Scala: An Introduction

By Arianna Jakositz

CSCI 5448 Object-Oriented Analysis and Design
University of Colorado at Boulder - Fall 2012
Outline

• About Scala
  • What Is It?
  • History
  • Syntax
• Object-Oriented Features of Scala
  • Inheritance
  • Polymorphism
  • Encapsulation
• Why Use Scala?
  • Scala vs. Java
  • Who Is Using Scala?
• Resources To Learn More
What’s Ahead:

✓ What Is Scala?
✓ A Brief History
✓ A Look At The Syntax
What Is Scala?

- Scala is a mixed programming language, combining features of functional and object-oriented programming aspects
  - Functional because every function is a value
  - Object-oriented because every value is an object
  - Allows users to program effectively in either style according to their needs and desires

- Scala runs on the Java Virtual Machine (VM)
  - Java and Scala libraries can directly call each other
  - Byte code compatibility between the two languages

- Name comes from the combination of “Scalable” and “Language”
  - Scala is meant to grow (scalability) as user needs evolve
History of Scala

• Development began in 2001 by Martin Odersky at the École Polytechnique Fédérale de Lausanne (EPFL) in Lausanne, Switzerland

• Created with the intent of improving many of the drawbacks associated with Java.

• Martin Odersky initially worked with developers at Sun to add Generics functionality, and improve the compiler, for the Java language

• Efforts truly began in the mid-90s on a language known as Pizza, that was eventually scrapped in favor of Scala
A Look At The Syntax of Scala

Changing the syntax of Java into that of Scala led to the following modifications:

- Built-in type inference, where the compiler deduces type from a variables initialization
- Implicit parameters, methods, and conversions
- Methods can be used like infix operators (+, -, *, etc.), i.e.,
  \[ \text{a.function}(b) \]
  can be written as
  \[ a \text{ function } b \]
Scala Syntax (cont’d)

- Variables are declared by the `var` keyword: [Note: Semicolons are NOT always required]
  ```scala
  var x = 5
  ```

- Constants are declared by the `val` keyword:
  ```scala
  val y = 10
  ```

- Types are explicitly declared as follows:
  ```scala
  var x: Int = 15
  ```

- Functions are declared by the `def` keyword: [Note: Semicolons are NOT always required]
  ```scala
  def function(x: Int, y: Double, z: Any) {...}
  ```

- Return values of functions are the value of the last line of code contained within the curly braces. If a function is one line, curly braces are not required.
Scala Syntax (cont’d)

- Classes are declared with the `class` keyword:
  
  ```scala
  class SomeClass() {...}
  ```

- Constructors are not separate methods, but the entire class bodies. Thus, any input parameters to a class can be used right away outside of any explicitly defined method. Overloading the “constructor” is done using calls to `this(...)`, where “...” may refer to any number of input parameters, including 0.

- Singleton objects are created by declaring with the `object` keyword instead of the `class` keyword:
  
  ```scala
  object Singleton() {...}
  ```

- There is no static modifier in Scala!

- Abstract classes are declared with the `abstract` keyword
## What’s Ahead:

- **Inheritance**
- **Polymorphism**
- **Encapsulation**
Inheritance and Polymorphism In Scala

• Every class inherits from one other class in Scala. When not specifically declared, this class is scala.AnyRef

• Inheritance is denoted by the extends keyword:
  ```scala
class Subclass() extends Superclass {...}
```

• Overridden methods must be explicitly denoted with the override modifier:
  ```scala
  override def toString() = {...}
  ```

• Scala classes can also implement from multiple traits, which are like interfaces in Java but can contain code. The implementation of traits is also denoted by the extends keyword. A trait is declared with the trait keyword:
  ```scala
  trait TraitDefinition() {...}
  ```
In order to keep a class from being subclassed, the `final` modifier may be added to its declaration:

```scala
final class EndOfTheLine() {... }
```

Similarly to other object-oriented languages, a subclass can be used in place of an expected superclass in Scala – that is, polymorphism is fully supported in Scala.
Encapsulation In Scala

• By default, all values in Scala are public

• The private modifier can be used to explicitly declare something private

• Since every function is a value, and every value is an object in Scala, every value has its own inherent getter and setter. While this initially appears to be a direct access to a class’s fields, the fields can be defined as follows to encourage further encapsulation:
  
  ```scala
  private var _field = 1
  ```
  
  // Getter:
  ```scala
def field = _field
  ```
  
  // Setter
  ```scala
def field_=(value: Int): Unit = _field = value
  ```
  
  • The underscore character allows the setter to essentially be “field = “
WHY USE SCALA?

What’s Ahead:

✓ Scala vs. Java
✓ Who Is Using Scala?
Scala Compared to Java

• Code written in Scala tends to be two to three times shorter than code written in Java.

• Scala allows for the following unsupported features in Java:
  • No backward-compatibility constraints to limit the functional programming capabilities
  • Unchecked exceptions
  • Operator overloading
  • No distinction between primitive types and all other types – all values are objects
  • No static modifier in Scala
Who Is Using Scala

*Scala is gaining traction in industry, and being used for many different purposes in many different companies:*

- **Twitter** -- The primary messaging queue transitioned from Ruby to Scala, which improved performance and reduced lines of code.

- **LinkedIn** – Implemented the Norbert library in Scala, which “provides easy cluster management and workload distribution” [More]

- **Sony Pictures Imageworks** – Scala Migrations library for database schema management [More]

- **Xerox** – ICE Project, which deals with invitations to Xerox showrooms in the United Kingdom

- …And much more
Resources To Learn More

• Official Scala website: http://www.scala-lang.org/

• Scala For Java Refugees (blog to help Java developers to transition!)
  http://www.codecommit.com/blog/scala/roundup-scala-for-java-refugees

• *Scala For The Impatient* by Cay S. Horstmann; 2012, 1st edition
  (introduction to Scala for experienced developers)

• Another Scala Reference
  http://www.simplyscala.com/