Clojure Basic Training - Module 9

Object Oriented Clojure

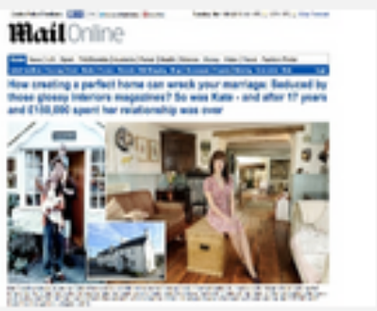
New Tab

how to quit

how to quit - Google Search
how to quit **smoking**
how to quit **a part time job**
how to quit **smoking weed**
how to quit **sugar**
how to quit **your job**

Search Google or type URL

Home | Daily Mail Onli



MailOnline

Il Fatto Quotidiano - N

Pinboard: bookmarks

Wait what?

- No classes or objects as such, but:
- Layers of functions can create an interface
- When same interface needs multiple implementations
- Long “conds” or checking types is a good indicator

Type based dispatch

- Clojure is dynamic but strictly typed
- Types can implement one or more protocols
- Function dispatch can then depend on types

Protocols

```
(defprotocol ProtocolName
  “documentation here"
  (a-fn [this arg1 arg2] "docstring”)
  (another-fn [this] [this arg] "docstring"))
```

- Protocol name followed by list of empty fns
- Supporting multiple “arities”
- Only camel-case convention in Clojure
- Dispatch happend by type only

Protocol Example

```
(defprotocol JSON
  (to-json [this]))

(extend-protocol JSON

  java.lang.Long
  (to-json [this]
    (str this))

  java.lang.String
  (to-json [this]
    (str "\"" this "\""))

  nil
  (to-json [this]
    "null")))
```

- Extends existing types
- Extended to nil as well
- “this” is the protocol instance itself

```
user=> (to-json "1")
"\"1\""
user=> (to-json 1)
"1"
```

Examples

- “Themes” (classic strategy pattern)
- “Renderers” (fetch, render)
- “Components” framework

Multimethods

```
(defmulti fill
  “dispatch function"
  (fn [node value] (:tag node)))

(defmethod fill :div
  [node value]
  (assoc node :content [(str value)]))
(defmethod fill :input
  [node value]
  (assoc-in node [:attrs :value] (str value)))
(defmethod fill :default
  [node value]
  (assoc node :content [(str value)]))
```

```
$> (fill {:tag :div} "hello")
{:content ["hello"], :tag :div}
$> (fill {:tag :input} "hello")
{:attrs {:value "hello"}, :tag :input}
$> (fill {:span :input} "hello")
{:content ["hello"], :span :input}
```

- “defmulti” is called dispatch function
- Each defmethod define a different behaviour
- :default is default behaviour
- In this case dispatching on “(:tag node)”

More Examples

- Standard modules (key dispatch)
- Modules (key dispatch)
- Video, channel, carousel…

Resources

- “Polymorphism in Clojure” http://blog.8thlight.com/myles-megyesi/2012/04/26/polymorphism-in-clojure.html
- “Clojure Programming” book
- Clojure Koans https://github.com/functional-koans/clojure-koans