U053XQP4S : the implicit aspect looks a bit dangerous to me, and the only benefit I see is to save a few characters when you write your functions
U0K064KQV : Question: How would I extend a protocol to all array types? I seem to only be able to do it for a specific array, like say "[Ljava.lang.Object". But I want "[L?"
U0NCTKEV8 : there is no such thing
U0K064KQV : :disappointed: Its strange that "[Ljava.lang.Object" doesn't even work for subtypes of Object. It only works if I have actual array of Objects
U0NCTKEV8 : that is how array types work on the jvm
U0NCTKEV8 : array types don't have that kind of type relation
U0NCTKEV8 : if A is an array of X and B is an array of Y, and Y is a subtype of X, B is not a subtype of A
U0K064KQV : I'm not convinced, since instanceof can tell the relation
U060FKQPN : <@U0NCTKEV8> java array are covariant, it's generics that aren't
U0NCTKEV8 : Oh
U050MP39D : yeah I might be missing something but I get no compile error```
```
11326-storage:tmp bfabry$ cat &gt; Foo.java
public class Foo {
  public Object[] fooey;
  public Foo() {
    fooey = new String[10];
  }
}
11326-storage:tmp bfabry$ javac Foo.java
11326-storage:tmp bfabry$
```

U0K064KQV : =&gt; IllegalArgumentException No implementation of method: :t of protocol: #'special.eagerize-test/Table found for class: [Ljava.lang.String;
U0K064KQV : My solution was to extend java.lang.Object, and do: (when (instance? (Class/forName "[Ljava.lang.Object;") &lt;implementation-for-arrays&gt;)
U0K064KQV : (when (instance? (Class/forName "[Ljava.lang.Object;") this) &lt;implementation-for-arrays&gt;)
U0K064KQV : Question: What is in clojurescript the type I need to extend to cover all types? Equivalent to java.lang.Object say?
U0K064KQV : Ok, clojurescript has default for that, awesome. I actually wished Clojure had that too.
U0K064KQV : I opened an issue into it: <https://dev.clojure.org/jira/browse/CLJ-2215>
U3QUAHZJ6 : hello everyone, how im supposed to create a spec where all keys are optional but at least one of the specified keys should be present?
```
(s/def ::my-spec (s/and (help-plz??)(s/keys :opt-un [::a ::b])))
(s/valid? ::my-spec {} =&gt; false
(s/valid? ::my-spec {:a 1}) =&gt; true
(s/valid? ::my-spec {:b 1}) =&gt; true
(s/valid? ::my-spec {:a 1 :b 1}) =&gt; true
(s/valid? ::my-spec {:A1 :B 1}) =&gt; true
```

U050MP39D : <@U3QUAHZJ6> (some-fn :a :b) not good enough?
U3QUAHZJ6 : some-fn is actually a function? i presumed i was supposed to fill in the gaps &gt;.&lt; feel so dumb right now
U050MP39D : oh I'm sorry yeah I could see how you'd read it that way. but no, some-fn is a higher order function that takes a list of predicate functions and returns a single predicate function that is the logical or of all of them
U3QUAHZJ6 : makes sense now!
U050MP39D : you could also do `#(some % [:a :b])`
U050MP39D : ^ relies on maps acting as functions
U050SC7SV : You can use :req-un [(or ::foo ::bar)]
U050MP39D : I totally forgot about that syntax
U050SC7SV : It's one thing I love in spec