Using Java from Clojure

Cosmin Stejerean
PSC Group, LLC

Importing Java Types

- (import `com.example.SomeClass)
- (import (`com.example `SomeClass))
- ;; no way to import *
- (import (`com.example [`ClassOne `ClassTwo])

Creating Objects

- (new ClassName args*)
- (ClassName. args*) ;; note the . at the end

Methods and Fields

- (. object methodName args*) ;; call method
- (. object field) ;; get value of field
- (set! (. object fieldName) value) ;; set field

Lisp-like syntax

- (.fieldName object)
- (.methodName object args*)
- Class/staticField
- (Class/staticMethod args*)

The dot-dot macro

- (.. obj (method1 args*) field (method2 args*))

  same as

- obj.method1(args*).field.method2(args*)

The doto macro

- ;; applies all functions to the given object

-
  ```
  (doto (new java.util.HashMap)
      (.put “a” 1)
      (.put “b” 2))
  ```

Beans

- (bean object) ;; get JavaBean properties

Working with Arrays

- (alength array)
- (aget array index+)
- (aset array index+ value)

Creating arrays

- (make-array class dim+)
- (to-array collection) ;; array of objects
- (to-array-2d collection-of-collections)
- (into-array collection) ;; array of first type

Arrays of primitives

- special constructors for arrays of primitives
  - float-array, int-array, etc...
- type hints for arrays of primitives
  - #^ints, #^floats, #^longs, etc...

Primitive coercion

- (int x) (float x)
- (double x) (short x)
- (char x) (byte x)
- (boolean x)

First class Java functions with memfn

- (def fn1 (memfn (methodName arg-names*)))
- (apply fn1 (object args*))
- useful for use with map, reduce, etc...

Creating types

-
  ```
  (proxy [class-and-interfaces] [constructargs*]
     (name1 [paramlist] body)
     (name2 ([paramlist1] body1)
                 ([paramlist2] body2)))
  ```

Proxy limitations

- No access to protected methods
- No access to super

gen-class

- at compilation creates a Java class
- does not have the same limitations as super