# INTEROPERABILITY

## CLOJURE ⇄ PYTHON

Enrico Franchi

# OUTLINE

Why Clojure? Why Java?

Clojure from 3000 m.

Jython-Clojure interoperability

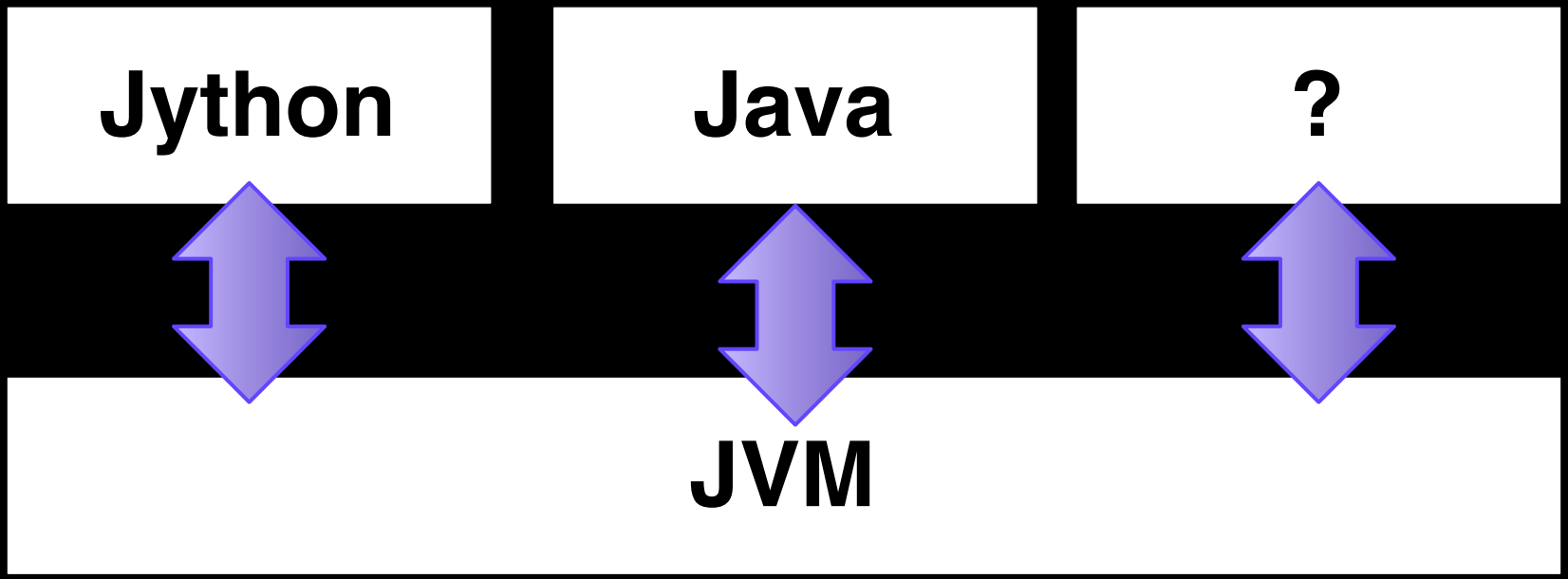
Clojure-Jython interoperability

# GENERAL NOTES

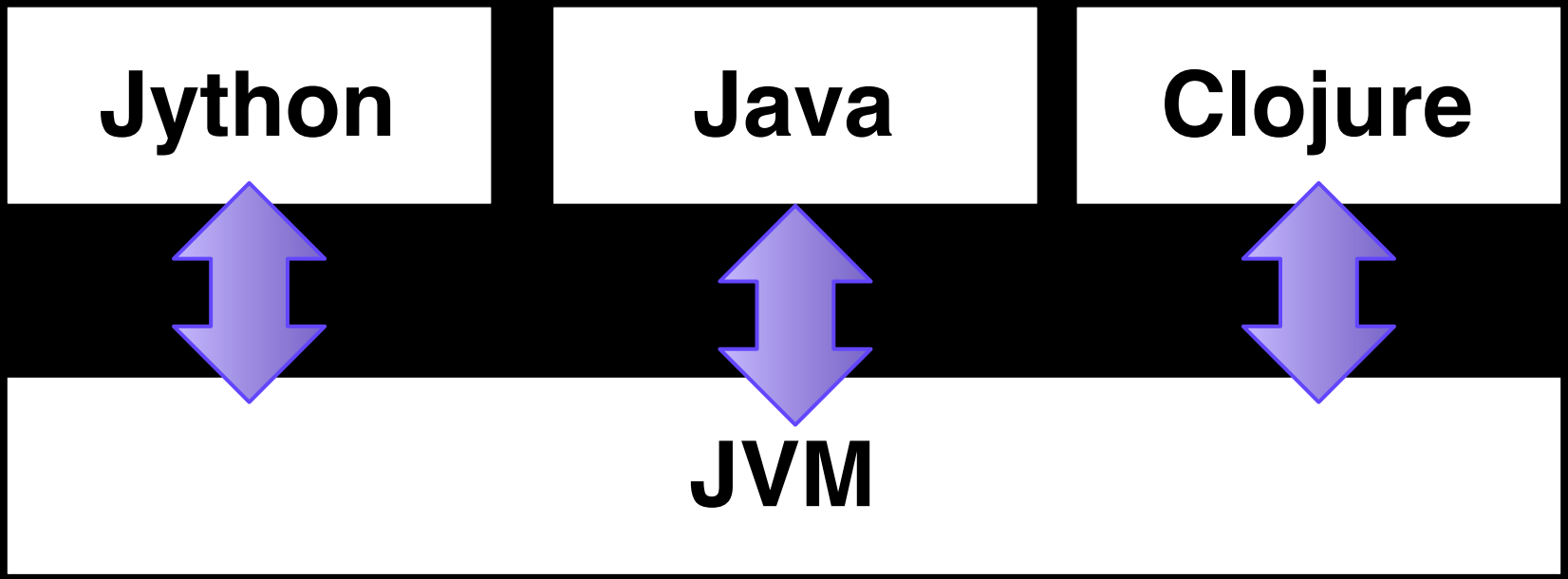
~~During this presentation some very explicit [illegible] be shown.~~

No duc[illegible]tion.

# JVM LANGUAGES



# JVM LANGUAGES



# CLOJURE

Clojure is Lisp

Clojure is a *good* Lisp

Clojure has a more functional flavor than Common Lisp;
stateful programming is banned unless in very controlled ways

Clojure lives on the JVM and perhaps its rejection of state
is a reaction to the heavily stateful model of JVM

Clojure is **not** a pure functional language by any means

# JVM LANGUAGES

Jython

Implementation of Python in Java

Jython calls Java

~

Clojure

New Programming Languages built on the JVM

Design choices in Clojure reflect design choices of the JVM

Interoperability

# CLOJURE

~ Good Parts ~

Functional Programming
Laziness
Full Macros
Multi-Methods
Immutable types, STM & Concurrency Model

~ Bad Parts ~

"Java"

# ATOMIC TYPES

Integers

Floating Point Numbers

Ratios (3/4 is not 0.75, is ¾)

BigDecimal

Strings (“foo”)

Booleans (true, false)

Nil (nil)

Characters (\a, \b)

:keywords

regexp’s (#“foo.*”)

# SEQUENCE TYPES

| | Python | | Common Lisp | | Clojure | |
|---|---|---|---|---|---|---|
| | Type | Syntax | Type | Syntax | Type | Syntax |
| **Random access sequence** | list | [1, 2] | vector | #(1 2) | vector | [1 2] |
| **Linked List** | - | No | list | (1 2) | list | (1 2) |
| **Set** | set | {1, 2} | No | No | set | #{1 2} |
| **Map** | dict | {1:2, 3:4} | hash-table | No | vector | {1 2, 3 4} |

In Clojure *all* collections are immutable;
all the functions return a new collections – as usual,
immutability allows easy data sharing –

Clojure collections implement corresponding Java collections interfaces

| | |
|---|---|
| vector, list | List |
| set | Set |
| map | Map |

# CLOJURE EXAMPLE

```
(defn rember [a lat]
  (cond
   (empty? lat) '()
   (= (first lat) a) (rest lat)
   :else (cons
          (first lat)
          (rember a (rest lat)))))

(rember 4 '(1 2 3 4 5 6 4))
; => (1 2 3 5 6 4)
```

# TAIL CALL OPTIMIZATION

```
(defn multirember [a lat]
  (letfn
     [(multirember-aux [source sink]
                (if (seq source)
                  (if (= (first source) a)
                    (recur (rest source) sink)
                    (recur (rest source)
                           (conj sink (first source))))
                  sink))]
    (multirember-aux lat [])))

(multirember 4 '(1 2 3 4 5 6 4))
; => [1 2 3 5 6]
```

```
(take 10
     (multirember 4 (iterate inc 0)))
; Evaluation aborted.
```

# LAZINESS

```
(defn lazy-multirember [a lat]
  (letfn
     [(multirember-aux [source]
                  (lazy-seq
                   (if (seq source)
                     (if (= (first source) a)
                       (multirember-aux (rest source))
                       (cons (first source)
                             (multirember-aux (rest source))))
                     '()))))]
    (multirember-aux lat)))

(take 10 (lazy-multirember 4 (iterate inc 0)))
; => (0 1 2 3 5 6 7 8 9 10)
```

# NAMESPACES & MODULES

```
(ns example.namespace)
```

Introduces a new namespace. With def (and defn, …) stuff is added to namespaces.

```
(ns example.namespace
    (:require clojure.set))

(ns example.namespace
    (:require [clojure.set :as s]))
```

```
(ns example.namespace
    (:use clojure.string))
(ns example.namespace
    (:use [clojure.string :only [capitalize]]))
(ns example.namespace
    (:use [clojure.string :exclude [capitalize]])
```

```
(ns example.namespace
    (:import [java.util HashMap]))
```

# CALLING JAVA

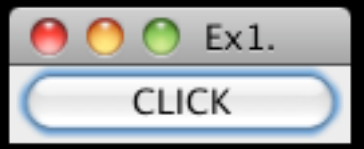
| Java | Clojure | Jython |
|---|---|---|
| import app.model.Person; | (import [app.model Person]) | from app.model import Person |
| new Person() | (new Person)<br>(Person.) | Person() |
| person.getFullName() | (. getFullName person)<br>(.getFullName person) | person.getFullName() |
| Locale.JAPAN | (. Locale JAPAN)<br>Locale/JAPAN | Locale.JAPAN |
| Integer.valueOf(“42”) | (. Integer valueOf "42")<br>(Integer/valueOf “42”) | java.lang.Integer.valueOf('42') |

# PROXIES

```
(ns gui-sample
  (:import [javax.swing JFrame JButton]
           [java.awt.event ActionListener]))

(defn build-gui [title message]
  (let [frame (JFrame. title)
        button (JButton. "CLICK")]
    (.addActionListener button
                        (proxy [ActionListener] []
                          (actionPerformed [evt]
                                           (println message))))
    (.. frame getContentPane (add button))
    (.pack frame)
    (.setVisible frame true)
    frame))
```

```
(gui-sample/build-gui
   "Rock & Roll"
   "Hello, world!")
```



# PROXIES

```
(ns gui-sample
  (:import [javax.swing JFrame JButton]
           [java.awt.event ActionListener]))

(defn build-gui [title message]
  (let [frame (JFrame. title)
        button (JButton. "CLICK")
        px (proxy [ActionListener] []
                  (actionPerformed [evt]
                                   (println message)))]
    (.addActionListener button px)
    (.. frame getContentPane (add button))
    (.pack frame)
    (.setVisible frame true) px))

(update-proxy p {"actionPerformed" (fn [this evt] (println "foo!"))})
```

# GEN-CLASS

```
(ns example.ClassExample
  (:gen-class
   :name example.ClassExample
   :extends Object
   :implements []
   :methods [
             [foo [java.util.Collection] int]
             ^{:static true} [show [java.util.Collection] void]])
  (:import (java.util Collection)))
```

# GEN-CLASS (2)

```
(defn show [coll]
  (if (seq coll)
    (do
      (print (first coll))
      (recur (rest coll)))
    (println "")))
```

```
(defn -show [coll]
  (show coll))

(defn -foo [this coll]
  (let [coll (seq coll)]
    (if coll (count coll) -1)))
```

```
>>> import example
>>> example.ClassExample.show([1, 2, 3])
123

>>> ce = example.ClassExample()
>>> ce.foo([1, 2, 3])
3
```

# IMPLEMENTING PYOBJECTS IN CLOJURE

```
(ns clj.ImmutableList
  (:gen-class
    :name clj.ImmutableList
    :extends org.python.core.PyObject
    :implements [clojure.lang.IPersistentList]
    :state state
    :init init
    :constructors {[java.util.Collection], []})
  (:import [org.python.core PyObject PyInteger Py]))

(defn -init [coll]
  [[coll] (apply list coll)])
```

```
import clj

py_list = ['a', 'b', 'c']
my_list = clj.ImmutableList(py_list)
```

# IMPLEMENTING PYOBJECTS IN CLOJURE (2)

```
(defmacro delegate-to-state [sym]
  `(defn ~(symbol (str "-" sym))
     [this#]
       (~sym (.state this#))))

(delegate-to-state peek)
(delegate-to-state pop)
(delegate-to-state count)
(delegate-to-state empty)
(delegate-to-state seq)
```

```
(defn -cons [this other]
  (cons (.state this) other))

(defn -equiv [this other]
```

```
print my_list.peek()
print my_list.pop()
print my_list.count()
print my_list.empty()
print my_list.seq()
print my_list.cons('d')
print my_list.equiv(py_list)
print my_list.equiv(['a', 'b', 'c'])
```

# IMPLEMENTING PYOBJECTS IN CLOJURE (3)

```
(defn -__finditem__ [this index]
  (let [index (if (instance? Number index)
                index
                (Py/tojava index Number))]
    (try
      (Py/java2py (nth (.state this) index))
      (catch IndexOutOfBoundsException e
        (throw (Py/IndexError (.toString e)))))))
```

```
print my_list[0]
print my_list[1]
print my_list[2]
print my_list[2.4]


try:
    print my_list[3]
except IndexError, e:
    print e
try:
    print my_list['a']
except TypeError, e:
    print e

try:
    my_list[0] = 1
except TypeError, e:
    print e
```

# CLOJURE RT

```
source = '''
(ns rember)

(defn rember [a lat]
  (cond
    (empty? lat) '()
    (= (first lat) a) (rest lat)
    :else (cons
      (first lat)
      (rember a (rest lat)))))
'''
```

```
from clojure.lang import RT, Compiler

Compiler.load(java.io.StringReader(source))

rember = RT.var('rember', 'rember')
print rember.invoke(2, range(4))
```

# CLOJURE DECORATOR

```
import pyclj

@pyclj.clojure
def rember(a, lat):
    ""(defn rember
       "Remove first occurrence of a from lat"
      [a lat] (cond
                (empty? lat) '()
                (= (first lat) a) (rest lat)
                 :else (cons (first lat)
                             (rember a (rest lat)))))""

if __name__ == '__main__':
    print rember(2, range(4))
    help(rember)
```

# IMPLEMENTATION

```
def clojure(fn):
    """Decorator that substitutes an empty python function with clojure
     in the doc with a callable which delegates to the clojure function.
    """
    clj_namespace = determine_clojure_namespace(fn)
    clojure_fnc = build_clojure_function_object(clj_namespace, fn)

    fn.__doc__ = get_docs(clojure_fnc)

    def aux(*args):
        return clojure_fnc.invoke(*args)
    functools.update_wrapper(aux, fn)

    return aux
```

# IMPLEMENTATION (2)

```
def determine_clojure_namespace(fn):
    try:
        clj_namespace = fn.__module__
    except AttributeError:
        clj_namespace = 'user'
    return clj_namespace


def get_docs(clojure_fnc):
    meta = clojure_fnc.meta()
    return meta.get(Keyword.intern('doc'))
```

# IMPLEMENTATION (3)

```
def build_clojure_function_object(clj_namespace, fn):
    clojure_code = '(ns %s)\n%s' % (
        clj_namespace,
        fn.__doc__)
    clojure_compile_string(clojure_code)
    clojure_fnc = RT.var(clj_namespace, fn.func_name)
    return clojure_fnc
```

# CALLING JYTHON FROM CLOJURE

```
(defn make-factory [modulename klassname]
  (let [interpreter (Pythor
        import-command

    (.exec interpreter impo
    (.get interpreter klassn

(def spam-and-eggs (ma

(def inst (.__call__ spam

(println inst)
(println (.invoke inst "has
```

```
class SpamAndEggs(object):
    def __init__(self, eggs):
        self.eggs = eggs

    def hasSpam(self):
        return True

    def getEggs(self):
        return self.eggs

    def __repr__(self):
        return 'Spam and %s eggs.' % self.eggs
```

# CALLING JYTHON FROM CLOJURE (PT. 2)

```
(defn make-factory
  [module klassname]
  (let [interpreter (PythonInterpreter.)
        import-command (str-join " " ["from" module
                                      "import" klassname])]
    (.exec interpreter import-command)
    (let [klass (.get interpreter klassname)]
      (fn [& args]
        (.__call__ klass
          (into-array PyObject
            (map #(Py/java2py %) args)))))))

(def spam-and-eggs (make-factory "example" "SpamAndEggs"))
(def inst (spam-and-eggs 1))
```

# THROW MACROS IN

```
(defmacro pyclass [q-class jtype]
  (let [[klass-name module-name] (split-module-and-class q-class)]
    `(def ~(symbol klass-name)
      (make-factory ~(str module-name) ~(str klass-name) ~jtype))))

(pyclass example.PrintSomething java.awt.event.ActionListener)

(def evt-printer (PrintSomething))
```

# MAKE-FACTORY (AGAIN)

```
(defn make-factory
  [module klassname interface]
  (let [interpreter (PythonInterpreter.)
        import-command (str-join " " ["from" module "import" klassname])]
    (.exec interpreter import-command)
    (let [klass (.get interpreter klassname)]
      (fn [& args]
        (.__tojava__ (.__call__ klass
          (into-array PyObject
            (map #(Py/java2py %) args)))
          interface)))))
```

# BACK TO THE BEGINNING

```
(defn build-gui [title]
  (let [frame (JFrame. title)
        button (JButton. "CLICK")]
    (.addActionListener button (PrintSomething))
    (.. frame getContentPane (add button))
    (.pack frame)
    (.setVisible frame true)
    frame))
```

## CONCLUSION

# Thanks for Your Kind Attention

https://github.com/rik0/PyCLJ_Examples
https://github.com/rik0/pyclj