Namespaces and Testing

The most exciting Clojure lecture to date

So you’ve heard of namespaces...right?

- What the heck is a namespace?
    - Object of type clojure.lang.Namespace;
    - Contain maps between symbols and shelf addresses (vars)
        - Think of shelf addresses as spots where Clojure knows where to find something, but not WHAT is on the shelf there

- Why would I ever have to use a namespace?
    - Super beneficial when organizing a project or library
    - Allows nested scoping of variables and functions to allows for a sort of insurance policy against any side effects involved with using them inside a project (ie, you can have variables defined in one namespace, and the same variables defined in another)
        - We’ll see this in a minute

Some simple namespace commands

What namespace am I in?

`(ns-name *ns*)`

Create a namespace pls

`(create-ns ‘symbolic-name-for-ns)`

`(in-ns ‘symbolic-name-for-ns)` - creates and changes to namespace

`(ns ‘ns-name-here)` - same as in-ns, but also loads clojure.core

Delete a namespace pls

`(remove-ns ‘symbolic-name-for-ns)` - be careful with this though!

What namespaces are defined in my environment?

- `(all-ns)`

  Which returns a sequence of all the namespaces defined in your environment

- A little spicier output:

  - `(doseq [namespace (all-ns)] (println (ns-name namespace)))`

What symbols are defined in a namespace?

- `(ns-map ‘namespace-name)` OR `(ns-map *ns*)` for current namespace

  This will return a huge list of symbols that’re defined in the namespace…

- `‘<keyword> (ns-map ‘namespace-name)` will be nicer ;)

Testing

You guys don’t write tests?

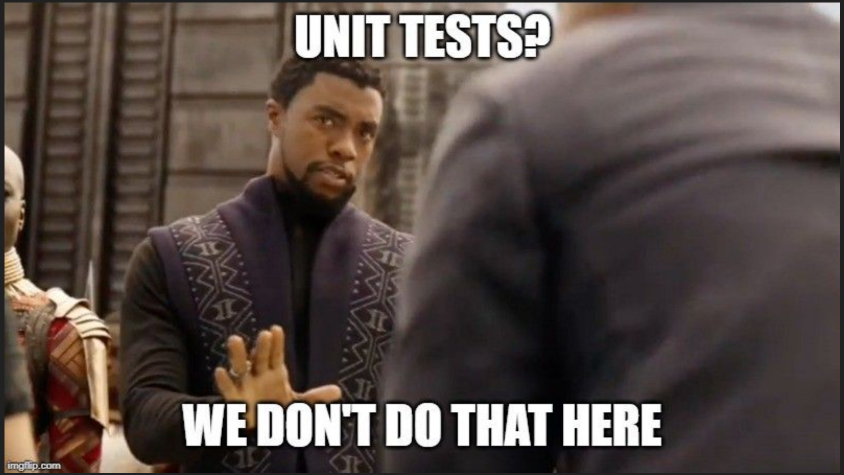
Unit Testing vs Functional Testing

- Written from a developer’s perspective
- Ensure that a method (unit) performs a specific task, or set of tasks
- Dependencies are mocked out

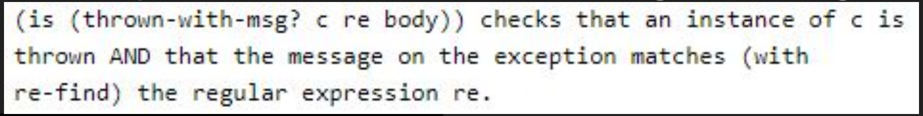
- Written from a user’s perspective
- Tests *how* the application is working with its dependencies, like databases or web services
- End-to-end

Why Write Tests?

- We want to verify that our code is performing the tasks it should, as it should
- We want to verify that new functionality doesn't impede the ability of old functionality to perform
- It forces developers to think critically about how their code works
- It helps to catch bugs!



Unit Testing in Clojure - clojure.test

- Some useful methods to be aware of:
  - Assertions are done with the is macro
  - Check if exceptions are thrown using thrown?
  - Check if exceptions are thrown with a message matching a regex using thrown-with-msg?
    - 
      (is (thrown-with-msg? c re body)) checks that an instance of c is thrown AND that the message on the exception matches (with re-find) the regular expression re.
- (run-tests 'namespace-name) will run all tests within a namespace
- (run-all-tests) will run ALL tests in ALL namespaces, including println's
- (with-redefs …) allows us to redefine a function within a test (I'll show an example of this in a minute)

Unit Testing in Clojure - clojure.test with Leiningen

- Clone my repo pls
- Let’s use Leiningen to run some tests!
  - This functionality is built-in to Leiningen
  - We can run all tests defined with `lein test`
  - We can specify tests to be run using namespaces (ooooo)
    `lein test :only <namespace>`

Exercise time! Let’s try to write some tests verifying that messages were thrown in `test\lecture\exceptions_test.clj`

Spies, Stubs, and Mocks in Clojure

- What the heck are these?
- Spy
  - Wraps a real object, allowing you to verify parameters, calls, and throws to a function
- Stub
  - Focused on verifying state - it is both a stub and a mock. These allow us to mimic return vals.
- Mock
  - Focused on verifying behavior. Allow you to completely mock a function’s behaviour at your discretion.

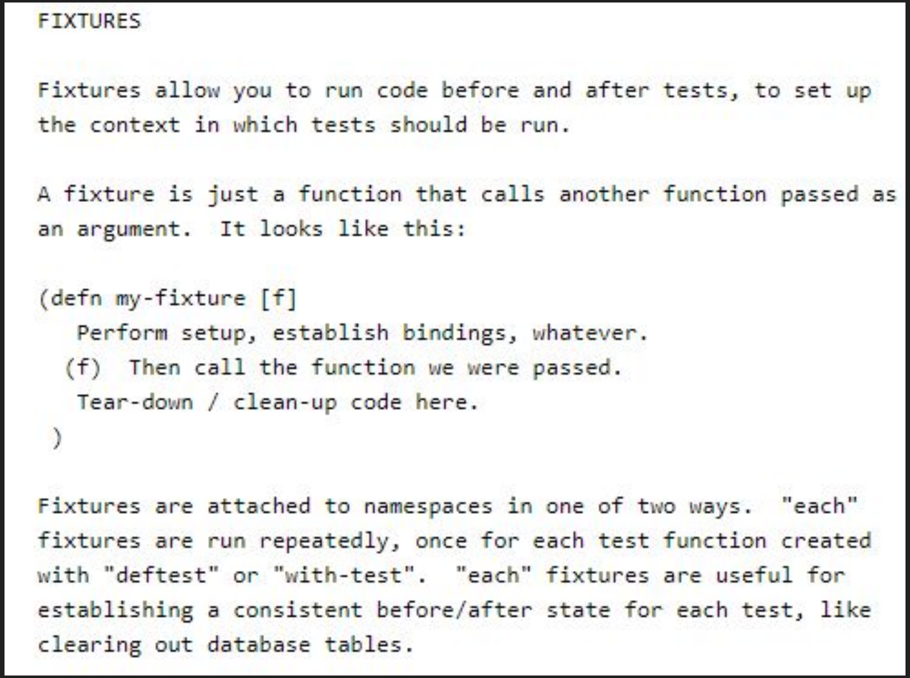
- We’ll use this repo to test out how these work
  - Fairly similar to sinon if anyone’s used that for jest mocking in JS

Exercise v2

- Let’s go through stub and spy examples first…
- And then you can write your own tests using them!

Fixtures

After I'd written all of the tests I discovered fixtures … they seem cool



```
FIXTURES

Fixtures allow you to run code before and after tests, to set up
the context in which tests should be run.

A fixture is just a function that calls another function passed as
an argument.  It looks like this:

(defn my-fixture [f]
   Perform setup, establish bindings, whatever.
  (f)  Then call the function we were passed.
   Tear-down / clean-up code here.
 )

Fixtures are attached to namespaces in one of two ways.  "each"
fixtures are run repeatedly, once for each test function created
with "deftest" or "with-test".  "each" fixtures are useful for
establishing a consistent before/after state for each test, like
clearing out database tables.
```

References

- https://stackoverflow.com/questions/2741832/unit-tests-vs-functional-tests
- https://clojurebridge.org/community-docs/docs/clojure/namespace/
- https://www.braveclojure.com/organization/
- https://github.com/alexanderjamesking/spy
- https://clojure.github.io/clojure/clojure.test-api.html