Scala: A Functional, Object-Oriented Language

COEN 171

Darren Atkinson

What is Scala?

- Scala stands for <u>Sca</u>lable <u>La</u>nguage
  - It was created in 2004 by Martin Odersky.
  - It was designed to grow with the demands of its users.
  - It was designed to overcome many criticisms of Java.
  - It is compiled to Java bytecode and is interoperable with existing Java classes and libraries.
  - It is more of a high-level language than Java, having higher-order containers and iteration constructs built-in.
  - It encourages a functional programming style, much like ML and Scheme.
  - It also has advanced object-oriented features, much like Java and C++.

Using Scala

- Using Scala is much like using Python or ML, and is not as unwieldy as using Java.
- The Scala interpreter can be invoked directly from the command line:

```
$ scala
Welcome to Scala 2.11.8

scala> println("Hi!")
```

- The Scala interpreter can also be given a file on the command line to execute:

```
$ scala foo.scala
```

Scala Syntax

- Scala has a Java-like syntax with braces.
  - The assignment operator is simply =.
  - Strings are built-in and use + for concatenation.
  - Indexing is done using ( ) rather than [ ].
  - The first index is index zero.
  - Parameterized types use [ ] rather than < >.
  - A semicolon is inferred at the end of a line.
- However, since it is functional, everything is an expression and there are no “statements”.

Scala Types

- In Java, the primitive types are not objects and wrapper classes must be used.
  - `Integer` for `int`, `Boolean` for `bool`, etc.
- In Scala, everything is an object including the more “primitive” types.
  - The Scala types are `Int`, `Boolean`, `String`, etc.
- However, the Scala primitives are transparently converted to Java types by the Scala compiler.
  - So, “boxing” and “unboxing” of types is automatic.

Type Inference

- Like ML, Scala performs type interference, so it is not always necessary to declare the types of objects.
- Unlike ML, Scala's type interference is local, rather than global, so some type declarations are needed.
- In particular, parameters (but not return values) need to have their types declared.

No return needed

```
def add(x: Int, y: Int) = {
    x + y
}

def sub(x: Int, y: Int) = x - y
```

Variables vs. Values

- Variables are declared using the `var` keyword.
- However, Scala encourages functional programming and also supports values declared using `val`.
- Variables can be reassigned to many times. Values can only be assigned to once.

```
import scala.collection.mutable.Set

val movies = Set("Vertigo", "Topaz", "Rope")
movies += "Psycho"

movies = Set("Jaws", "Munich")   // error!
```

Mutable vs. Immutable

- Scala encourages functional programming through immutable objects.

- Arrays are mutable objects.

Type inference

```
scala> val x = Array(1,2,3)
x: Array[Int] = Array(1,2,3)

scala> x(0) = 10
```

- Lists are immutable objects.

```
scala> val y = List(1,2,3)
y: List[Int] = List(1,2,3)

scala> y(0) = 10      // error!
```

Functional Programming

- Anonymous functions are called "unnamed literals."

```
val increase = (x: Int) => x + 1
```

- Functions can be higher order, and a number of common utility functions are provided.

```
val x = List(1,2,3,4)
val y = x.filter(x => x > 2)
val z = x.map(x => x + 1)
```

- Curried functions are permitted.

```
def steph(x: Int)(y: Int) = x + y
val incr = steph(1)_
```

Parameter placeholder

Object-Orientation

- We've already seen that arrays and lists are objects that can have methods invoked.

- All operators are actually methods and vice versa!

```
val x = 1.+(2)
val y = List(1,2,3)
val z = y filter (x => x > 2)
```

- Scala also support classes, inheritance, and overriding inherited methods.

Immutable

```
class Rational(n: Int, d: Int) {
    val num = n
    val dem = d
    override def toString = num + "/" + dem
}
```

Conclusion

- Scala is a functional, object-oriented language.
- It has a Java-like syntax.
- Everything is an object, much like Smalltalk.
- Operators can be overloaded, much like C++.
- Functions are first-class values, much like ML.
- It does type inference, much like ML.
- Container classes are built-in, much like Python and other scripting languages.