Student Name: Yi-Ting, Hsieh
Student ID: a1691807
Report option: Option 2
Course Code: COMP SCI 2204

Topic: Application in the software industry of functional programming language: Scala

What is Scala?
Scala is a general-purpose programming language providing support for functional programming and a strong static type system. Scala source code is intended to be compiled to Java bytecode, so that the resulting executable code runs on a Java virtual machine(JVM). Scala provides language interoperability with Java, so that libraries written in both languages may be referenced directly in Scala or Java code. Like Java, Scala is object-oriented programming language. Unlike Java, Scala has many features of functional programming language like Scheme and Haskell, including currying, type inference, immutability, lazy evaluation, and pattern matching. It also has an advanced type system supporting algebraic data types, convariance and contravariance, higher-order types, and anonymous types. Other featuers of Scala not present in Java include operator overloading, optional parameters, named parameters, and raw strings. Scala has the same compiling model as Java, C#, namely separate compiling and dynamic class loading, so that Scala code can call Java libraries.

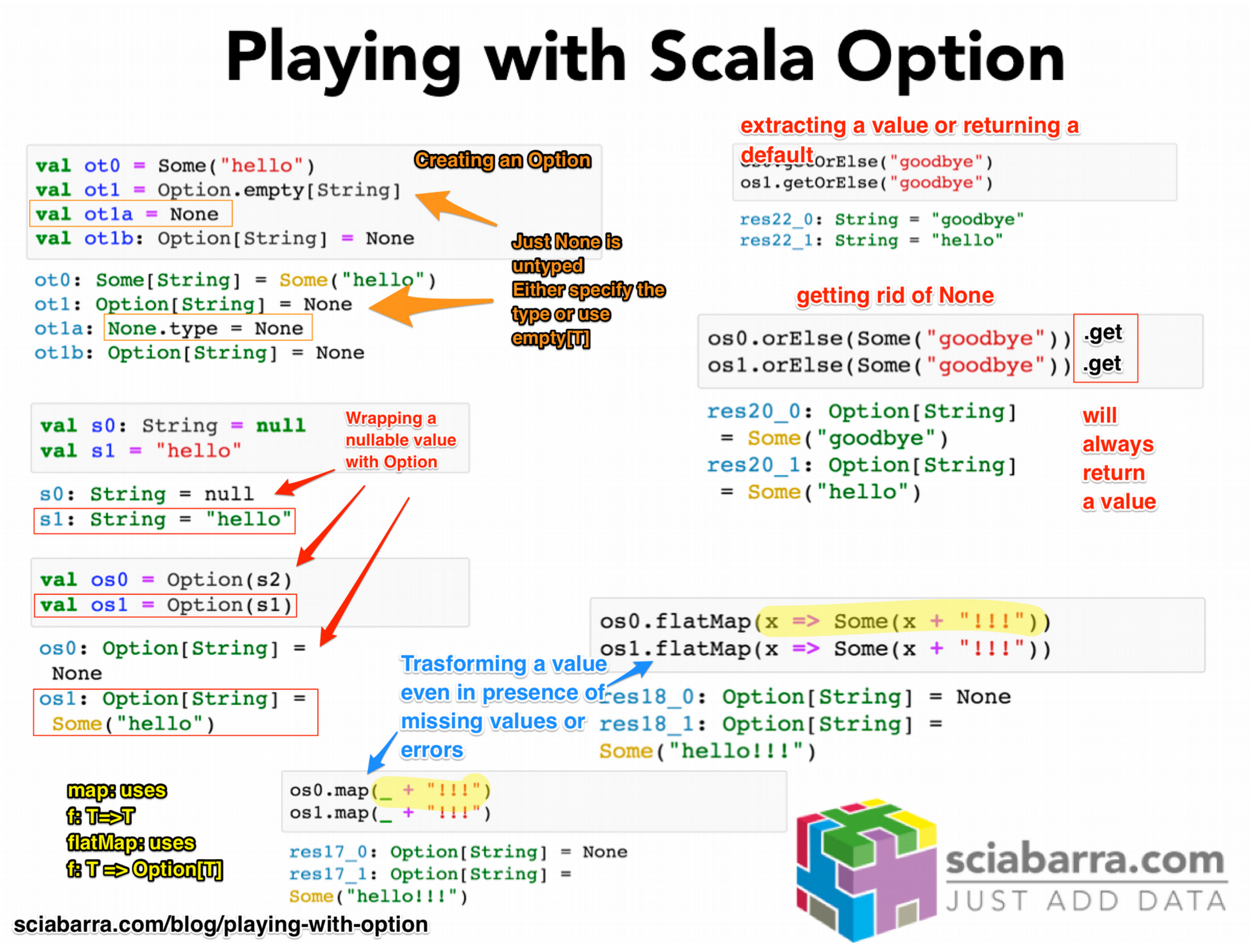
Scala
Java
toptal

```
def products = orders.flatMap(o => o.products)
```

```
public List<Product>getProducts(){
   List<Product> products = new ArrayList<Product>();
   for (Order order : order) {
      products.addAll(order.getProducts());
   }
   return products;
}
```

Scala’s operational characteristics are the same as Java’s. The Scala compiler generates byte code that is nearly identical to that generated by the Java compiler. In fact, Scala code can be decompiled to readable Java code, with the exeception of certain constructor operations. To the Java virtual machine, Scala code and Java code are indistinguishable. The only difference is on extra runtime library.

Scala adds a large number of features compared with Java, and has some fundamental difference in its underlying model of expressions and types, which make the language theoretically cleaner and eliminate several corner cases in Java. From the Scala perspective, this is practically implortant because several added features in Scala are also avilable in C#, which includes: Syntactic flexibility, unified type system, for-expressions, functional tendencies, type inference, anonymous functions, immutability, lazy(non-strict) evaluation, tail recursion, case classes and pattern matching, partial functions, object-oriented extensions, expressive type system and type enrichment. More than these, Scala standard library includes support for the actor model, in addition to the standard Java concurrency APIs. Where an actor is like a thread instance with a mailbox. Scala also comes with built-in support for data-parallel programming in the form of Parallel Collections. Besides actor



Playing with Scala Option

Creating an Option

```
val ot0 = Some("hello")
val ot1 = Option.empty[String]
val ot1a = None
val ot1b: Option[String] = None
```

```
ot0: Some[String] = Some("hello")
ot1: Option[String] = None
ot1a: None.type = None
ot1b: Option[String] = None
```

Just None is untyped
Either specify the type or use empty[T]

Wrapping a nullable value with Option

```
val s0: String = null
val s1 = "hello"
```

```
s0: String = null
s1: String = "hello"
```

```
val os0 = Option(s2)
val os1 = Option(s1)
```

```
os0: Option[String] =
 None
os1: Option[String] =
 Some("hello")
```

extracting a value or returning a default

```
os0.getOrElse("goodbye")
os1.getOrElse("goodbye")
```

```
res22_0: String = "goodbye"
res22_1: String = "hello"
```

getting rid of None

```
os0.orElse(Some("goodbye")) .get
os1.orElse(Some("goodbye")) .get
```

```
res20_0: Option[String]
 = Some("goodbye")
res20_1: Option[String]
 = Some("hello")
```

will always return a value

Trasforming a value even in presence of missing values or errors

```
os0.flatMap(x => Some(x + "!!!"))
os1.flatMap(x => Some(x + "!!!"))
```

```
res18_0: Option[String] = None
res18_1: Option[String] =
Some("hello!!!")
```

```
os0.map(_ + "!!!")
os1.map(_ + "!!!")
```

```
res17_0: Option[String] = None
res17_1: Option[String] =
Some("hello!!!")
```

map: uses
f: T=>T
flatMap: uses
f: T => Option[T]

sciabarra.com/blog/playing-with-option

sciabarra.com
JUST ADD DATA

support and data-parallelism, Scala also supports asynchronous programming with features and promises, software transactional memory, and event streams.

Who used Scala in industry?

LinkedIn

The Play Framework at LinkedIn

Yevgeniy Brikman
Staff Software Engineer
LinkedIn since 08-08

I'm excited to announce the next step in LinkedIn's service infrastructure: the Play Framework. Play is a modern web framework that combines the performance and reliability of Java and Scala, the power of reactive programming, and the productivity of full hot reload support.

We've been running Play 2.0 in production for several months and are now gradually rolling it out to more teams at LinkedIn. In this blog post, I'll talk about the benefits we've seen from Play, show a brief walk-through of the developer experience, and talk about our future plans.

The Scala programming language is used by many companies to develop their commercial softwares and production systems. For example, LinkedIn, was launched in 2003, is now the largest professional networking site in the world. Because the usages and information are quite large, their Search, Network and analytics team said the LinkedIn Social Graph represented by some 65+ million nodes, 680+ million edges and 250+ million request per day. They also said their People Search Engine receives around 15 million queries a day, or 250 queries per second with up to 100 tokens per query. Queries are satisfied by a scatter-gather approach across a large server farm which presented a challenge for efficient message routing and resource management reliable 24 and 7 opersation and easy application development.
Norbert is a framework written in Scala that makes it fast and easy to write asynchronous, cluster aware, message based cliend or server applicantions. Built on Apache ZooKeeper and Jboss Netty, Nobert provides out of the box support for notifications of cluster topology changes, application specific routing and load balancing, scatter or gather style APIs and partitioned workloads. Written by the SNA team at LinkedIn, Nobert is open sourced under the Apache License. While the most well-known open source cluster computing solution, written in in Scala, is Apache Spark and Apache kafka.
They use Scala because it makes concurrent support easier with Actors and promotes code reuse with traits and mixins. An Actor is like a thread instance with a mailbox. It can be created by system.actorof(), overriding the receive() method to receive message using the “!”.
Here is the example of how to use server as Actor that can receive messages and print them.

```
Val server = actor(new Act {
     become {
          case msg => println(“echo “ + msg)
     }
}}
server ! “hi”
```

More than that, because Scala has built-in support for parallel programming, so it makes it easier to write asynchronous programming. Here’re some examples of how parallel collections could improve performance.

```
Val urls = List("http://scala-
lang.org","https://github.com/scala/scala")

def fromURL(url: String) = scala.io.Source.fromURL(url)
  .getLines().mkString("\n")

val t = System.currentTimeMillis()
urls.par.map(fromURL(_))
println("time: " + (System.currentTimeMillis - t) + "ms")
```

It is really nice that the “cake pattern” could be implemented elegantly using self types to declare dependencies and have the compiler do the checking. The “killer feature” of Scala is the seamless integration of Java and Scala, making it low risk to introduce, and reduces the overhead to experiment. It makes software development more fun and a lot less frustrating.

Twitter

On the other hand, Twitter, one of the biggest community network stream, they also use Scala to build up their software as well. It provides an extremely popular real-time messaging service, which allows friends, family, and coworkers to stay in touch. Right now, it has more than 70 millioin members. It began as a Ruby on Rails application, and still uses Ruby on Rails to deliver most front-end web pages. But they replace some of the back-end Ruby services with applications running on JVM and written in Scala.

So, why they uses Scala to develop their API and framework?
Well, Twitter is a communcations service that allows people to share the post in 140 characters or less through mobile device, from a web browser, or from API, and are available for all the operating system,

mobile platform, or web platform. Alex Payne, the API lead at Twitter, said the core to their business is providing “open APIs” for everything you can do on the website. So all the functionality that’s available there for users is also available for developers to access programmatically. That is Twittwe in a nutshell.

At first, they use Ruby on Rails with a bunch of Ruby daemons doing asynchronous processing on backend. Over time, they founod that although Rails works fine for doing front-end web development, for doing heavy weight back-end processing, Rails had some performance limitations at runtime because Ruby language lacks some things that contribute to reliable, high performance code, which is something they are really care about when they are groing as a business. They want not only the code is correct, but also it should easy to maintain. Ruby, like many scripting languages, has trouble being an environment for long lived processes, but the JVM is very good at that, because it has been optimized for that over decades. And Scala provides a basis for writing long-lived servers.

In addition, Ruby doesn’t have optional type annotation, and they find that really useful in Scala, to be able to specify the type information. Besides, Ruby didn’t have good thread support then. It’s getting better, but when they were writing their servers, green threads were the only thing availble, however, it doesn’t use the actual operating system’s kernel threads. It emulate threads by periodically stopping what they are currently doing and checking whether another thread wants to run. That means Ruby was emulating threads within a single core or a processor. But they want to run on multi-core servers that don’t have an infinite amount of memory. Also, because Ruby’s garbage collector was not quite as good as Java’s, each process took a lot of memory, so they couldn’t run very many Ruby daemon processes on a single machine without consuming large amount of memory.

Another reason is that they like the flexibility of theh language, like a full featured language. They find it quite fun to code in. And that is why they switch to Scala. More than that, it’s fast, it’s runs on a JVM and can borrow from Java. And the flexible syntax makes it easy to write readable, maintainable code. Traits for cross-cutting modularity and lots of Object-oriented programming goodness. Because Scala is a powerful funnctional programming language, you can make use of some of it’s features like mapping, filtering, folding, currying and so much more. You can also define some lazy values or pattern matching to help your works. And because they need to handle some concurrency issues, while using Scala has a good advantage on it, that is you can do actor concurrency, which is message passing, as what Erlang can do. Or, you can use threads, some java tools, or use other JVM concurrency frameworks. And that’s basically why they choose Scala to build up their software.

So what did they do Scala? They built up services, isolated components, talked to the rest of the system via Thrift, inpendently tested for correctness, and had some custom operational properties. Here’re some Scala services at Twitter:

- Kestrel: queueing
- Flock(and Gizzard): social graph store
- Hawkwind: people search
- HoseBird: streaming API
- …

More than above services, they also built their own libraries through Scala, which are more reusable chunks of code that could be shared between projects. And they open source their libraries as well.

Criticism:
However, even with lots feature above, Scala is quite a hard language for many people. Most of people, even from industry, stated their decision to minimise dependence on Scala due to the learning curve for the new team members and incompatibility from one version of the Scala compiler to the next.