U11BV7MTK : That's awesome. It's just so handy. Much appreciated
U61KCTX8S : can anyone explain me what a java.lang.IllegalArgumentException: array element type mismatchmeans?

U0NCTKEV8 : it means the array type you are calling a java method with doesn't match
U61KCTX8S : so fi it expects an array of ints and i am using an array of strings
U61KCTX8S : thanks
U0CMVHBL2 : Yes, there is. You can verify yourself whether this is so using identical? on the pieces you hope are being shared, as shown below
U0CMVHBL2 : user=> (def mymap {:mykey [{:A 1, :B 2} {:A 2, :B 2} {:A 3, :B 2}]})#'user/mymap
user=> (def m2 (assoc mymap :mykey (map #(assoc % :A (inc (:A %))) (:mykey mymap))))
#'user/m2
user=> (identical? (-> mymap :mykey (nth 0) :B) (-> m2 :mykey (nth 0) :B))
true

U0DATSMH6 : Hmm, my gut tells me that's not a good example because: ```user=> (identical? 2 2)
true
```

U0DATSMH6 : This would be a better test:```
user=&gt; (identical? {:test 1} {:test 1})
false
user=&gt; (def mymap {:mykey [{:A 1, :B {:test 1}} {:A 2, :B {:test 1}} {:A 3, :B {:test 1}}]})
#'user/mymap
user=&gt; (def m2 (assoc mymap :mykey (map #(assoc % :A (inc (:A %))) (:mykey mymap))))
#'user/m2
user=&gt; (identical? (-&gt; mymap :mykey (nth 0) :B) (-&gt; m2 :mykey (nth 0) :B))
true
```

U0CMVHBL2 : The better test works, too, as I expected it would. Thanks.
U0DATSMH6 : Also, for clarity, there isn't any structural sharing going on _within_ the vectors themselves - only at the same key "paths" between `mymap` and `m2`:```
user=> (identical? (-> mymap :mykey first :B) (-> mymap :mykey last :B))
false
user=> (identical? (-> m2 :mykey first :B) (-> m2 :mykey last :B))
false
user=> (-> mymap :mykey first :B)
{:test 1}
user=> (-> mymap :mykey last :B)
{:test 1}
```

U0DATSMH6 : Yeah - this is a cool approach to discover the structural sharing. I hadn't thought of using `identical?` for this.
U3JURM9B6 : keep, for, map -- they want pure functions and return a lazy list
U3JURM9B6 : I want something which is okay to pass an unpure function to ... and returns a strict list
U3JURM9B6 : in good clojure style, do people do (map impure-function ...) or do we do something else when we have to run an impure function and also get its return value ?
U051KLSJF : <@U3JURM9B6> usually you'll use `doall`
U051KLSJF : if you care about the return value
U1ALMRBLL : <@U3JURM9B6> it is fine, as far as I know, to pass a function with side-effects to `keep`, `filter`, etc -- the gotcha is that you should not *expect* that your code will necessarily execute. So, your side effect may happen, and if you don't mind, great -- but, your side effect might *not* happen, and that's why it says to avoid impure functions (or in some cases, may get called *more than once*).
U051SS2EU : it also might happen 31 times more than you expected
U051SS2EU : right
U1ALMRBLL : <@U3JURM9B6> not sure why you want a "strict list", but for eager evaluation with guaranteed "run-once", `reduce` would be a decent candidate to build it:```(defn eager-map [f coll]
  (reverse (reduce #(conj % (f %2)) () coll)))```
```

U1C03090C : Does anyone know how to specify a stylesheet for a scene in fn-fx (a clojure wrapper for JavaFx)? <https://github.com/halgari/fn-fx>
U3JURM9B6 : <@U051KLSJF> , <@U1ALMRBLL> : okay, so despite the docs saying "use pure function", it's actually okay to assume *atmost once, in order* semantics -- i.e. we don't know how far it will exec (due to evaling thunks at a time) -- but we can assume that:
1. thunks, if evaluated, at evaluated at most once
2. thunks, if evaluated, as "in order"
so
(head (map f '(1 2 3 4 5)))

won't do

(f 1)
(f 3)
eh, let's skip 3 & 4 ... and then just eval (f 5) just for kicks

U46LFMYTD : hey <@U0DATSMH6> , <@U0CMVHBL2> , I didn't know that `identical?` could be uesd in such a way to examine structural sharing - thanks!
U1ALMRBLL : <@U3JURM9B6> in the case of the current implementation of `map`, I'd say so -- with the obvious disclaimer that making assumptions like this and relying on these details is not safe, and that I wouldn't use an approach like this (I'd prefer the `eager-map` function I put above, for example)in the case of something like a comparator, it's quite likely that it will be executed more than once, per pair, so you would definitely not get any guarantees there.

U5JEJN1CP : I'm getting a bizarre repl error that occurs only when reloading a file:```
2. Unhandled clojure.lang.Compiler$CompilerException
   Error compiling *cider-repl webtools* at (1:1)

1. Caused by java.lang.RuntimeException
   No such var: user/reset
```
The project compiles fine.  The repl starts fine.  The problem only happens if I use `C-c C-x` to recompile.  If I restart the whole repl, it compiles fine.  It seems like it's trying to compile the repl buffer, but I can't for the life of me fathom WHY.  Wondering if anyone has any ideas what would cause this.  The only hit on stack overflow suggests that I must be requiring something `:as user`, but I'm definitely not.

U3JURM9B6 : <@U1ALMRBLL> : thanks for the eager map; it seems surprising that 1. core has no eager-map 2. many are probaly using (doall (map ...)) to simulate it, but 3. this revolves around assumptions that map doesn't really guarantee
U5JEJN1CP : Tried a `lein clean` but that didn't make any difference.
U3JURM9B6 : <@U5JEJN1CP> : what is user/reset ? is it part your lein config, part of cider, part of some reload package that you are using? [I use boot]
U5JEJN1CP : <@U3JURM9B6> User is the init-ns for a repl in my project.clj (lein config).
```