Apollo design guidelines

Revision number: 1

Date:

Name: P. van der Velde

* Collection.Remove, should this return a Boolean indicating if remove was successful?
* Allow predicates for selecting /searching in a collection
* Make API powerful 🡪 Work in layers 🡪 Low level upwards. But make sure that we have a high level abstract API that is easy to use
* Throw exception if a member cannot live up to the design contract (i.e. can’t do it’s job)
* Collections always have to be IEnumerable

As a good citizen, I...

* Keep a consistent state at all times - init() or populate() is a code smell.
* Have no static fields or methods
* Never expect or return null.
* FailFast - even when constructing.
* Am Easy to test- all dependent object I use can be passed to me, often in my constructor (typically as [Mock Objects](http://docs.codehaus.org/display/PICO/Mock+Objects)).
* Accept dependent object that can easily be substituted with Mock Objects (I don't use [Concrete Class Dependency](http://docs.codehaus.org/display/PICO/Concrete+Class+Dependency)).
* Chain multiple constructors to a common place (using this(...)).
* Always define hashCode() alongside equals()
* Prefer immutable value objects that I can easily throw away.
* Have a special value for 'nothing' - e.g. Collections.EMPTY\_SET.
* Raise checked exceptions when the caller asked for something unreasonable - e.g. open a non-existant file.
* Raise unchecked exceptions when I can't do something reasonable that the caller asked of me - e.g. disk error when reading from an opened file.
* Only catch exceptions that can be handled fully.
* Only log information that someone needs to see.
* Classes that are designed for [Constructor Injection](http://docs.codehaus.org/display/PICO/Constructor+Injection) are better citizens than those that are not.