Overview of Java Virtual Machine Languages

Pavol Pidanič
pavol.pidanic@accenture.com

25. 10. 2016

Overview

- Java Virtual Machine
- JVM languages
- Scala features

Java Virtual Machine

Java Virtual Machine

- *„The original engineers of Java technology made a brilliant decision to separate the language from the runtime.“*
- *„This architecture is crucial for the platform's long-term vitality, because computer programming languages typically have short lifespans“*

Neal Ford

Java Virtual Machine

- Bytecode runs on a virtual machine
- „Compile once, run everywhere“
- Automatic garbage collection

Why other languages?

- *„The Java language has proved quite elastic in capabilities, but its syntax and inherent paradigms have long-understood limitations. Despite the promising changes that are coming to the language, the syntax simply can't support some important future goals, such as elements of functional programming.“ Neal Ford*
- *„The legacy of Java will be the platform, not the language.“ Martin Fowler*

Why other languages?

- Compatibility, operability
  - Use Java libraries in other and vice versa

JVM Languages

- How many?

JVM Languages

- More than 200[1]
- 20 + 32[2]

1 http://www.ibm.com/developerworks/java/library/j-jn1/index.html
2 https://en.wikipedia.org/wiki/List_of_JVM_languages

JVM Languages

- More than 200
- 20 + 32
  - 20 implementations of existing languages
    - JRuby, Jython, Nashorn, ...
  - 32 new languages
    - Ceylon, Clojure, Groovy, Kotlin, Scala, ...

Ceylon

- Object-oriented, imperative
- Strong static typed
- Influenced by Java with new features
- Generic programming and (type-safe) metaprogramming, with reified generics
- Type inference, automatic getters, setters, packaging
- Released 2011
- Actual version 1.3.0

```
shared void hello() {
    print("Hello, World!");
}
hello();
```

Clojure

- Functional
- Lisp-based syntax
- Dynamically typed
- Version 1.0. released 4. 5. 2009
- Actual version 1.8

```
(println "Hello world!")
```

Groovy

- Object-oriented
- Imperative
- Dynamically typed
- Metaprogramming, scripting
- Version 1.0 released on 2.1.2007
- Actual version 2.4.7

```
class HelloWorld {
   static main( args ){
      println "Hello World!"
   }
}


println "Hello World!"
```

Kotlin

- Static typed
- Imperative
- Goal is to compile as quickly as Java
- Development with tooling in mind
- First appeared in 2012
- Actual version 1.0.4

```
fun main(args : Array<String>)
{
  val scope = "world"
  println("Hello, $scope!")
}
```

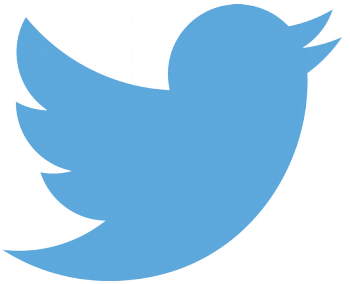
Scala

- Object-oriented
- Functional
- Strong static typed – type inference
- Released 2004
- Actual version 2.11.8

```
object HelloWorld extends App {
   println("Hello, World!")
 }
```

More details of Scala

Scala in enterprise



Quick (lazy) guide to Scala

- Everything is an object
- Source file does not need to match class name
- `var`, `val`
- `class, trait, object`
- index with - `()`
- generics/parametrized type - `[ ]`
- `Unit` type
- Syntactic sugar - optional `; . () { }`

Quick (lazy) guide to Scala

- `def`
- Wildcard - `_`
- `if-else` evaluates in a value
  - `else` is not optional
- Implicit return on function last line
- Multiline string with “““ “““
- Import anywhere in source file
- Primary constructor with class declaration

```
def myMethod(a: Type): RetType = {
   // code goes here
}
val myMethod2: (a: Type => RetType) =
   //code goes here
}
```

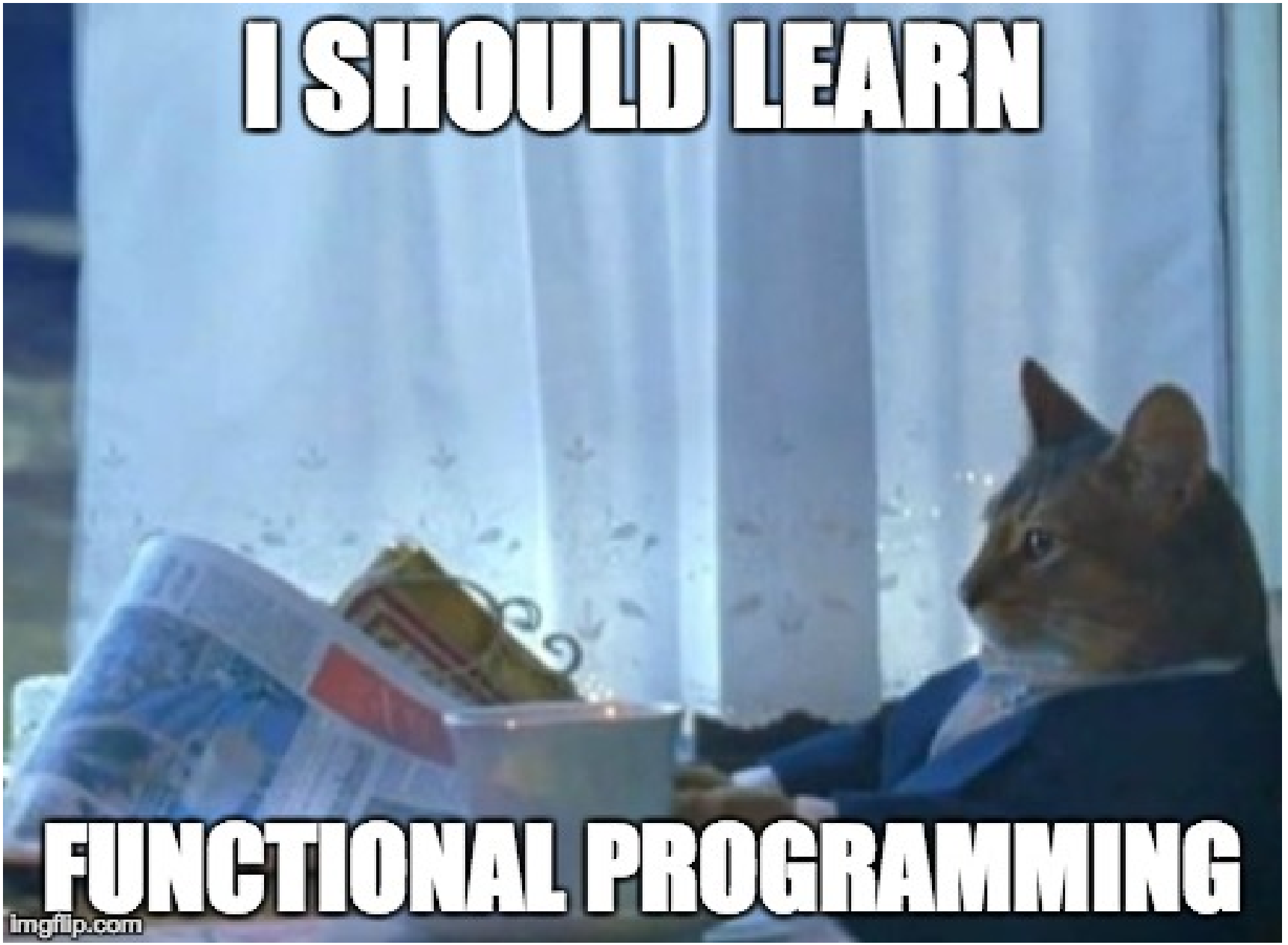
Imperative vs. Functional

Imperative vs. Functional

- Imperative programming focuses on step-by-step instructions, in many cases mimicking ancient low-level hardware conveniences
- Imperative languages try to make mutating state easier and include many features for that purpose.
- Functional languages try to minimize mutable state and build more general-purpose machinery
- In functional code, the output value of a function depends only on the arguments that are input to the function. Eliminating side effects.

Functional programming

- Lambda calculus
- Immutability
- Pure functions vs. Side effects
- First-class and higher-order functions
  - Currying
  - Partial application
- Closure
- (Tail) Recursion
- Strict vs non-strict evaluation
- *… next session*


I SHOULD LEARN
FUNCTIONAL PROGRAMMING
imgflip.com

Uniform Access Principle

Uniform Access Principle

- Client should not be affected by decistion to implement field or method
  - State and parameterless function should be accessed using same syntax
- Function calls without parentheses

Uniform Access Principle

- Example

Option type

Option type

- *„I call it my billion-dollar mistake. [...] This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.“*

  Tony Hoare – Inventor of Algol W at Null coference 25.8.2009

Option type

- Something returned can be „nothing“
- `Option[T]` parent class
  - `Some[T]`, `None` subclasses

Option type

- Something returned can be „nothing“
- `Option[T]` parent class
  - `Some[T]`, `None` subclasses

- Example

Implicit conversion

- What if you want to extend existing library which you don't have control
  - Whenever compiler sees type X but needs Y
  - Converting the receiver of a method call, the object on the method is invoked

Implicit conversion

- Example

Collection API

- Immutable and mutable collections
  - Default immutable
- Generic arrays
- Tuples
- Infinite collection
- View

Trait

- Unit of code reuse
- Type definition
- Mix into the classes (`extends` or `with`)
  - Stackable modification

Stackable modification

- „Trait lets you *modify* the method of a class, and they do so in a way that allows you to **stack** those *modifications* with each other“

Stackable modification

- Example

Pattern matching

- „`switch` on steroids“
- Case classes – non-encapsulated data structure
- Match evaluates as value
- Don't „fall through“

Pattern matching

- Example

Other features

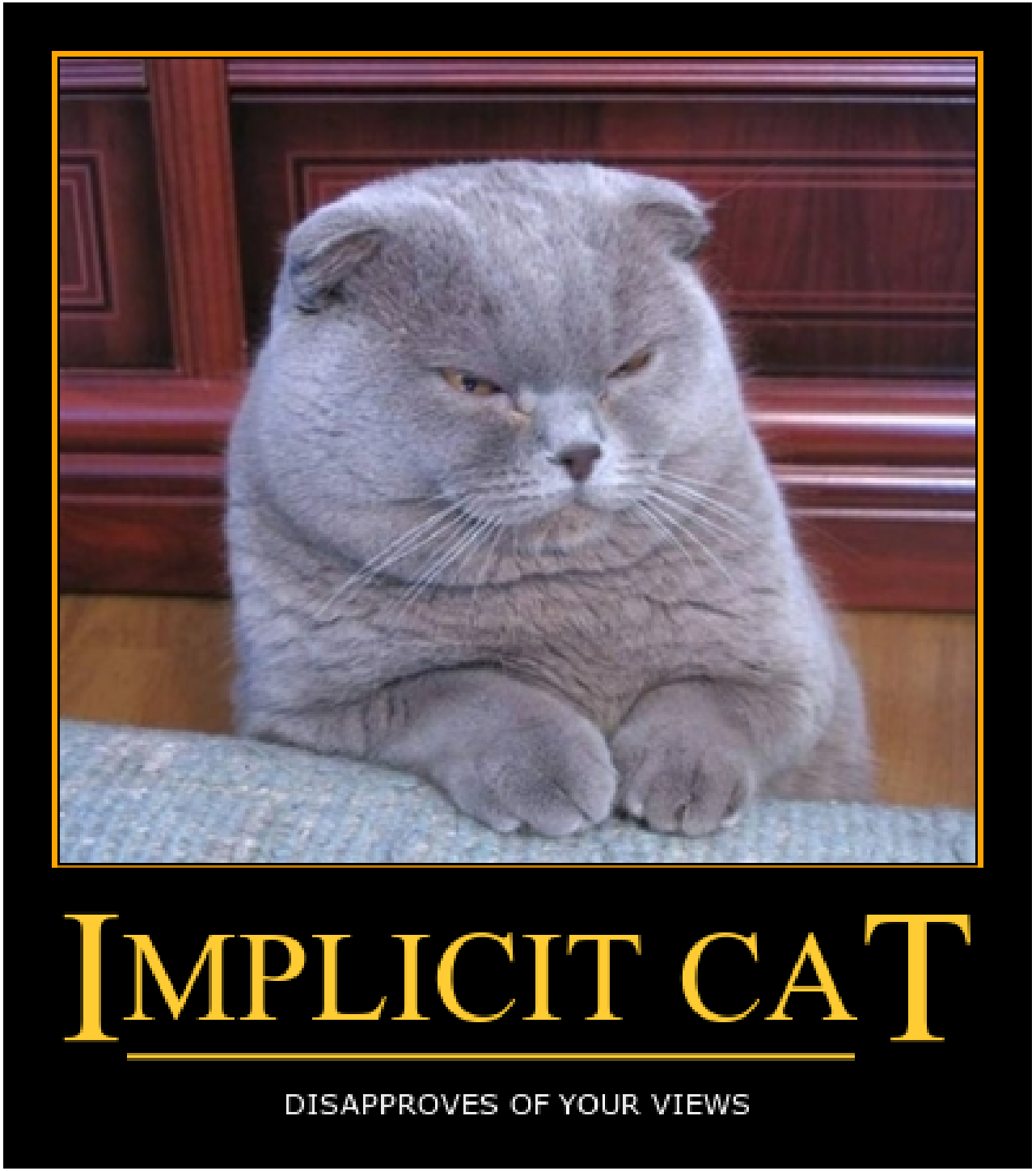
- Packaging refined
- Concurrency
- By-name parameters
- `Try-catch` results a value
- Writing your own control structure
- XML literals and functions
- … and more

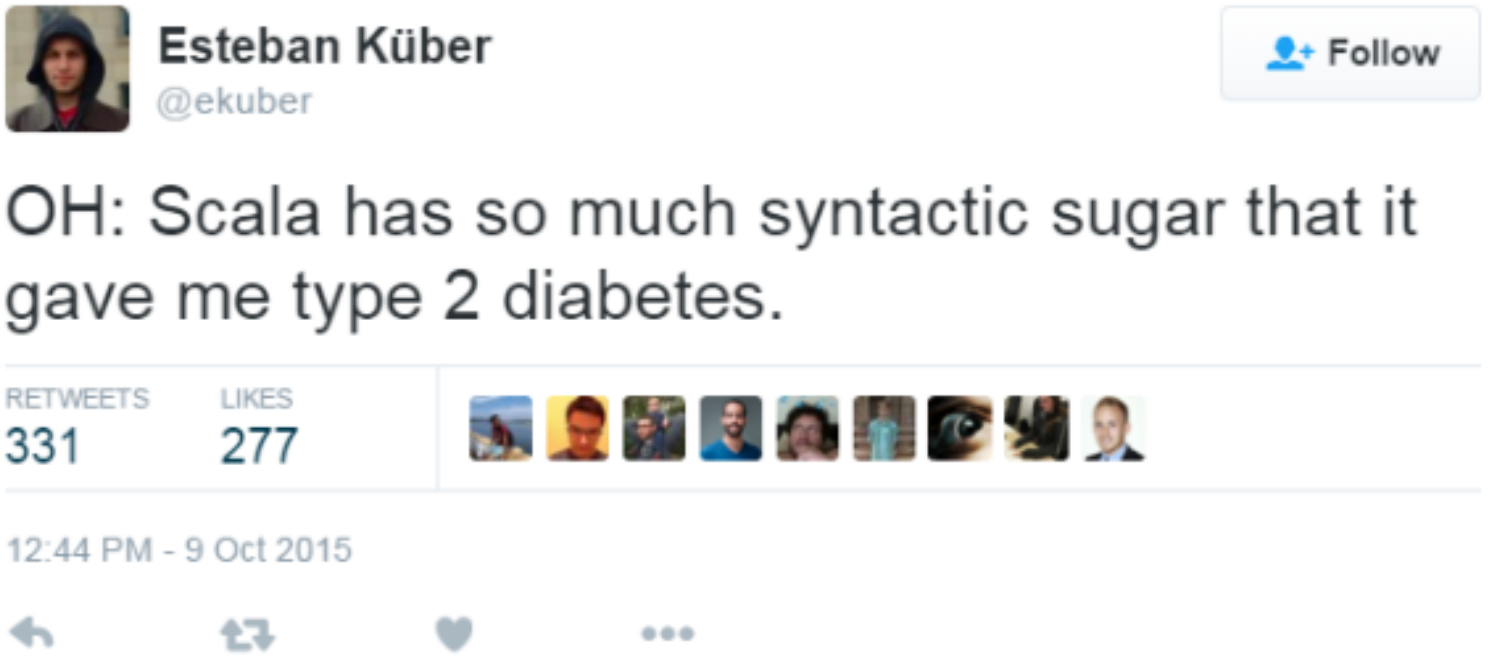
Negatives

Learning curve

- Learning advanced features
- Multiparadigm languages offer immerse power, giving you the ability to mix and match paradigms closely match the problem
- Require more developers discipline on large projects. Because of many abstractions and philosophies, isolated groups can create starkly different variants in libraries

- Weird syntax
  - `::`
  - `:::`
  - `/:`
  - `:/`
  - `#::`


IMPLICIT CAT
DISAPPROVES OF YOUR VIEWS


Esteban Küber
@ekuber
Follow
OH: Scala has so much syntactic sugar that it gave me type 2 diabetes.
RETWEETS
331
LIKES
277
12:44 PM - 9 Oct 2015

Resources

- Neal Ford - Java.next (blog series)
- Neal Ford - Functional Thinking (blog series)
- Wikipedia
- Martin Odersky, Bill Venners, Lex Spoon - Programming in Scala
- https://dzone.com/articles/a-qa-with-andrey-breslav-on-kotlin
- https://dzone.com/articles/java-scala-ceylon-evolution
- Busy Java developer's guide to Scala
- https://www.quora.com/What-startups-or-tech-companies-are-using-Scala
- https://medium.com/@kvnwbbr/transitioning-to-scala-d1818f25b2b7#.9dvmemmfi


IT CAME TO THEM WITH
A MESSAGE.
BUT THEY COULD NOT UNDERSTAND ITS ALIEN LANGUAGE
Y'ALL SHOULD WRITE SCALA
FUCKING WHAT?
poorlydrawnlines.com

Thank you

Examples

- https://github.com/PaulNoth/bbs-jvm-languages-and-scala