

JUG
Poznań



Scala

Scala

Maciej Biłas

maciej@inszy.org / @maciejb

What is Scala?

Scala is a general purpose programming language designed to express common programming patterns in a **concise, elegant,** and **type-safe** way. It smoothly integrates features of **object-oriented** and **functional** languages. It is also fully interoperable with **Java**.

Scala facts

- Created by Martin Odersky at EPFL in 2001. Among others the author of javac
- Multi-paradigm language (mix of functional and object-oriented)
- Executes on the JVM (.NET version also available)
- Can run in script or application modes

Scala facts cont'd

- Has a REPL (interactive shell)
- Pure OO (no primitive types or static members)
- Current stable version: 2.8.0
- Can be obtained at <http://www.scala-lang.org/downloads>

Who is using Scala?

- Twitter
- LinkedIn
- Novell Pulse
- Foursquare
- Sony, Siemens, Xerox and others

Why learn Scala?

- Write leaner and more concise code
- Leverage Java libraries
- Learn a different (arguably better) concurrency model
- To become a better programmer in *your* language

Fastest benchmark programs

compare 2	<u>+</u>	<u>---</u>	<u>25%</u>	<u>median</u>	<u>75%</u>	<u>---</u>	<u>-</u>
<u>C GNU gcc</u>	1.00	1.00	1.04	1.10	1.35	1.83	3.80
<u>C++ GNU g++</u>	1.00	1.00	1.00	1.11	1.24	1.55	1.55
<u>Java 6 steady state</u>	1.00	1.00	1.15	1.57	1.96	2.06	2.06
<u>Scala</u>	1.14	1.14	1.19	2.00	2.39	4.19	6.67
<u>Java 6 -server</u>	1.10	1.10	1.36	2.18	3.63	6.80	6.80
<u>C# Mono</u>	1.69	1.69	1.97	2.84	5.16	9.95	18.38
<u>JavaScript V8</u>	1.00	1.00	4.26	7.19	20.52	44.89	102.86
<u>Clojure</u>	1.55	1.55	4.68	11.47	15.59	29.58	29.58
<u>Ruby JRuby</u>	16.50	16.50	23.95	43.08	153.45	225.45	225.45
<u>Python CPython</u>	2.28	2.28	5.74	47.96	93.11	106.47	106.47
<u>Python 3</u>	2.23	2.23	7.10	50.09	128.75	157.73	157.73
<u>Ruby 1.9</u>	7.64	7.64	15.33	63.66	106.91	244.27	265.59
<u>PHP</u>	6.90	6.90	48.32	104.15	149.90	233.84	233.84
<u>Ruby MRI</u>	15.28	15.28	24.72	158.27	481.13	856.12	856.12

x86 Ubuntu™ Intel® Q6600® one core

Benchmark game: <http://shootout.alioth.debian.org/> Data: <http://j.mp/9Y7xo2>

Hello, World!

```
println("Hello, World!")
```

Hello, World!

For a start: no semicolon at the end

```
println("Hello, World!")
```



...or...

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

Something a bit more complex

```
Map<String, Integer> phonebook =  
    new HashMap<String, Integer>();  
phonebook.put("Bob", 12345);  
phonebook.put("Tim", 23154);  
  
System.out.println(  
    phonebook.get("Bob"));
```

Yes, it is Java...

Something a bit more complex

```
val phonebook = Map("Bob" -> 12345,  
                    "Tim" -> 23154)  
  
println(phonebook("Bob"))
```

Quick language tour ahead

- `val` or `var`
- Methods and closures
- Case classes and pattern matching
- Traits
- `for` Comprehensions
- `Implicits`

Things left out

- A description of the List class (shame on me)
- Language support for XML
- Actors library (sorry folks)
- Structural typing
- All the gory language details that won't fit into a 45 minute talk

val or var?

```
val i = 2  
var foo = "bar"  
foo = "baz"
```

```
i = i + 2  
throws a compiler error
```

val or var?

```
val i = 2  
var foo = "bar"  
foo = "baz"
```

```
i = i + 2
```

throws a compiler error

- You need to think in advance if something is mutable or not
- No excuse for missing `final` keyword
- Watch out for mutable objects

Methods

```
def sum(a : Int, b : Int) =  
  a + b
```

```
sum(1, 2) // returns 3
```

Method cont'd

```
object Foo {  
  def +*%!(a : Int) =  
    a+2  
}
```

```
Foo.+*%!(3)
```

```
Foo.+*%! 3
```

```
1.toString
```

```
1.+(2)
```

Method cont'd

```
object Foo {  
  def +*%!(a : Int) =  
    a+2  
}
```

```
Foo.+*%!(3)  
Foo.*%! 3
```

```
1.toString  
1.+ (2)
```

- non-alphanumeric method identifiers are valid
- no operator overloading, operator notation instead
- DSL anyone?

Complex.scala

```
class Complex(val re: Double, val im: Double) {  
  
  def + (that : Complex) = new Complex(this.re + that.re,  
                                       this.im + that.im)  
  def - (that : Complex) = new Complex(this.re - that.re,  
                                       this.im - that.im)  
  def * (that : Complex) =  
    new Complex(this.re * that.re - this.im * that.im,  
               this.re * that.im + this.im * that.re)  
  
  def / (that : Complex) = {  
    import math.sqrt  
    val denom = sqrt(that.re) - sqrt(that.im)  
    new Complex((this.re * that.re + this.im * that.im) / denom,  
               (this.im * that.re - this.re * that.im) / denom)  
  }  
  
  override def toString = re + (if (im < 0) "-" else "+") + "i" + im  
}
```

Local methods

```
import scala.io.Source

object LongLines {
  def processFile(filename: String,
                  width: Int) {
    def processLine(line: String) {
      if (line.length > width)
        println(filename + ": " + line)
    }

    val source = Source.fromFile(filename)
    for (line <- source.getLines)
      processLine(line)
  }
}
```

- Methods can be nested in one another
- Alternative to private class methods

Functions are objects

```
def sum(a : Int, b : Int) = a + b
val sumValue = sum _
val add2 = sum(2, _ : Int)

val dec2 = (x : Int) => x - 2
```

Function objects

Java

```
boolean hasUpperCase = false;
Predicate<Character> isUpperCase = new
Predicate<Character>() {
    @Override
    public boolean apply(Character c) {
        return Character.isUpperCase(c);
    }
};

for (int i = 0; i < str.length(); i++) {
    if (isUpperCase.apply(str.charAt(i))) {
        hasUpperCase = true;
        break;
    }
}
```

Function objects

Scala

```
val hasUpperCase =  
  str.exists(_.isUpper)
```

Function objects

which can be expanded to

```
val hasUpperCase =  
  str.exists(c => c.isUpper)
```

Function objects

which can be expanded to

```
val hasUpperCase =  
  str.exists(c => c.isUpper)
```

C is statically typed. Compiler infers it's a Char



Closures

```
var more = 1  
val addMore =  
    (x: Int) => x + more  
addMore(10) // returns 11
```

```
more = 2  
addMore(10) // returns 12
```

Pattern matching

```
def matchAny(a : Any) : Any = a match {  
  case 1           => "one"  
  case "two"       => 2  
  case i: Int      => "scala.Int"  
  case <tag>{ t }</tag> => t  
  case head :: tail => head  
  case _          => "default"  
}
```

Case classes

Not in Java

```
public class Point {
    private final double x;
    private final double y;

    public Point(double x, double y) {
        super();
        this.x = x;
        this.y = y;
    }

    public double getX() {
        return x;
    }

    public double getY() {
        return y;
    }

    @Override
    public String toString() {
        return "Point [x=" + x + ", y="
+ y + "]";
    }

    ...
}
```

```
@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    long temp;
    temp = Double.doubleToLongBits(x);
    result = prime * result + (int) (temp ^ (temp >>> 32));
    temp = Double.doubleToLongBits(y);
    result = prime * result + (int) (temp ^ (temp >>> 32));
    return result;
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Point other = (Point) obj;
    if (Double.doubleToLongBits(x) != Double.doubleToLongBits(other.x))
        return false;
    if (Double.doubleToLongBits(y) != Double.doubleToLongBits(other.y))
        return false;
    return true;
}
}
```

Case classes

```
case class Point(x : Double,  
                y : Double)
```

...combined

```
val p = Point(0, 30)

p match {
  case Point(x, y) if y > 20.0
    => "foo"
  case _ => "bar"
}
```

Regular expressions

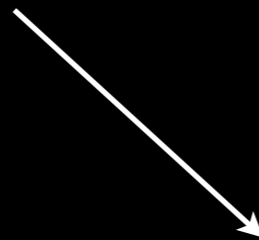
```
val Decimal = """"(-)?(\d+)(\.\d*)?""".r
```

```
val Decimal(sign, integerPart, decimalPart) = "-1.23"
```

```
println("sign: " + sign +  
        ", i: " + integerPart +  
        ", d: " + decimalPart)
```

Regular expressions

Raw string



```
val Decimal = """"(-)?(\d+)(\.\d*)?""".r
```

```
val Decimal(sign, integerPart, decimalPart) = "-1.23"
```

```
println("sign: " + sign +  
        ", i: " + integerPart +  
        ", d: " + decimalPart)
```

Regular expressions

Raw string

via implicit conversion

```
val Decimal = """"(-)?(\d+)(\.\d*)?""".r
```

```
val Decimal(sign, integerPart, decimalPart) = "-1.23"
```

```
println("sign: " + sign +  
        ", i: " + integerPart +  
        ", d: " + decimalPart)
```

Traits

Interfaces with methods... soft of

Traits

```
trait LikeAnInterface {  
    def aMethod(arg: List[Float])  
        : Unit  
}
```

Traits

```
trait Philosophical {  
    def philosophize() {  
        println("I consume memory,  
                therefore I am!")  
    }  
}  
  
new Philosophical() // won't compile  
  
(new AnyRef  
    with Philosophical) philosophize
```

Abstract values

```
trait HasId {  
    val id: Int  
}
```

Abstract values

```
trait HasId {  
    val id: Int  
}
```

You can have abstract vars as well

Stackable modifications I

```
abstract class IntQueue {  
  def get(): Int  
  def put(x: Int)  
}
```

```
class BasicIntQueue extends IntQueue {  
  import scala.collection.mutable.ArrayBuffer  
  private val buf = new ArrayBuffer[Int]  
  def get() = buf.remove(0)  
  def put(x: Int) { buf += x }  
}
```

```
val q = new BasicIntQueue()  
q.put(2); q.get // return 2
```

Stackable modifications 2

```
trait Incrementing extends IntQueue {  
  abstract override def put(x: Int) {  
    super.put(x + 1)  
  }  
}
```

```
trait Doubling extends IntQueue {  
  abstract override def put(x: Int) {  
    super.put(2 * x)  
  }  
}
```

Stackable modifications 3

```
val qdi = new BasicIntQueue()  
    with Doubling  
    with Incrementing  
qdi.put(2)  
qdi.get // returns 6
```

```
val qid = new BasicIntQueue()  
    with Incrementing  
    with Doubling  
qid.put(2)  
qid.get // returns 5
```

for comprehension

```
for (n <- names)  
  println(n)
```

for comprehension

```
val polishAlices =  
for { p <- people  
    if p.name == "Alice"  
    lang <- p.spokenLanguages  
    if lang == "pl"  
} yield p
```

for comprehension

- Works with user defined classes. No need to add any traits
- Just define `map`, `flatMap`, `filter` and `foreach` methods

Remember the Complex class?

Wouldn't `Complex(1.0, 2.0) + 2.0` be nice?

Implicit conversions
instead of open classes

Let's add a companion object

```
object Complex {  
  def apply(re : Double, im: Double) =  
    new Complex(re, im)  
  
  val i = new Complex(0, 1)  
  
  implicit def double2complex(x: Double) =  
    new Complex(x, 0)  
}
```

```
Complex.i + 2.0
```

```
import Complex.{double2complex, i}  
2.0 + i
```



```
package scala
object Predef {
  class ArrowAssoc[A](x: A) {
    def -> [B](y: B): Tuple2[A, B] = Tuple2(x, y)
  }

  implicit def any2ArrowAssoc[A](x: A): ArrowAssoc[A] =
    new ArrowAssoc(x)
}
```

should have matchers

map should have key ("a")

Uff...

Scala Test

```
class StackSpec extends FlatSpec with ShouldMatchers {  
  "A Stack" should "pop values in last-in-first-out order" in {  
    val stack = new Stack[Int]  
    stack.push(1)  
    stack.push(2)  
    stack.pop() should equal (2)  
    stack.pop() should equal (1)  
  }  
  
  it should "throw NoSuchElementException if an empty stack is popped" in {  
    val emptyStack = new Stack[String]  
    evaluating { emptyStack.pop() } should produce [NoSuchElementException]  
  }  
}
```

Tools

- Eclipse – <http://www.scala-ide.org/>
- NetBeans – <http://wiki.netbeans.org/Scala#4. Adventure with NetBeans Nightly Build>
- IntelliJ IDEA
- Emacs, vim, JEdit, TextMate...

Resources: books

- Odersky et al. Programming in Scala. Artima (2008)
- Wampler and Payne. Programming Scala Objects. O'Reilly (2009) *HTML version*
- Subramaniam. Programming Scala: Tackle Multi-Core Complexity on the Java Virtual Machine. Pragmatic Programmers (2009)

Resources: web

- www.scala-lang.org
- www.simplyscala.com
- liftweb.net
- [http://stackoverflow.com/questions/476111/
scala-programming-for-android](http://stackoverflow.com/questions/476111/scala-programming-for-android)

Questions?