U09LZR36F : I figured it would be useful for that, yes
U628K7XGQ : Going back to my question from this morning to introspect a Java object, here's a fn I've written to expose the fields of an object via reflection (clojure.reflect really didn't help).I've crammed it all into a single fn and it looks quite ugly. Any suggestions on how to improve the code style. I'm still a clnewb?

```
(defn java-&gt;map
  "Turns fields of a Java object into a map, up to 'level' deep"
  ([obj] (java-&gt;map obj 1))
  ([obj level]
   (when (some? obj)
     (let [c (class obj)]
       (cond
         ;;;(.isPrimitive c) obj never works because clojure implicitly wraps primitives
         (contains?
           #{java.lang.Long java.lang.Character java.lang.Byte
             java.lang.Double java.lang.Float java.lang.Short java.lang.Integer} c) obj
         (= 0 level) (.toString obj)
         (instance? java.lang.String obj) obj
         (.isArray c) (concat
                        (-&gt;&gt; obj
                             (take 5)
                             (map (fn [e] (java-&gt;map e (dec level)))))
                        (when (&gt; (count obj) 5) [:more (count obj)]))
         :else
         (assoc (into {} (-&gt;&gt; (concat (.getDeclaredFields c)
                                      (.getFields c))
                              (filter #(= (bit-and (.getModifiers %) java.lang.reflect.Modifier/STATIC) 0)) ;;; ignore static fields
                              (map
                                #(do (.setAccessible % true)
                                     [(keyword (.getName %))
                                      (java-&gt;map (.get % obj) (dec level))]))))
           :-type c  ;;; add the type as well
           ))))))
```

Examples:
```
(java-&gt;map (java.util.Date.) 4)
=&gt; {:fastTime 1500771066642, :cdate nil, :-type java.util.Date}

(java-&gt;map (java.text.AttributedString. "bubu") 2)
=&gt;
{:text "bubu",
 :runArraySize 0,
 :runCount 0,
 :runStarts nil,
 :runAttributes nil,
 :runAttributeValues nil,
 :-type java.text.AttributedString}
```

U2TCUSM2R : I'm having trouble adding metadata inside a macro. It's a bit weird (as usual) since it overrides `defn` in order to capture the ast after compilation, but works otherwise: ```(defmacro defn [name &amp; decls] `(def ^{:ast ~decls} ~name (fn ~decls)))```
U051SS2EU : defmacro never sees the reader macro, it's applied before it sees the form
U051SS2EU : you need to use with-meta or vary-meta to add metadata to the symbol you emit
U2TCUSM2R : I did try that, but perhaps didn't do it correctly
U2TCUSM2R : (and good call because I definitely don't want this done at runtime)
U051SS2EU : `^{:foo :bar} baz ` translates to `(with-meta baz {:foo :bar})`

U2TCUSM2R : right
U2TCUSM2R : lemme give it another go and post the results if it doesn't work
U2TCUSM2R : ok, so yeah so for some reason the actual `fn` part doesn't compile when i write it like that: ```(defmacro defn [name &amp; decls] `(def ~(with-meta ~name {:ast ~decls}) (fn ~decls)))```
U051SS2EU : nested ~ isn't valid like that
U051SS2EU : it needs to be one of those positions or the other, not both
U2TCUSM2R : oh. i tried unquote splicing it, but then got an error on the macro
U051SS2EU : splicing isn't valid there either
U2TCUSM2R : is it enough to just unquote the expression? that works
U2TCUSM2R : but same error when calling the macro
U051SS2EU : the next problem is that decls isn't going to be a valid data literal
U051SS2EU : so it needs to be escaped or quoted in some manner
U2TCUSM2R : hmm. it worked fine without the metadata
U051SS2EU : right but the decls are a data literal in the metadata - so they need to be a valid one
U051SS2EU : ```+user=&gt; (defmacro defn [name &amp; decls] `(def ~(with-meta name {:ast (cons 'quote decls)}) (fn ~decls)))#'user/defn
+user=&gt; (defn baz [])
#'user/baz
+user=&gt; (meta #'baz)
{:ast [], :line 19, :column 1, :file "NO_SOURCE_PATH", :name baz, :ns #object[clojure.lang.Namespace 0x373ebf74 "user"]}
user=&gt;
```

U2TCUSM2R : oh that one
U2TCUSM2R : i confused the error message
U051SS2EU : wait I think cons ins the wrong way to do that... checking
U051SS2EU : yeah, my bad ```+user=&gt; (defn baz ([]) ([_]))CompilerException clojure.lang.ExceptionInfo: Wrong number of args (2) passed to quote {:form (quote ([]) ([_]))}, compiling:(NO_SOURCE_PATH:21:1)
```

U051SS2EU : this fixes an issue with multi arities in the original too ```+user=&gt; (defmacro defn [name &amp; decls] `(def ~(with-meta name {:ast (cons 'quote (list decls))}) (fn ~@decls)))#'user/defn
+user=&gt; (defn baz ([]) ([_]))
#'user/baz
+user=&gt; (meta #'baz)
{:ast (([]) ([_])), :line 25, :column 1, :file "NO_SOURCE_PATH", :name baz, :ns #object[clojure.lang.Namespace 0x373ebf74 "user"]}
```

U051SS2EU : there's probably a better way to rewrite`(cons 'quote (list decls))` - that's super ugly
U2TCUSM2R : it works if i just do `(list 'quote decls)`
U051SS2EU : oh, right, much nicer
U2TCUSM2R : wow, thanks for explaining that to me as always
```