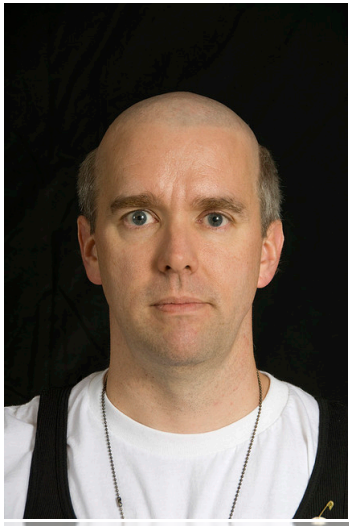
# ‘Object-Oriented’ Clojure

by ck

# !!! WARNING !!!



I am influenced by these people!


# Object-Orientation

# Object-Orientation has …

- encapsulation
- inheritance
- polymorphism
- mutable objects
- polymorphism baked into inheritance
- interfaces are optional
- implementation inheritance

# OO Design
# =
# Create (Mini) DSL

# Clojure on the other hand ...

- public fields
- polymorphism through protocols
- immutable objects
- polymorphism separate from inheritance
- interfaces are mandatory
- no implementation inheritance

# Records

# Maps

```
(def m {:a 1 :b 2 :c 3})

(m :b) ;also (:b m)
=> 2

(keys m)
=> (:a :b :c)

(assoc m :d 4 :c 42)
=> {:d 4, :a 1, :b 2, :c 42}

(dissoc m :d)
=> {:a 1, :b 2, :c 3}

(merge-with + m {:a 2 :b 3})
=> {:a 3, :b 5, :c 3}
```

```
(def jdoe {:name "John Doe"
           :address {:zip 27705, ...}})

(get-in jdoe [:address :zip])
=> 27705

(assoc-in jdoe [:address :zip] 27514)
=> {:name "John Doe", :address {:zip 27514}}

(update-in jdoe [:address :zip] inc)
=> {:name "John Doe", :address {:zip 27706}}
```

# Records

```
(defrecord Bar [a b c])

(def b (Bar. 5 6 7))

(baz b)
=> "Bar baz 7"

(:b b)
=> 6

(class b)
=> user.Bar

(supers (class b))
=> #{clojure.lang.IObj clojure.lang.IKeywordLookup java.util.Map clojure.lang.IPersistentMap
 clojure.lang.Associative java.lang.Object java.lang.Iterable clojure.lang.ILookup
 clojure.lang.Seqable clojure.lang.Counted clojure.lang.IMeta}
```

# From Maps ...

```
(def stu {:first-name "Stu"
          :last-name "Halloway"
          :address {:street "200 N Mangum" :city "Durham" :state "NC" :zip 27701}})

(:last-name stu)
=> "Halloway"

(-> stu :address :city)
=> "Durham"

(assoc stu :first-name "Stuart")
=> {:first-name "Stuart", :last-name "Halloway", :address ...}

(update-in stu [:address :zip] inc)
=> {:address {:street "200 N Mangum", :zip 27702 ...} ...}
```

# ... to Records

```
(defrecord Person [first-name last-name address])
(defrecord Address [street city state zip])
(def stu (Person. "Stu" "Halloway"
                  (Address. "200 N Mangum" "Durham" "NC" 27701)))

(:last-name stu)
=> "Halloway"

(-> stu :address :city)
=> "Durham"

(assoc stu :first-name "Stuart")
=> :user.Person{:first-name "Stuart", :last-name "Halloway", :address ...}

(update-in stu [:address :zip] inc)
=> :user.Person{:address {:street "200 N Mangum", :zip 27702 ...} ...}
```

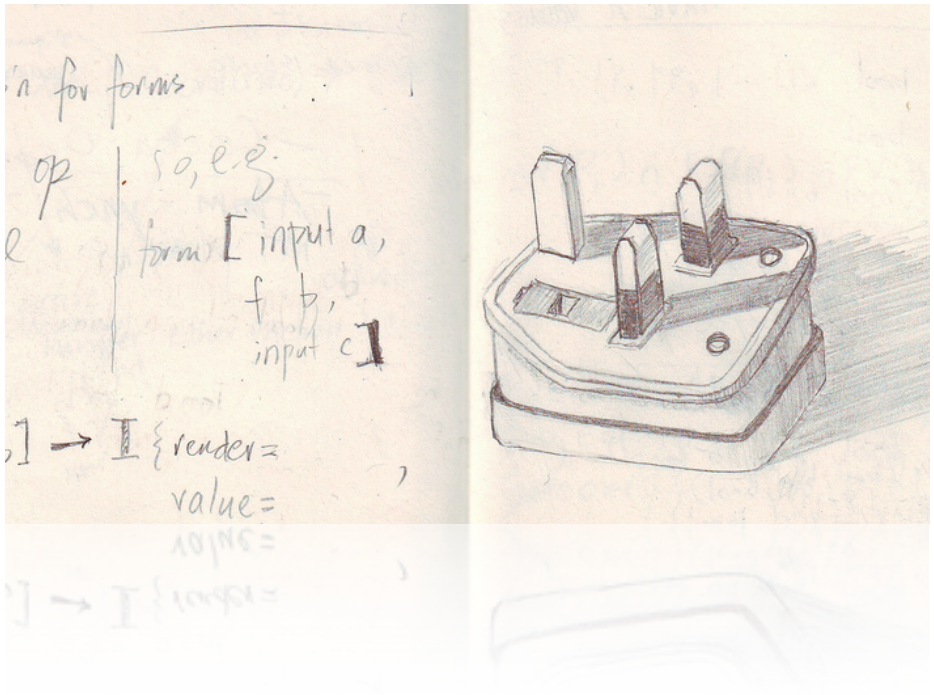
# Maps vs. Record

```
(:last-name stu)
=> "Halloway"

(-> stu :address :city)
=> "Durham"

(assoc stu :first-name "Stuart")
=> {:first-name "Stuart", :last-name
"Halloway", :address ...}


(update-in stu [:address :zip] inc)
=> {:address {:street "200 N
Mangum", :zip 27702 ...} ...}
```

```
(:last-name stu)
=> "Halloway"

(-> stu :address :city)
=> "Durham"

(assoc stu :first-name "Stuart")
=> :user.Person{:first-name
"Stuart", :last-name
"Halloway", :address ...}

(update-in stu [:address :zip] inc)
=> :user.Person{:address {:street "200
N Mangum", :zip 27702 ...} ...}
```

# its all data

use objects if you like




# Protocols

# Protocol

```
(defprotocol Foo
    "A doc string for Foo abstraction"
    (bar [this a] "bar docs")
    (baz [this] "baz docs"))
```

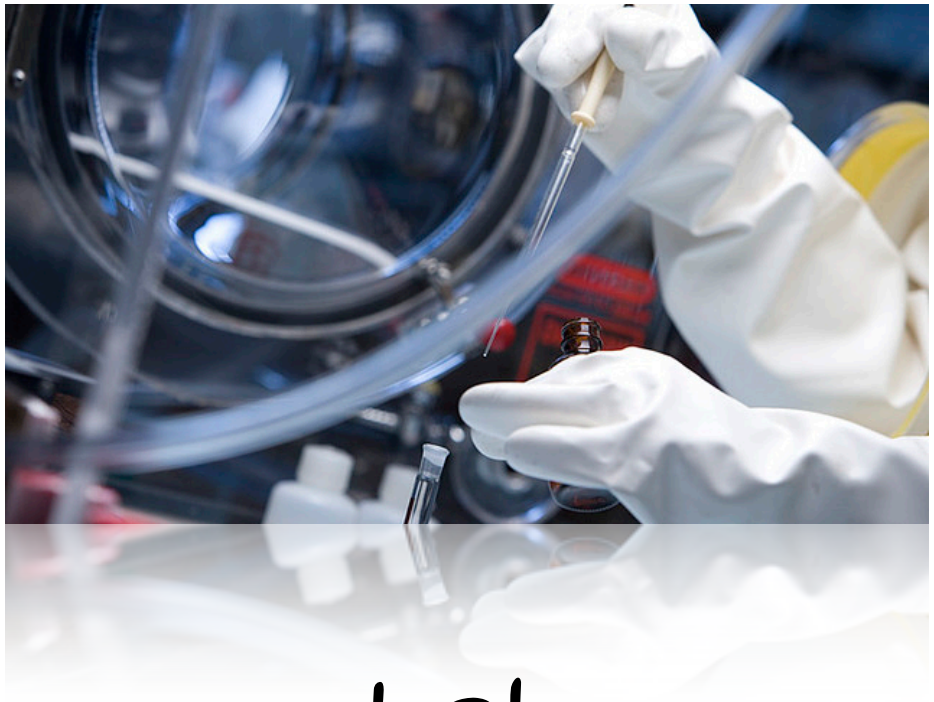
- named set of generic functions
- polymorphic on type of first argument
- no implementation
- define functions in same namespace as protocol

# Implementing a Protocol

```
(defrecord Bar [a b c]
  Foo
  (bar [this b] "Bar bar")
  (baz [this] (str "Bar baz " c)))

(def b (Bar. 5 6 7))

(baz b)
=> "Bar baz 7"
```

# Extending a Protocol

```
(baz "a")

java.lang.IllegalArgumentException:
No implementation of method: :baz
of protocol: #'user/Foo
found for class: java.lang.String

(extend-type String
  Foo
  (bar [this s2] (str s s2))
  (baz [this] (str "baz " s)))

(baz "a")
=> "baz a"
```



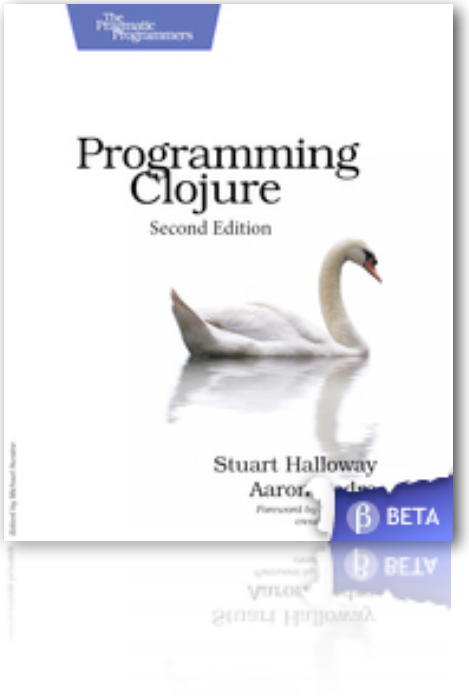
# Lab

http://github.com/pittsburghclj/rock-paper-scissors

# There is more …

- deftype
- reify
- multi-methods

# Resources





Relevance LabREPL