CMPT 383 Assignment 2
Jieung Park

Introduction

Scala is one of the popular high-level, multi-paradigm programming language that express programming patterns in a succinct and type-safe way. The name “Scala” stands for Scalable language designed to grow along with the demands of the users. Scala is a functional programming language in the sense that every function is a value and include features such as currying, immutability, lazy evaluation, and pattern matching. It is also a pure object-oriented programming language that defines every value as an object (no primitive data) and every operation is a method call.

Scala is influenced by many languages such as Java, Lisp, Haskell, Pizza, Scheme, etc. Importantly, Scala is highly influenced by Java because it was first released on the Java platform, modified for the .NET Framework (does not support .NET Framework anymore). Scala programs runs on Java platform (Java Virtual Machine), its source code can be compiled to Java bytecode and is compatible with Java applications. Although Scala is run on JVM, it has many features that are different from Java to provide better functionality/application than Java.

There exist many reasons for programmers to choose Scala over other languages. Since Scala is a high-level language that provides similar (useful) features and styles with Java, C, C++, it is easy to learn. Also, because Scala is a functional language and is statically typed, errors are caught at compile time which help avoid bugs in complex application (clear description of errors).

Type System

Built-in Types

Scala supports “Top Type” that provides Unified Type System, diagram provided as Figure 1. In the Top Type hierarchy, all types are introduced by “Any” which is a supertype of “AnyRef” and “AnyVal”. AnyRef is a reference types which is a supertype of all objects (java.lang.Object of Java) such as Classses. AnyVal is a value types such as Int, Long, Char, Boolean, etc. The advantage of having Any, the supertype of both AnyRef and AnyVal, is programmers can define a method that is compatible with both a value and a reference type or any other sub-types by taking Any as shown in Figure 2.

Scala supports “Bottom Types”, Nothing and Null. Null is a type of null reference that allows the null value in place of any reference type. Nothing is the “no value” type, does not have any methods or values (throws an exception as shown in Figure 3). Since Nothing and Null are bottom types, it is subtype of Any and subtypes of all other types in Scala. The property of Bottom Type provides advantages such as defining a generic empty base class. For example, Nil extends List(Nothing) which allows List to be any type: Int, String, Classes, etc.

Objects

Scala is a pure object-oriented language as every value is an object like Java, Python. Type and behavior of the object are classified by classes and traits and fields and methods (function) can be defined. Classes are declared with the keyword “class” and methods are declared with the keyword “def”.

Classes in Scala supports only a single inheritance which means a child class can only have one parent class unless it is the root of the class hierarchy, “Any”. In Scala, “extend” is used to inherit a trait as its parent, when it mixes in other traits, or when one trait is the child of another trait or class as shown in Figure 5. If a parent class is not extended, the default parent is AnyRef, a direct child root class, Any.

Scala allows two or more methods to have the same name as long as their type name, list of parameters and the return values are unique (signatures). However, the exception exists due to the type erasure (violation of method’s signatures) as shown in the Figure 6.

Type System Features

Scala is a strongly and statically typed language because all values are explicitly statically typed. Every variable and expression have a type that is already known at compile time and if invalid type is classified, then throws an error at compile time. Strongly statically typed language also restricts the possible values a variable can refer, or an expression can produce at run time. However, sub-typing allows programmers to safely use a value of type A where a value of type B is requested, if and only if A is sub-type of B. Otherwise, it will be checked at run-time and throw an exception when the actual values of the type parameters are referenced.

Additionally, the types are distinct from class because class that takes type as parameters can construct many different types. For example, List is a class in Scala, not a type. However, List can be a class of the types: List[Int], List[String], or List[Any_Type] which are defined as types as well as a List[T] which is also a type with a free type parameter that references one generic type T where “T” represents any type.

Memory Management

Garbage Collection

The advantage that Scala has is that it is run on Java Virtual Machine (JVM) and JVM provide garbage collection for memory management, therefore, programmers do not need to worry about manual memory management. The Figure 4 shows the memory structure of JVM. From the Figure 4, heap memory is where the objects are stored and where the garbage collection takes place.

All the function calls, storing primitive types, object types, function returns are handled in the stack memory. Once the main process is completed (when a function returns), the objects are no longer used because the pointers from the stack memory are removed unless a copy is created explicitly for more usage. Then, from the Figure 4, JVM automatically manages the

heap memory using the garbage collection process. The garbage collection process frees the memory used by orphan objects that are no longer used to make space for new objects. The garbage collector in JVM provide features such as memory allocation from OS, determining which parts of the allocated memory is still in use, and reclaiming the unused memory for reuse. If the program tries to allocate more memory on the heap that is defined to be no longer used, the program encounters out of memory errors.

Other Interesting Features

Immutability

Immutable objects and data structures are the first-class in Scala which means Scala always provide immutable collections by default. When a value is assigned to be an immutable, it cannot be reassigned. If a value is being reassigned as an immutable object, it will result in a compilation error. Since Scala is a hybrid language, it supports 'val', 'var', and availability of mutable collections as shown in the Figure 7, however, it is the best practice to use Scala as a functional programming language as much as possible for a thread-safe program. Immutability allows safe multi-threaded program because non-modifiable objects on each thread can run its job without worrying about the other threads. Therefore, Scala's immutability offers simple concurrent programming.

Lazy Evaluation

Lazy evaluation is a evaluation strategy where the evaluation of an expression is delayed until it is needed. Scala is a strict (not lazy) language by default, however, can be specified explicitly to be lazy (shown in the Figure 8). Lazy evaluation can be used to optimize CPU (performance) computation process from a never used and wasted complex evaluation (very costly). Additionally, it helps to resolve circular dependencies and allows modularity of code into parts. Although the lazy evaluation provides some pros, programmers must keep in mind that incorrect usage of lazy evaluation may lead to unpredictable performance and space complexity (all the delayed operations are stored).

Type Inference

In Scala, programmers are often not required to explicitly declare the type of an expression because the compiler can infer its type (Figure 9). The advantage of the type inference is that it contributes the code to be considered concise. Some cases when the compilers cannot automatically detect the missing type is when a return type is omitted while declaring methods and recursive methods. When a return type is omitted from a method, only a variable must be left at the end of the method instead of writing "return variable". Otherwise, the compiler will result in error because the return type is not specified, but the programmer is demanding for a return value.

Pattern Matching

Pattern matching in Scala provide useful abilities to match elements against complex patterns while allowing for more concise code. First, the pattern matching is not restricted to primitives and other types and can also match with tuples, lists, and classes (Figure 10). Second, allows

programmer to create more specific cases using pattern guards. The pattern guard is a simple Boolean expression (if statements) placed after the pattern (Figure 11). Lastly, match blocks of the pattern matching are expressions, not statements. The expressions in the matched cases are evaluated and returns a value whereas the statements do not return a value and mostly used to modify existing data.

Comparison with Other Languages

Scala vs. Java

Since Scala is run on Java Virtual Machine, a comparison with Java is better than the comparisons with other languages such as C, C++, or Go. Both Java and Scala are strongly, statically typed programming language that support object-oriented programming and functional programming. Scala may look similar to Java because it runs on JVM, but there exists major differences between them. First, Scala variables are immutable by default, whereas Java variables are mutable by default except Strings. Second, Scala support lazy evaluation if explicitly stated, but Java do not provide lazy evaluation. Third, “static” keyword or member only exists in Java. Moreover, Scala only support multiple inheritance using classes, but Java only supports multiple inheritance by interfaces. Lastly, the compilation and compiling process are slower with Scala because compiling process of source code into byte code is much slower with Scala compared to Java.

Scala vs. Haskell

Scala and Haskell have various features in common. The common features supported by both languages are lazy evaluation, type inference, immutability, and statically typed languages. Additionality, both languages are concise, safe, and support garbage collection to handle memory management. Aside from similarity, there also exists many differences between the two languages. First, Haskell is a pure functional language, but Scala is functional and pure object-oriented programming language. Mainly, Haskell has a compilation model of Glashow Haskell Compiler (GHC), whereas Scala has a similar model to Java (compiled into .class file and run on JVM). Since Scala run on JVM, Scala supports many libraries that are compatible with JVM, but Haskell does not support third-party tools or multiple libraries.

Conclusion

Scala supports many useful features that attract more programmers and companies in the industry to use the language. Scala is strongly, statically typed multi-paradigm programming language that support object-oriented and functional programming. The similarity with Java language allows Java programmers to approach the new language and learn faster. Scala also provides features that are similar to the other languages such as Python, C#, Go, Haskell and more. The useful features include type inference, lazy evaluation, immutability, and pattern matching which makes the language to become more concise and efficient. Therefore, even if the programmers are not familiar with Java or any other languages, Scala is very approachable language to new users.

Appendix

Figure 1: Unified Type System

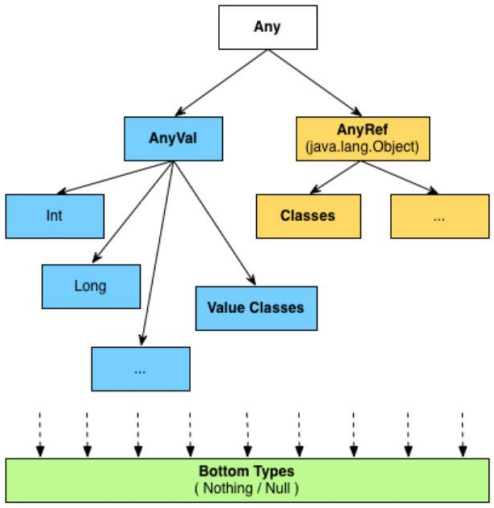

Figure 2: Example using Any

```
class Person
val anything = ArrayBuffer[Any]()   //Any allows compatibility with both AnyVal, AnyRef

val myInt = 42
anything += myInt                   //myInt (Int, AnyVal)
anything += new Person()            //Person (Class, AnyRef)
```

Figure 3: Bottom Type (Nothing)

```
val always_int_type =
    if (test) 42                            // Type – Int -> Any
    else throw new Exception("Wrong") // Type – Nothing -> Int -> Any
```

Figure 4: JVM Memory Structure

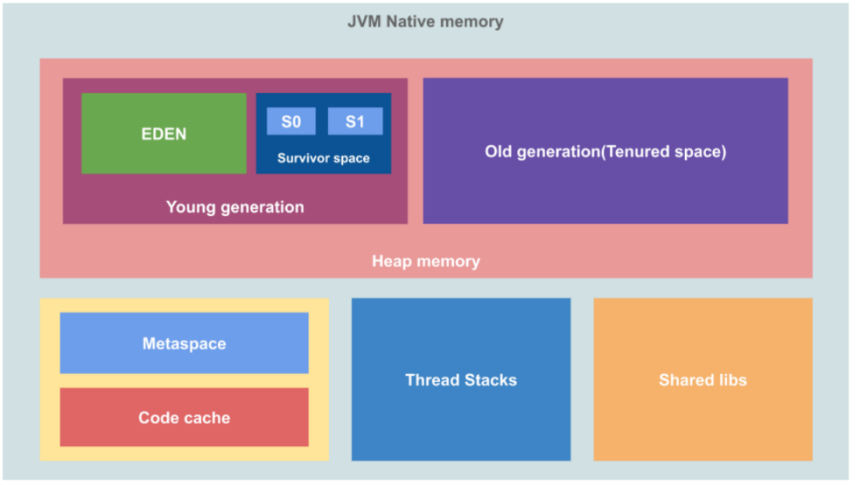

Figure 5: "extend" Classes

```
abstract class HelloWorld { … }
class hello(language: String) extends HelloWorld { … }
```

Figure 6: Same Name Methods and Type Erasure

```
Object Foo {
        //Allows same name method "test_list"
        //However, compilation error on second method because
        //    list.size.toString return the same type as the first method
        def test_list (list: List[String]) = list.toString
        def test_list (list: List[Int]) = list.size.toString
}
```

Figure 7: val, var, and mutable objects

```
var hello = "Hello!"
hello = "Hello World!" //var allows a variable to change (mutable)

val hi = "Hi!"
hi = "will it change? NOPE" //val will not allow a variable to be modified (immutable)

val b = MutableList[Int]() //Mutable collection to assign List as a mutable List.
```

Figure 8: Lazy Evaluation in Scala

```
val nums = List(1, 2, 3)
lazy val output = nums.map(1=> 1*2) //Expression is explicitly stated to be lazy
println(output)                     //The output evaluates now instead of previous line
```

Figure 9: Infer the type of an expression

```
val hello = "Hello World!" //The compiler can detect that hello is a String
def square(x: Int) = x * x   //The compiler can detect that the method square will return Int
```

Figure 10: Pattern Matching

```
tup match { case (x, y) => … }   //Pattern matching with tuples
xs match { case h :: t => …  }    //Pattern matching where xs is a List

case class Person(name: String, age: Int)
hello match {
        case Person(name, 42) => … //Pattern matching where Person is a class
}
```

Figure 11: Pattern Guard

```
tup match {
        case (x, y) if ( … ) => … // Pattern must satisfy the Boolean expression as well
}
```

Reference

https://www.geeksforgeeks.org/introduction-to-scala/

https://ktoso.github.io/scala-types-of-types/#unified-type-system-any-anyref-anyval

https://docs.scala-lang.org/glossary/index.html

https://deepu.tech/memory-management-in-jvm/

https://docs.huihoo.com/scala/programming-scala/ch05.html

https://stackoverflow.com/questions/52425954/how-to-append-elements-to-list-in-a-for-loop-scala

https://www.geeksforgeeks.org/scala-lazy-evaluation/

https://docs.scala-lang.org/tour/pattern-matching.html

https://www.guru99.com/scala-vs-java.html

https://www.geeksforgeeks.org/differnece-between-haskell-and-scala/