Chapter 1
Introduction

1.1 Introduction

This book is intended as an introduction to Scala for computer science students or those actively involved in the software industry. It assumes some familiarity with standard computing concepts, such as the idea of compiling a program and executing this compiled form, and with the basics of procedural language concepts such as variables and allocation of values to variables. However, the early chapters of the book do not assume any familiarity with Object Orientation nor functional programming. They also step through other concepts with which the reader may not be familiar (such as list processing). From this background, it provides a practical introduction to object and functional technology using Scala, one of the newest and most interesting programming languages available.

This book introduces a variety of concepts through practical experience. It also tries to take you beyond the level of the language syntax to the philosophy and practice of Object-Oriented development and functional programming.

In the remainder of this chapter, we will consider what Scala is, why you should be interested in Scala and whether this book is for you.

1.2 What Is Scala?

Scala is a new programming language developed by Martin Odersky and his team at the EPFL (Ecole Polythenique Fererale de Lausanne, Lausanne, Switzerland) and now supported by Lightbend Inc. (previously known as Typesafe).

The name Scala is derived from Sca(lable) La(nguage) and is a multi-paradigm language, incorporating Object-Oriented approaches with functional programming.

What does this mean in practice? It means that you can write applications as pure Object-Oriented solutions using Classes, Objects and Traits. You can exploit

J. Hunt, *A Beginner's Guide to Scala, Object Orientation and Functional Programming*, https://doi.org/10.1007/978-3-319-75771-1_1

inheritance, polymorphism and abstraction and encapsulation techniques. In this respect, Scala is very much like any other Object-Oriented language (such as Java, C# or C++). However, you can also develop solutions using purely functional programming principles in a similar manner to languages such as Haskell or Clojure. In such an approach, programs are written purely in terms of functions that take inputs and generate outputs without any side effects.

Scala though is different in that it is a hybrid programming language. That is, it is possible to combine the best of both worlds when creating a software system. You can therefore exploit Object-Oriented principles to structure your solution but integrate functional aspects when appropriate. Whilst this approach is not unique (the Common Lisp Object Systems did something similar in the 1980s), it is certainly bringing functional programming to the mainstream and integrating it within an environment that can execute almost anywhere.

Of course Scala has not been developed in isolation and has been influenced by many of these and other languages. The influences on the Scala language are shown in Fig. 1.1.

1.3 Why Scala?

This of course raises the question why Scala and why now? There are a number of reasons why Scala should be a language that is given serious consideration by any development project. We have already mentioned that fact that it coherently brings together two very powerful programming paradigms that combined can allow very elegant, concise and maintainable systems to be created. However, there are other reasons why Scala is of interest. The first is that Scala can be compiled to Java Byte

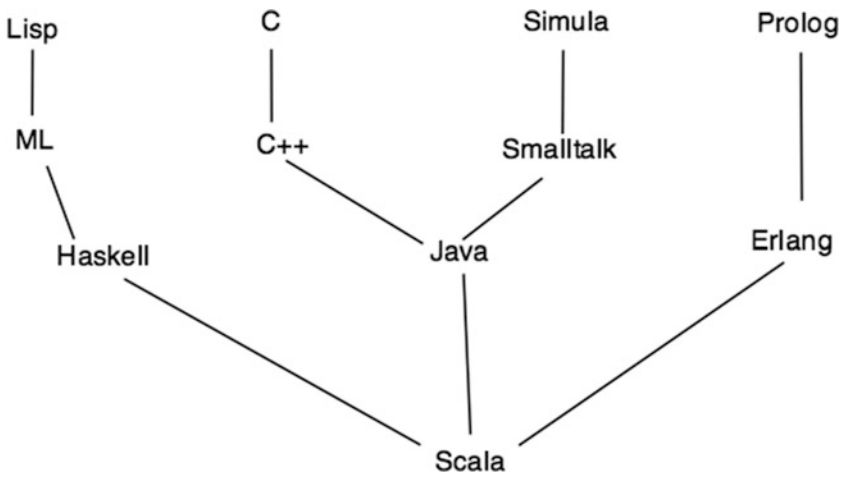

Fig. 1.1 Scala Genealogy

Codes. This means that a Scala system can run on any environment that supports the Java Virtual Machine (or JVM). There are already several languages that compile to Java Byte Codes. This list includes Java but also extends to Ada, JavaScript, Python, Ruby, Tcl and Prolog. Scala is just another such language. However, this has the additional advantage that Scala can also be integrated with any existing Java code base that a project may have. It also allows Scala to exploit the huge library of Java projects available both for free and for commercial use.

Another reason to consider Scala is that one of the design goals for the Scala development team was to create:

> A scalable language suitable for the construction of component based software within highly concurrent environments.

This means that it has several features integrated into it that support large software developments. For example, the Actor model of concurrency greatly simplifies the development of concurrent applications. In addition, the syntax reduces the amount of code that must be written by a developer (at least compared with Java). This is because it avoids a lot of the boilerplate code that any Java developer will be familiar with.

To summarise then, the following points can be made that Scala:

- Provides Object-Oriented concepts including classes, objects, inheritance and abstraction.
- Extends these (at least with reference to pre Java 8) to include Traits which represent data and behaviour that can be *mixed* into classes and objects.
- Includes functional concepts, such as functions as first-class entities in the language, as well as concepts such as Partially Applied functions and Currying which allow new functions to be constructed from existing functions.
- Uses statically typed variables and constants with type inference used wherever possible to avoid unnecessary repetition.
- Has interoperability (mostly) with Java.

To return the question of 'Why now?'—now is a good time to be learning about Scala. At the time of writing Scala has been in commercial use (at least to my knowledge) for seven years and has stabilised and addressed some of the concerns that commercial development projects had about early version of Scala.

1.4 Java to Scala Quick Comparison

As a comparison, for those who are familiar with Java, the following two listings compare and contrast equivalent code defined in Java and Scala. Do not worry at this point too much about the syntax; it is more for illustration than the specifics of either Java or Scala at this point.

Here is the Java class:

```
class Person {
    private String firstName;
    private String lastName;
    private int   age;



  public Person(String firstName, String
lastName, int age) {
      this.firstName = firstName;
      this.lastName  = lastName;
      this.age  = age;
  }


  public void setFirstName(String firstName) {
       this.firstName = firstName; }
  public void String getFirstName() {
this.firstName; }
  public void setLastName(String lastName) {
    this.lastName = lastName; }
  public void String getLastName() {
this.lastName; }
  public void setAge(int age) { this.age = age;
}
  public void int getAge() { return this.age; }
}
```

And here is the equivalent Scala class:

```
class Person(var firstName: String, var lastName: String, var
age: Int)
```

As you can see, the Scala version is much shorter but actually captures the same concepts. The core concepts here are that:

> A Person has three properties `firstName`, `lastName` and `age`. These properties are readable and writable. When a new `Person` is constructed, you must provide values for the `firstName`, `lastName` and the age.

Both listings implement these concepts; however, in Java's case it has no concept of a property and thus we must define how the data is held internally to a Person and how it can be accessed or updated via various getter and setter methods. In contrast, Scala has a concept of properties and thus we do not need to write the update and access style methods. Instead, we need to decide if they are read-only (known as *vals)* or read–write properties (as indicated by the keyword *var*).

1.5 Scala Versions

There have been several *significant* Scala versions over recent years. It is useful to be aware of these so that if you are looking at blogs or articles on the Web you can see which ones are relevant to you. Previous significant versions have been Scala 2.9, 2.10 and 2.11. This book focuses on Scala 2.12 which is the current release at the time of writing.

1.6 Is This Book for You?

This book does not assume a great deal of programming experience. However, it is not a basic introduction to programming. Instead, it is aimed at those with little programming and no functional or Object-Oriented experience. It does introduce concepts such as lists, data collections, `for` loops and conditional control statements. However, it assumes a basic understanding of how programs work, of what a programming stack might be, that memory must be allocated for data, etc.

It can also be used to develop some basic knowledge of programming into a more in-depth knowledge of a particular technology. It could also be used to support an introduction to programming course.

1.7 Approach Taken by This Book

In general, the book takes a very "hands-on" approach to the whole subject and assumes that you will implement the examples as you progress. It supports this through many examples that take you through how to use the Scala IDE to support what you are doing as well as providing complete code examples with indications of the expected outcomes. Unlike many books on Scala, the focus is on using Scala within an IDE and constructing simple applications rather than using the interactive Scala interpreter. In addition, all the samples used in the book are available from Springer to be downloaded and used in your own IDEs.

References Haskell

- http://www.haskell.org/

Clojure

- http://clojure.org/

Common Lisp Object System

- Sonya E. Keene, Object-Oriented Programming in Common LISP: A Programmer's Guide to CLOS, Pub. Addison Wesley, (Jan 1989) 0201175894.

- List of JVM Languages http://en.wikipedia.org/wiki/List_of_JVM_languages

The Scala programming language home page

- see http://www.scala-lang.org

The Scala mailing list

- see http://listes.epfl.ch/cgi-bin/doc_en?liste=scala

The Scala wiki

- see http://scala.sygneca.com/

Lightbend Inc.

- https://www.lightbend.com/