as a base and rebuild the function under functional programming styles and methods. Since Scala has the ability to have Java code written and compiled with the language some Scala functions can also be reused in other Java applications as well. The fact that Scala can take in Java functions and code is a huge benefit. If we have part of an application already prebuilt in Java, we can just have an effective one-to-one ratio on reusing that code, that is that we can literally copy all of the code into the new application and lose no development time or spend time rebuilding something that is already prebuilt.

The inferred data type casting is a huge bonus in Scala reusability. What this means is that a function such as a search or a sort can easily be reused in a different application down

the line, regardless if the program must be sorting objects that were built by the programmer. Now instead of having to build a unique search and sort algorithm for every data type and data structure we can reuse one function for all of those scenarios.

Reliability

When it comes to the reliability of Scala code, it shares a lot of its reliability with the Java programming language. These features are the same for both languages, that is that both languages can run on any operating system thank to the Java Virtual Machine. Scala is also reliable with the application of automatic garbage collection. This allows Scala programs to not have to worry about memory leaks. Scala is more reliable than Java because Scala code can be written in the functional Paradigm. Functional programming induces less bugs into a program and this allows it to be more reliable than Java. From this point Scala's multiple paradigms allow for the reliability of both object-oriented programming styles and functional programming styles.

Evaluation

The scope in Scala is also nested just as Java's scope is nested. This allows for easy code readability and writability, since it shares scoping so closely with Java, I was able to identify and write classes in Scala. The allocation of memory is also very easy to understand and since Scala eliminated the need to write new it is very easy to write code. Scala is very similar to Java in many ways, but for me learning how to write functional programming is very easy since the writability of Scala is simple and not complex to read or write. The data types in Scala took me a

little bit to understand but now that I understand them the ability to create not only any primitive in Java but also the ability to create lists and access list like data types without having to instantiated them like in Java is very efficient and is less verbose.

Reflection

When studying this language there were many things that Java and Scala shared that allowed me to understand and develop in Scala code without having to spend too much time on understanding and learning the language. One of the difficulties I encountered when learning and implementing the algorithms in this language was a great difficulty understanding and implementing a functional program. Furthermore, using the Scala syntax for a loop and arrays is very different than that of Java. The really difficult part was to be able to understand and read a line of Scala code immediately after it followed a line of Java code. It was very easy to get confused when switching between two different language syntaxes. Understanding data types in Scala was really unique and interesting. Since everything is an object it was time consuming to understand how to call functions and treat functions like objects. Scala has some unique data types that other languages do not have such as a data type that can hold comments. Overall Scala is a very reliable and powerful language that is still used today by big companies for data manipulation. By using Java as the base for the language and then adding in the functional paradigm and shortening the syntax of the language Martin Odersky was able to

create a programming language that produces programs that are short, easy to read and powerful.

Citations

Odersky, M. (2018). *The Scala Language Specification Page*. [online] Scala-lang.org. Available at: https://www.scala-lang.org/docu/files/ScalaReference.pdf [Accessed 8 Oct. 2018].

Pankaj. "Scala Variables, Variable Scopes, Field Variables, Method Parameters Example." *JournalDev*, 21 Apr. 2015, www.journaldev.com/7581/scala-variables-variable-scopes-field-variables-method-parameters-example.

"Declarations." *News*, docs.scala-lang.org/style/declarations.html.

"Packages." *Scala Standard Library 2.12.1 - Scala.util.MurmurHash*, www.scala-lang.org/api/current/scala/collection/index.html.

Odersky, Martin. "Scala Spefication." Types, 2014, www.scala-lang.org/files/archive/spec/2.11/03-types.html.

Miller, Heather. "Types." News, 2016, docs.scala-lang.org/style/types.html.

https://www.scala-lang.org/files/archive/spec/2.11/03-types.html

https://docs.scala-lang.org/style/types.html