

# Ioke



A folding language

**Ola Bini**

ola.bini@gmail.com

<http://olabini.com/blog>



# About me

Stockholm, Sweden

ThoughtWorks

JRuby core developer

Ioke creator

... ask me about programming languages. or AI.



# What is Ioke?

An experiment

A programming language

Dynamic and strong typing

Prototype based object orientation

Homoiconic

Inspirations: Io, Ruby, Self/Smalltalk, Lisp

Hosted on the JVM

Current version: Ioke S i.k.j 0.2.0



# Getting started

Make sure you have Java 5 or better

Download latest tar/zip from <http://ioke.org/download.html>

Unpack into \$IOKE\_HOME

Add \$IOKE\_HOME/bin to \$PATH

```
git clone git://github.com/olabini/ioke.git
```

```
cd ioke
```

Make sure you have ant installed

```
ant
```

Add `pwd`/bin to \$PATH



# Philosophy

Expressiveness over performance

Abstraction over low level interfaces

Higher order functionality over explicitness

“Right is better” over “Worse is better”

Language oriented programming over APIs

“Code as data” over “data as code”

Homoiconicity over syntax

Syntax over explicit API's



Goal

ThoughtWorks®

How expressive can you make a language?

# What is expressiveness?

## Power

Is Perl 5 more powerful than Perl 4?

Ruby 1.8 vs Ruby 1.6?

Java 1.5 vs Java 1.4?

Basic vs assembler?

Cobol vs C++?

C++ vs Java?

Java vs Lisp?

## Abstraction

Mapping from mind to code



# Syntax

**ThoughtWorks®**

No dots for method calls

Period is just that - the end of a sentence

Semicolon is the start of a comment

Most other things will remind you of Ruby



Demo  
Syntax





# Operators

“Binary” operators

+, -, \*, /, and, return ...

“Trinary” operators

=, +=, ++, /= ...

Inverted operators

::



# Control structures

if, cond, case

loop, while, until, times, each

break, continue, return

for



Demo  
Control

# Prototype based OO

Only one kind of object

Every object has zero or more *mimics*

Every object has zero or more *cells*

Convention: everything that has a name that starts with a capital letter, is called a *kind*. A kind can be thought of as a class like object whose main purpose is to be the mimic of other objects.





# Methods

Positional arguments

Keyword arguments

Rest arguments

Keyword rest arguments

Splatting

# Conditions

## Generalization of Exceptions

### Concepts:

Condition - something that needs to be signalled

Rescue - a piece of code that rescues from a condition signal

Handler - a piece of code that handles a condition signal

Restart - a piece of code that can recover from the condition

How do you handle exceptions?

Exceptions are only for errors - what about warnings?

What about other things?



Demo  
Conditions



# Homoiconic?

```
class Foo
  def bar(x, y)
    puts "hello"
    puts [1,2,3].map {|z| z*z}
    x*y
  end
end
```



# JRuby AST (the MRI is worse)

```
RootNode
  NewlineNode
  ClassNode
    NewlineNode
    DefnNode |bar|
      ArgsPreTwoArgNode
      BlockNode
        NewlineNode
        FCallOneArgNode |puts|
          ArrayNode
            StrNode == "hello"
        NewlineNode
        FCallOneArgNode |puts|
          ArrayNode
            CallNoArgBlockNode |map|
              ArrayNode
                FixnumNode == 1
                FixnumNode == 2
                FixnumNode == 3
              IterNode
                DAsgnNode |z| &0 >0
                NilImplicitNode |nil|
              NewlineNode
              CallOneArgNode |*|
                DVarNode |z| &0 >0
                ArrayNode
                  DVarNode |z| &0 >0
            NewlineNode
            CallOneArgNode |*|
              LocalVarNode |x| &0 >0
              ArrayNode
                LocalVarNode |y| &1 >0
```



# The Ioke version

```
Foo = Origin mimic do(  
  bar = method(x, y,  
    "hello" println  
    [1,2,3] map(z, z*z) println  
    x*y))
```

AST:

```
=(Foo, Origin mimic do(  
  =(bar, method(x, y,  
    "hello" println  
    [] (1,2,3) map(z, z *(z)) println  
    x *(y))))))
```



# Messages

Everything is a message

Message is a kind in Ioke - and can be created from scratch

A message has a name

A message has zero or more arguments

A message has a prev pointer

A message has a next pointer

This is the AST



Demo  
Messages

# Types of code

activatable?

DefaultMethod

**method**

LexicalBlock

**fn, fnx**

DefaultMacro

**macro, dmacro**

LexicalMacro

**lecro, lecrox, dlecro, dlecrox**

DefaultSyntax

**syntax, dsyntax**



Demo  
Macros





# Aspects

before, after and around returns pointcuts

Pointcuts have methods to add and remove advice

Only affects those cells that use it

Typical usage:

Origin `around(:mimic)` defines aspect to call initialize

Implemented fully in Ioke



Demo  
Advice



# ISpec

**ThoughtWorks®**

Minimal, inspired by RSpec

A few hundred lines of code

No mocking right now

Brian Guthrie is working on support for this

Used for the Ioke test spec - currently about 2700 specs.



Demo  
ISpec



# DokGen

Generates HTML documents

Walks the living structure of the Ioke system

Combines this structure with ISpec information



Demo  
DokGen



# Java Integration

It's coming...

Not finished yet, though

Will support calling and using Java methods and fields

Implementing interfaces

Extending classes

Ioke E will contain full Java Integration support



# Future

Ioke E, Ioke P, Ioke F

Java Integration

Concurrency support

Units

Important libraries

IO, Sockets

Cane - Package management tool

SQL DSL?

JRuby integration

CLR and V8 runtimes



# The project

Git at Github - <http://github.com/olabini/ioke>

Hg mirror at Kenai - <http://kenai.com/hg/ioke~mercurial>

Mailing lists at Kenai - <http://ioke.kenai.com>

[ioke.org](http://ioke.org)

- Programming guide

- Wiki

- IRC logs

- Current release Doks

IRC: *#ioke* at Freenode



# Contributors

TW

Sam Aaron

Carlos Villela

Brian Guthrie

Martin Elwin

Felipe Rodrigues de Almeida

Q

and

A



Q



A



>

<