Scala: What’s the deal?

CMPT 383 Assignment 2

Alec Pershick

Scala, short for Scalable Language, is a hybrid functional programming language. It was created by Martin Odersky. Scala smoothly integrates the features of object-oriented and functional languages. Scala is compiled to run on the Java Virtual Machine. Many existing companies, who depend on Java for business critical applications, are turning to Scala to boost their development productivity, applications scalability and overall reliability.

Here we have presented a few points that makes Scala the first choice of application developers.

**Scala is object-oriented**

Scala is a pure object-oriented language in the sense that every value is an object. Types and behavior of objects are described by classes and traits which will be explained in subsequent chapters.

Classes are extended by **subclassing** and a flexible **mixin-based composition** mechanism as a clean replacement for multiple inheritance.

**Scala is functional**

Scala is also a functional language in the sense that every function is a value and every value is an object so ultimately every function is an object.

Scala provides a lightweight syntax for defining **anonymous functions**, it supports **higher-order functions**, it allows functions to be **nested**, and supports **currying**. These concepts will be explained in subsequent chapters.

**Scala is statically typed**

Scala, unlike some of the other statically typed languages (C, Pascal, Rust, etc.), does not expect you to provide redundant type information. You don't have to specify a type in most cases, and you certainly don't have to repeat it.

**Scala runs on the JVM**

Scala is compiled into Java Byte Code which is executed by the Java Virtual Machine (JVM). This means that Scala and Java have a common runtime platform. You can easily move from Java to Scala.

The Scala compiler compiles your Scala code into Java Byte Code, which can then be executed by the '**scala**' command. The '**scala**' command is similar to the **java** command, in that it executes your compiled Scala code.

**Scala can Execute Java Code**

Scala enables you to use all the classes of the Java SDK and also your own custom Java classes, or your favorite Java open source projects.

**Scala can do Concurrent & Synchronize processing**

Scala allows you to express general programming patterns in an effective way. It reduces the number of lines and helps the programmer to code in a type-safe way. It allows you to write codes in an immutable manner, which makes it easy to apply concurrency and parallelism (Synchronize).

**Scala vs Java**

Scala has a set of features that completely differ from Java. Some of these are −

* All types are objects
* Type inference
* Nested Functions
* Functions are objects
* Domain specific language (DSL) support
* Traits
* Closures
* Concurrency support inspired by Erlang
* **Object** − Objects have states and behaviors. An object is an instance of a class. Example − A dog has states - color, name, breed as well as behaviors - wagging, barking, and eating.
* **Class**− A class can be defined as a template/blueprint that describes the behaviors/states that are related to the class.
* **Methods** − A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
* **Fields** − Each object has its unique set of instance variables, which are called fields. An object's state is created by the values assigned to these fields.
* **Closure** − A **closure** is a function, whose return value depends on the value of one or more variables declared outside this function.
* **Traits** − A trait encapsulates method and field definitions, which can then be reused by mixing them into classes. Traits are used to define object types by specifying the signature of the supported methods.
* **Case Sensitivity** − Scala is case-sensitive, which means identifier **Hello** and **hello** would have different meaning in Scala.
* **Class Names** − For all class names, the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.

**Example** − class MyFirstScalaClass.

* **Method Names** − All method names should start with a Lower Case letter. If multiple words are used to form the name of the method, then each inner word's first letter should be in Upper Case.

**Example** − def myMethodName()

* **Program File Name** − Name of the program file should exactly match the object name. When saving the file you should save it using the object name (Remember Scala is case-sensitive) and append ‘**.scala**’ to the end of the name. (If the file name and the object name do not match your program will not compile).

**Example** − Assume 'HelloWorld' is the object name. Then the file should be saved as 'HelloWorld.scala'.

* **def main(args: Array[String])** − Scala program processing starts from the main() method which is a mandatory part of every Scala Program.
* **Declaring variables**:
* **Var**: regular mutable variable (able to change value)
* **Val**: immutable variable (const?)
* **Types**: Statically declared as var myVar :Int;
* If you don’t declare it, can be type inferenced

**Variable Scope**

Variables in Scala can have three different scopes depending on the place where they are being used. They can exist as fields, as method parameters and as local variables. Below are the details about each type of scope.

**Fields**

Fields are variables that belong to an object. The fields are accessible from inside every method in the object. Fields can also be accessible outside the object depending on what access modifiers the field is declared with. Object fields can be both mutable and immutable types and can be defined using either **var** or **val**.

**Method Parameters**

Method parameters are variables, which are used to pass the value inside a method, when the method is called. Method parameters are only accessible from inside the method but the objects passed in may be accessible from the outside, if you have a reference to the object from outside the method. Method parameters are always immutable which are defined by **val**keyword.

**Local Variables**

Local variables are variables declared inside a method. Local variables are only accessible from inside the method, but the objects you create may escape the method if you return them from the method. Local variables can be both mutable and immutable types and can be defined using either **var** or **val**.

Pros/Cons

Pros

- ‘Scala’ is a portmanteau of ‘scalable’ and ‘language’, as we’ve discussed. So, scalability is definitely one of biggest Scala advantages. This means we can use it to build highly concurrent, fault-tolerant systems. For this, we can use the multiagent concurrency model like in Language Erlang in Akka.

- Scala can be a great choice for data analytics with support from tools like Apache Spark, among others. A lot of huge companies make use of Scala for their products and services. Great for machine learning companies.

- Inherently immutable objects. Scala’s programming language reduces many thread-safety concerns that spring up in traditional Java applications.

- Highly scalable solutions like Klout, LinkedIn, Amazon, Blizzard, Coursera, Twitter are using Scala.

- Lazy evaluation like Haskell

Cons

- Since it runs on the JVM, it has no true tail-recursive optimization. As a workaround, you can use the @tailrec annotation for partial benefits.

- Hard to learn. Quite a bit more complicated than Java it seems.

Examples

First, the Java version:

public class User {

private String name;

private List<Order> orders;

public User() {

orders = new ArrayList<Order>();

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public List<Order> getOrders() {

return orders;

}

public void setOrders(List<Order> orders) {

this.orders = orders;

}

}

public class Order {

private int id;

private List<Product> products;

public Order() {

products = new ArrayList<Product>();

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public List<Product> getProducts() {

return products;

}

public void setProducts(List<Product> products) {

this.products = products;

}

}

public class Product {

private int id;

private String category;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getCategory() {

return category;

}

public void setCategory(String category) {

this.category = category;

}

}

Phew. Lotta code.

Now the Scala version:

class User {

var name: String = \_

var orders: List[Order] = Nil

}

class Order {

var id: Int = \_

var products: List[Product] = Nil

}

class Product {

var id: Int = \_

var category: String = \_

}