# Scala

Meeting of Functional and Object Oriented Languages

Jessica Ebert

CSCI 5448, Fall 2011

# Introduction and Summary

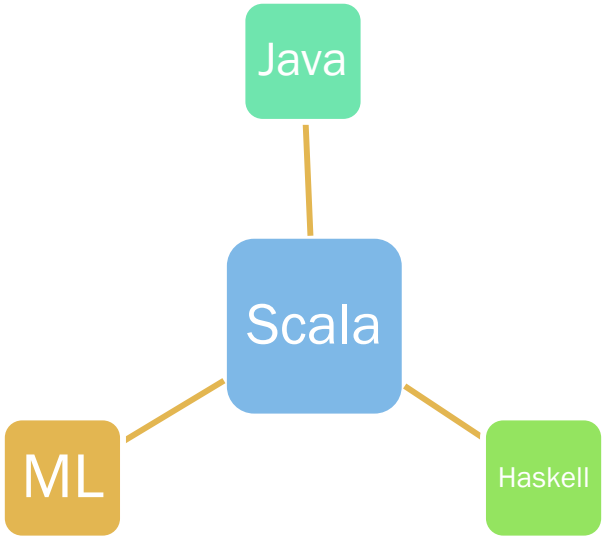
- Scala is a hybrid language that uses ideas from both the functional and object oriented programming ideas
- It arises from the intention of creating a programming language that removed a lot of the hassle from Java and was user friendly
- The mantra of Scala is to create highly modular elements of a program
  - There are very few hard-written links or references to other objects or instances in Scala. Most modules can be easily ported to be reused.
- Scala was runs using the Java Virtual Machine and is compatible with every Java library.
- Its relation to Java makes it very easy to switch to.
- Scala is not just an academic language, but has made impressive inroads into the corporate world.

# Naming and Intentions

- Scala was designed and named from the idea that a program should contain components that are reusable and be easily scalable.
  - The need for reusable and scalable is  some of the main rational for adding the functional language elements
- Scala's feature's are generally directed for the creation of modular programs.
  - Minimizes hard links to specific components
  - Allows for abstraction over the required support and services required.
- One of the key notions of Scala is "immutability first"
  - Having immutable code makes it much easier to create modular code.

# Roots

- Scala is a hybrid language
- Combines object-oriented and functional programming
  - Blends both worlds seamlessly
  - Enables the programmer to use the Java open-source availability
  - Uses the principles from Haskell and ML for functional programming



# Short History of Scala

- Designed in 2001 by Martin Odersky
  - Along with his group at ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE (EPFL)
- Odersky worked on programming language fundamentals and became fascinated with the idea of simplifying and improving Java.
- In 1995 he worked with Philip Wadler to write a functional language that compiled to Java Bytecodes,
  - This was called Pizza.
- Next he developed GJ, a new javac compiler, then Java Generics.
- He designed Scala with the idea of making a user friendly language that combined functional and Object oriented programming that eliminated the restrictions in Java.
- Scala was first publicly released in 2003, since then it has been fully supported and gaining popularity

(Scala, 2008)

# Use of other Resources

- Anything that looks like a language level keyword for nearly any object oriented language is probably defined in the library.
- Because Scala compiles to Java bytecode, it enables you to use the plentiful Java resources.
  - You can even use the libraries that are available for Java
- The Java bytecode basis has an ever further benefit, it can be almost seamlessly integrated with any system that can be with Java.

# Fibonacci With Haskell

```
module Main where
 import System.Environment
 fibo = 1 : 1 : zipWith (+) fibo (tail fibo)
main = do
        args <- getArgs
        print (fibo !! (read(args!!0)-1))
```

# Fibonacci in Java

```
public class Fibonacci {
    public static long fib(int n) {
        if (n <= 1) return n;
        else return fib(n-1) + fib(n-2); }
     public static void main(String[] args) {
         int N = Integer.parseInt(args[0]);
        for (int i = 1; i <= N; i++) System.out.println(i + ": " +
fib(i));
    }
}
```

# Fibonacci Sequence in Scala

```
import scala.math.BigInt
lazy val fibs: Stream[BigInt] = BigInt(0) #::
          BigInt(1) #::
          fibs.zip(fibs.tail).map { n => n._1 + n._2 }
```

- While Scala infers types, this recursive value needs to be defined so that Scala knows to run it on a stream and not another unrelated type.
- The #:: is searched by Scala to find a conversion type that has the #:: command
  - This is why Stream was defined earlier, so that Scala knew to find a conversion type that had the method.
  - Similarly to java, there is the standard main argument
- The ability to infer a conversion type is one of the best features of Scala. However, it can be dangerous too.

# Hello Scala: Understanding Scala More.

```
object HelloWorld {
    def main(args: Array[String]) {
        println("Hello, world!")
    }
}
```

- It is not necessary to declare a return type, even if this were to return something
  - The last line of an application, if it is a value, is assumed to be the return object.
  - Scala can also return methods and functions
- The main method is not declared as static because there are no static fields or methods.
  - Instead they are declared as singleton objects.
- The class, unlike Java, is declared as an object.
  - Scala is a pure-object oriented language because everything is an object.
  - This enables the programmer to manipulate functions as a value.

# Singleton Objects

- Singleton object
  - A class with a single instance
  - This declares both a class and an instance at once without having to instantiate it.
- Limitations
  - A new instance of a singleton object can not be created.
    - However, a class of the same name may be created that allows for the creation of and the compiler will deduce that the class is being called.
      - This will allow you to create a regular object of the same type, depending on how the instance in the class is setup.
- Static methods or fields do not exist in Scala.

# Functions as Objects

- Functions are treated as objects in Scala
  - This is also known as First-Class functions.
- These functions have the literal syntax of an object.
- Because functions are objects, it opens up a world of possibilities.
- Functions can be passed as arguments
  - Stored as variables
  - Be returned from other functions
- This ability is the cornerstone of functional programming and one of the main demonstrations of Scala's dual nature.

# Classes in Scala

- There are classes in Scala, not just objects.
- They differ a bit from Java's classes, although they are declared similarly.
- In Scala, Classes can have parameters.
- As a reminder, return  types do not have to be declared explicitly.
  - They will be inferred by the compiler.
  - The compiler looks at the right-hand side of the method and determines what the return type is.
  - Unfortunately, it can not always do this.
  - The general rule is to try to omit type declarations and see if the compiler complains.

# Classes and Inheritance

- Every class in Scala inherits from a super-class
- If a super-class is not defined, the class will have an implicit super-class
  - The default implicit super-class is scala.AnyRef.
- It is possible to override any inherited method.
  - This is one of the few times when something must be explicitly specified.
    - The **override** modifier must be specified.

# Case Classes

- Uses pattern matching
  - Pattern matching in Scala allows the user to match any type of data with a first-match implementation
- Constructor parameters are inherently associated with an accessor/mutator method pair
- Has automatic conversion of the constructor parameters of the case class into fields.
  - Used mainly for the purpose of isolating the data into an easily accessible manner.

# Scala as an Object Oriented Language

- In addition to previous examples, Scala has many of the qualities of an Object oriented language.
- Scala can create stand-alone objects
- It has both classes and traits, which in its object oriented format are described by classes and traits.
- Typical OO abstraction is done with abstract and sub classes.

# Benefits Over Java

- First and foremost is the functional language side of Scala
  - This helps to eliminate mutable states
  - It also helps to write concise solutions.
- Scala uses a concise syntax.
  - It is less verbose than Java, which makes it easier to debug as well as to review.
- Helps to create more reliable code through the powerful type system

# Scala as a Functional Language

- Functions are first class values
  - A function has all the functional properties of any of the default and more “typical” types.
  - You can treat a function in the same way that you would treat any of the built-in types.
- Lazy evaluation
- Pattern Matching
- Currying Functions
  - Allows for functions with several arguments to be called as a chain of functions
  - Each of these curried functions has only one argument.
- Type Inference
  - Scala only has local type inference.
  - This is less powerful than the type of inference that is used in Haskell and ML.

# More Benefits to Scala

- Similarly to Ruby, Scala can create internal Domain Specific Languages.
  - This allows for the creation of meta-object protocols that allow developers
- Multi-threaded asynchronous coding is hard.
  - Actors make it easier
  - Code reuse with traits
    - Contains all of the common code
    - Has an abstract version of the code
    - Can create a class that fills in the code
- Scala's combination of the two programming paradigms allows for the code to be much shorter
  - LinkedIn reported an almost 50% reduction in the number of lines of code
- Scala is also incredibly fast.
  - This is a benefit of Sun Microsystems time put into Java optimization
  - Scala benefits from the Java optimization from its relation and use of the Java framework.
- It has very few reserved words.
  - This helps with the scalability
    - One  can easily grow types.
    - This is partially due to the First Class functional types  from the functional languages.

# Downsides of Scala

- Combination of Functional and Object oriented elements:
  - Wasn't this a benefit?
- Conceptually large system.
  - There is an almost overwhelming  amount of available tools.
  - Most programmers only use a small portion of the tools until something requires them.
- The flexibility can overwhelm even the best
  - There are many ways to solve a problem.
  - This only increases the possibilities for code inconsistency.
- Using libraries from other languages
  - While this can be powerful, it can also make code readability and debugging daunting.
- Scala has inherited several of the pitfalls and issues of JVM.

# Scala in the Real World

- Scala is not just a theoretical language used for giggles.
- Many large and well know companies are using Scala for development in some important fields.
  - Replaced Java for mission critical business applications.
    - "Source, supply transport, store, and convert physical commodities across the wholesale energy markets." (Scala in the enterprise, 2011)
  - Distributed client server applications
  - Simple and concise methods for improved performance
  - Allows for companies to continue to use their existing infrastructure.
    - Integrates with most middleware and backend systems.
  - Web Based home security systems
  - Mobile game logic
  - Energy management and analysis to help use energy more efficiently

# Practical Applications

- LinkedIn → Norbert, used in the social graph, search engine and used to help implement distributed client server applications.
- Sony Pictures Imageworks → "The Scala Migrations library is written in Scala and makes use of the clean Scala language to write easy to understand migrations, which are also written in Scala. Scala Migrations provides a database abstraction layer that allows migrations to target any supported database vendor."(Scala in the enterprise, 2011)
- Sygenca → Designs many products, including several for the UK Government
- **Twitter** → Moved their main message queue from Ruby to Scala for improved performance and half the code.

# Some Other Companies










# Resources

- The official site→ http://www.scala-lang.org/
  - This has the largest and most thorough documentation of Scala.
- Style guide → http://davetron5000.github.com/scala-style/index.html
- Information from the Creator → http://www.slideshare.net/Odersky/fosdem-2009-1013261
- Learning Scala from a Java background → http://www.ibm.com/developerworks/java/library/j-scala01228/index.html#N100BC

# Bibliography


- *Scala*. (2008, August 19). Retrieved 1 2011, November, from Scala: http://www.scala-lang.org/
- *Scala in the enterprise*. (2011, January 12). Retrieved November 1, 2011, from Scala: http://www.scala-lang.org/node/1658#LinkedIn
- Gougen, J. A. (1984, November 5). Parameterized Programming. *IEE Transactions on Software Engineering, SE-10*.
- Odersky, M., & Zenger, M. (2005). Scalable Component Abstractions. *OOPSLA Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications.*