Improving the Interoperability of Java and Clojure

Stephen Adams

University of Minnesota: Morris

April 14th, 2012

A little reminder about Java

An object-oriented language released by Sun Microsystems in 1995.

- Compiled to Java bytecode
- Java bytecode is run on the Java Virtual Machine
- Second language in TIOBE programming Community Index (as of April 2012)

Clojure was released in 2007 by the software developer Rich Hickey. Clojure was designed with four features in mind:

Clojure was released in 2007 by the software developer Rich Hickey. Clojure was designed with four features in mind:

The Four Features of Clojure

- A Lisp
- Functional programming
- Symbiosis with an established platform
- Designed for concurrency

LISP and Functional Programming

- Lisp was developed in 1958
- The earliest functional language
- Emphasizes the application of functions
- Imperative programming emphasizes changes in state

Functional Programming and concurrency

- Functional languages have immutable data
- No state = no side effects
- A function will always return the same results given the same arguments
- Easier to predict behavior of a program in a concurrent environment

Who this talk is for

- Need to parallelize your work?
- Are you tired of Java?

Who this talk is for

- Need to parallelize your work?
- Are you tired of Java?
- Have a project already written in Java?
- Have a Java library you use a lot?

Who this talk is for

"Clojure does Java better than Java" - Stuart Halloway, at the
Greater Atlanta Software Symposium, 2009.

1 **Introduction**

2 **Introduction to Clojure**

3 **Functional Programming in Clojure**

4 **Java Interop**
  ● Basic Java calling
  ● Java Objects in Clojure
  ● Custom types in Clojure

5 **Conclusion & References**

Prefix Notation

```
(+ 2 3)
=> 5
```

Prefix Notation

```
(+ 2 3)
=> 5

(+ 2 3 4)
=> 9
```

Clojure Data Structures

- Many of Clojure's data structures are just Java data structures; strings, characters, and all numbers are just Java types.

Clojure Data Structures

- Many of Clojure's data structures are just Java data structures; strings, characters, and all numbers are just Java types.
- Clojure provides its own collections.

Every Clojure collection is denoted by a different literal symbol
pair.

Collection Literals

List	(1 2 3 4)
Vector	["apple" "banana" "orange"]
Hashmap	{ :name "Stephen Adams" :phone 555555555 }

Keywords

- Symbolic identifiers, denoted with a leading colon
- The colon is not considered part of the keyword
- Keywords evaluate to themselves
- Very fast equality tests

```
:keyword
=> :keyword
```

Variables and Functions

```
(def vect [1 2 3 4 5])

(defn square [x]
    (* x x))
```

Namespaces

```
(ns some.SampleNamespace
   (:import javax.swing.JFrame))
```

- Provide a symbolic identifier for pieces of Clojure code

The functional features of Clojure

Functional programming primarily refers to two language features:

- First class functions
- Anonymous functions

Clojure supports both of these features.

First Class Functions
Passing functions to other functions

```
(defn square [x]
    (* x x))
```

First Class Functions
Passing functions to other functions

```
(defn square [x]
    (* x x))

(map square [1 2 3 4 5])
=> [1 4 9 16 25]
```

First Class Functions
Passing functions to other functions

```
(defn square [x]
    (* x x))

(map square [1 2 3 4 5])
=> [1 4 9 16 25]

(reduce + [1 2 3 4 5])
=> 15
```

Anonymous Functions

```
(defn all-same? [vect]
 (if (empty? vect)
  true
  (every?
   (fn [x] (= (first vect) x )) (rest vect))
```

Introduction to Java Interop

The idea for Clojure always involved interoperability with an existing language. Java was chosen for various reasons:

- Access to previously written Java libraries
- Already implemented, garbage collection and other memory & resource management tools.
- JVM is OS agnostic.

Basic Java calling

The dot special form

- Provides basic access to Java fields and methods
- Can also be read as "in the scope of."

```
(. "fred" toUpperCase)
=> "FRED"
```

Basic Java calling

The dot special form cont.

```
(.toUpperCase "fred")
```

Basic Java calling

The dot special form cont.

```
(.toUpperCase "fred")
```

Accessing Static methods and fields

```
(. Math PI)
=> 3.141592653589793

(Math/PI)
=> 3.141592653589793
(Math/abs -2)
=> 2
```

Object Construction and modification

```
(new StringBuffer "fred")
=> #<StringBuffer fred>
```

Java Objects in Clojure

Object Construction and modification

```
(new StringBuffer "fred")
=> #<StringBuffer fred>

(doto (new StringBuffer "fred")
(.setCharAt 0 \F)
(.append " is a nice guy!"))
=> #<StringBuffer Fred is a nice guy!>
```
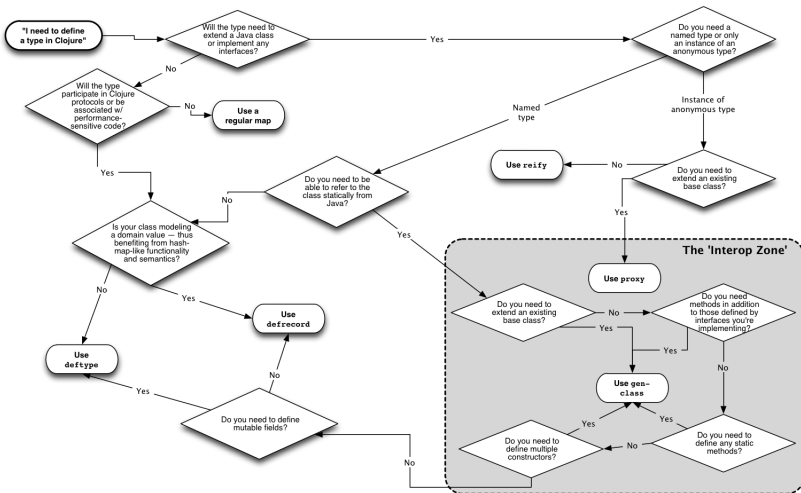
Java Objects in Clojure

Object Construction and modification cont.

```
public class ExampleBuffer {
   public static void main(String[] args){
      StringBuffer buff = new StringBuffer("fred");
      buff.setCharAt(0, "F");
      buff.append(" is a nice guy");
   }
}
```
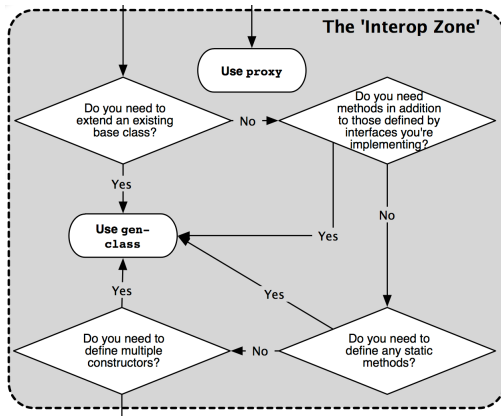
Custom types in Clojure

Many ways of defining a type

By Chas Emerick, from: http://bit.ly/IiozRP

Custom types in Clojure

Many ways of defining a type

Custom types in Clojure

The Java interop zone

Clojure functions designed to work with existing Java code.

The interop zone

- proxy
- gen-class

Proxy

- Must implement a Java interface or extend a Java class
- Creates a single instance of an anonymous Java class
- Cannot define methods not declared by a superclass or interface

Proxy cont.

```
public interface TestInterface {
int square(int x);
}
```

Outline   Introduction   Introduction to Clojure   Functional Programming in Clojure   **Java Interop**   Conclusion & References
○○○○○○○○○●○○○○

Custom types in Clojure

Proxy cont.

```
public interface TestInterface {
int square(int x);
}

(def test-inter
  (proxy [TestInterface] [] (square [x] (* x x))))
```

Proxy cont.

```
public interface TestInterface {
int square(int x);
}

(def test-inter
  (proxy [TestInterface] [] (square [x] (* x x))))

(. test-inter square 5)
=> 25
```

Outline   Introduction   Introduction to Clojure   Functional Programming in Clojure   **Java Interop**   Conclusion & References
○○○○●○○○○○●○○○

Custom types in Clojure

Proxy cont.

```
(defn add-mousepressed-listener
    [component f args]
      (let [listener (proxy [MouseAdapter] []
                       (mousePressed [event]
                         (apply f event args)))]
      (.addMouseListener component listener)
    listener))
```

Gen-class

- Proxy will only allow you to do so much because you can only overload functions defined by a super class or interface.

Outline Introduction Introduction to Clojure Functional Programming in Clojure **Java Interop** Conclusion & References
○○○○○○○○○○○●○○

Custom types in Clojure

Gen-class

- Proxy will only allow you to do so much because you can only overload functions defined by a super class or interface.
- There are cases when defining your own Java methods and objects is necessary.
- For instance working with a library that requires you to extend some object.

Custom types in Clojure

Gen-class cont.

```
(ns some.Example
  (:gen-class
      :prefix "method-"))

(defn method-toString
  [this]
  "Hello, world!")
```

Custom types in Clojure

Gen-class cont.

```
(ns some.Example
  (:gen-class
      :prefix "method-"))

(defn method-toString
  [this]
  "Hello, world!")


(def aClass (new some.Example))
(.toString aClass)
=> "Hello, world!"
```

```
(ns some.Example)

(gen-class
  :name   some.Example.classA
  :prefix classA- )

(gen-class
  :name some.Example.classB
  :prefix classB- )

(defn classA-toString
  [this]
  "I'm an A.")

(defn classB-toString
  [this]
  "I'm an B.")
```

Java - Clojure Relationship

"Clojure does Java better than Java" - Stuart Halloway, at the
Greater Atlanta Software Symposium, 2009.
Proxy is an example of this.

Java - Clojure Relationship

- You want to program Clojure in Clojure, not Java.
- Gen-class and, to some extent, proxy break from Clojure-like syntax.
- These functions should be used sparingly.
- C. Emerick's figure (http://bit.ly/IiozRP).

Recommendations

- Push Clojure's native abstractions into interop zone.
- Centralize documentation sources.
- Streamline IDE setup for beginners.

Why this matters

"The killer app for Clojure is the JVM itself. Everyone's fed up
with the Java language, but understandably we don't want to
abandon our investment in the Java Virtual Machine and its
capabilities: the libraries, the configuration, the monitoring, and
all the other entirely valid reasons we still use it."
Steve Yegge, in the forward to Joy of Clojure.

References

- CLOJUREDOCS.ORG. Mar. 2012. [Online; accessed March-2012].
- CLOJURE.ORG. Mar. 2012. [Online; accessed March-2012].
- Fogus, M., and Houser, C. The Joy of Clojure, Manning Publications.
- Halloway, S. Clojure-Java interop: A better Java than Java. QCon.