Introduction to Scala

Amir H. Payberah
SICS Swedish ICT

amir@sics.se
Feb. 25, 2016


Scala

Scala

- Scala: scalable language
- A blend of object-oriented and functional programming
- Runs on the Java Virtual Machine
- Designed by Martin Odersky at EPFL

Functional Programming Languages

- Functions are first-class citizens:
  - Defined anywhere (including inside other functions).
  - Passed as parameters to functions and returned as results.
  - Operators to compose functions.

Scala Variables

- Values: immutable
- Variables: mutable

```
var myVar: Int = 0
val myVal: Int = 1
```

- Scala data types:
  - Boolean, Byte, Short, Char, Int, Long, Float, Double, String

If ... Else

```
var x = 30;

if (x == 10) {
  println("Value of X is 10");
} else if (x == 20) {
  println("Value of X is 20");
} else {
  println("This is else statement");
}
```

Loop

```
var a = 0
var b = 0
for (a <- 1 to 3; b <- 1 until 3) {
  println("Value of a: " + a + ", b: " + b )
}
```

```
// loop with collections
val numList = List(1, 2, 3, 4, 5, 6)
for (a <- numList) {
  println("Value of a: " + a)
}
```

Functions

```
def functionName([list of parameters]): [return type] = {
  function body
  return [expr]
}

def addInt(a: Int, b: Int): Int = {
  var sum: Int = 0
  sum = a + b
  sum
}

println("Returned Value: " + addInt(5, 7))
```

Anonymous Functions

- Lightweight syntax for defining functions.

```
var mul = (x: Int, y: Int) => x * y
println(mul(3, 4))
```

Higher-Order Functions


```
def apply(f: Int => String, v: Int) = f(v)

def layout(x: Int) = "[" + x.toString() + "]"

println(apply(layout, 10))
```

Collections (1/2)

- Array: fixed-size sequential collection of elements of the same type

```
val t = Array("zero", "one", "two")
val b = t(0) // b = zero
```

Collections (1/2)

- Array: fixed-size sequential collection of elements of the same type

```
val t = Array("zero", "one", "two")
val b = t(0) // b = zero
```

- List: sequential collection of elements of the same type

```
val t = List("zero", "one", "two")
val b = t(0) // b = zero
```

Collections (1/2)

- Array: fixed-size sequential collection of elements of the same type

```
val t = Array("zero", "one", "two")
val b = t(0) // b = zero
```

- List: sequential collection of elements of the same type

```
val t = List("zero", "one", "two")
val b = t(0) // b = zero
```

- Set: sequential collection of elements of the same type without duplicates

```
val t = Set("zero", "one", "two")
val t.contains("zero")
```

Collections (2/2)

- Map: collection of key/value pairs

```
val m = Map(1 -> "sics", 2 -> "kth")
val b = m(1) // b = sics
```

Collections (2/2)

- Map: collection of key/value pairs

```
val m = Map(1 -> "sics", 2 -> "kth")
val b = m(1) // b = sics
```

- Tuple: A fixed number of items of different types together

```
val t = (1, "hello")
val b = t._1 // b = 1
val c = t._2 // c = hello
```

Functional Combinators

- map: applies a function over each element in the list

```
val numbers = List(1, 2, 3, 4)
numbers.map(i => i * 2) // List(2, 4, 6, 8)
```

- flatten: it collapses one level of nested structure

```
List(List(1, 2), List(3, 4)).flatten // List(1, 2, 3, 4)
```

- flatMap: map + flatten

- foreach: it is like map but returns nothing

Classes and Objects

```
class Calculator {
  val brand: String = "HP"
  def add(m: Int, n: Int): Int = m + n
}

val calc = new Calculator
calc.add(1, 2)
println(calc.brand)
```

Classes and Objects

```
class Calculator {
  val brand: String = "HP"
  def add(m: Int, n: Int): Int = m + n
}

val calc = new Calculator
calc.add(1, 2)
println(calc.brand)
```

- A singleton is a class that can have only one instance.

```
object Test {
  def main(args: Array[String]) { ... }
}

Test.main(null)
```

Case Classes and Pattern Matching

- Case classes are used to store and match on the contents of a class.
- They are designed to be used with pattern matching.
- You can construct them without using new.

```
case class Calc(brand: String, model: String)

def calcType(calc: Calc) = calc match {
  case Calc("hp", "20B") => "financial"
  case Calc("hp", "48G") => "scientific"
  case Calc("hp", "30B") => "business"
  case _ => "Calculator of unknown type"
}

calcType(Calc("hp", "20B"))
```

Questions?