Vad clojure lärt mig om objektorientering

(som jag inte lärt mig av Java)

Jag

agical

Ville Svärd

```
{
    :twitter     @villesv
    :email       ville.svard@agical.com
    :lojalitet   Agical
    :skador      [ Clojure, TDD ]
}
```

agical

Nyfikenhet

"I'd leave out classes"

- James Gosling

(åtminstone om man får tro Allen Holub)

(clojure)

(Beware, here be parens)

Clojure metodanrop

Java

```
object.method(arg)
```

Clojure

```
(method object arg)
```

Hello World

```
(defn hello-world []
    (println "Hello" "world"))

(hello-world)

=> Hello world
   nil
```

Inkapsling

Skydda

Abstrahera

Inkapsling

(Java)

```
public class Capsule {
    private int seeds;

    public int getSeeds() {
        return seeds;
    }

    public void setSeeds(int seeds) {
        this.seeds = seeds;
    }
}
```

Inkapsling

(Java)

```
Capsule c = new Capsule();
c.setSeeds(42);
c.getSeeds();
```

Inkapsling

(Clojure)

```
{:seeds 0}
;; or
(defrecord Capsule [seeds])
```

Inkapsling

(Clojure)

```
(def c {:seeds 0})
;; or
(def c (Capsule. 0))

;; regardless
(:seeds c)
=> 0
```

Inkapsling

(Clojure)

Inga setters!

Inkapsling

(Clojure)

Accessor är en funktion

Abstraktion!

Inkapslar associativt beteende

```
(map :seeds [(Capsule. 14)
             (Capsule. 38)])

=> (14 38)
```

Polymorfism

Lägga till beteende till data

Expression problem

"...

add new cases to the datatype and new functions over the datatype,

..."

--Philip Wadler, 1998

Expression problem

Objektorientering gör det lätt att lägga till nya datastrukturer

	.size	**.add**	**.get**	**.clear**	***SVÅRT!***
ArrayList					
LinkedList					
Existerande implementation					
Stack					
Ny (data)typ					

Expression problem

Funktionell programmering gör det lätt att lägga till nya funktioner

	count	**conj**	**nth**	**empty**	***Ny funktion***
list					
vector					
Existerande implementation					
map					
SVÅRT!					

I clojure?

Protocols!

Definiera...

```
;; New function(ality)
(defprotocol Ordered
  (ordered? [this]))
```

...och lös problemet!

```
;; Implement for existing type
;; (without touching it!)
(extend-type java.util.ArrayList
  Ordered
  (ordered? [this] true))

;; or, even better,
(extend-type java.util.List
  Ordered
  (ordered? [this] true))

(ordered? (java.util.ArrayList.))
=> true
```

...och lös problemet!

```
;; New type
(deftype Bucket [...]
  ;; Implements existing functionality
  Ordered
  (ordered? [this] false)
  ...
  )
```

```
OO 'goes to eleven'
```

Separation of concerns

Räds ej

Inga klasser eller hierarkier skadades

under förberedandet av detta tal

Lärdomar

Arv är ingen förutsättning

Klass är ingen förutsättning

"I'd leave out classes"

För den diskussionsvillige:

```
{
    :twitter    @villesv
    :email      ville.svard@agical.com
}
```

Attributions

- Stuart Sierra, "Solving the expression problem with Clojure 1.2"
- Rich Hickey, "Clojure Protocols"
- Wikipedia
- Debasish Gosh, Random Thoughts On Clojure Protocols