Dry 3

Chosen library: 208582585-206486904

Pros:

1. minimal but complete – The point was to wrap secureStorage and when using this library (that supplies read & write) we can implement everything we could have when using the under-laying secureStorage. (“what's common should be easy, what's uncommon should be possible”)
2. Easy to explore since it’s very minimal (good Discoverability)

Cons:

1. Too minimal – does not support any thing else besides bare minimum (only read & write) e.g. “exists”.
   - Would change – add more convenience methods.
2. “IBasicStorageManager” (the layer used for convenient read/write above secureStorage) methods request the “secureStorage” instance, essentially making it stateless. in that case I would expect it to be a **companion object** clarifying that any IBasicStorageManager implementation is stateless.
   Also, a downside of this API is that the user of this class needs to supply it with it’s inner secureStorage making them coupled to each other, instead of incapsulating the use of secureStorage inside the BasicStorageManager (also exposing the secureStorage instance to outside use that would surprise the BasicStorageManager wrapper)
   - Would change – mark IBasicStorageManager methods as static **or** (preferred) encapsulate secureStorage in it and make it stateful.

SifriTaub:

Pros:

1. Follows YAGNI and is very mission oriented to the specific needs of the client. there are no vague generic methods, iterators etc. that we support for unknown future needs of the client.
2. Minimized mutability – books, users and loans cannot be removed. This is fine since it doesn't bother the system from moving forward and doing what it needs to do. We are protecting the client from a situation where something was removed and later expected to exists by some other module that was not notified of the deletion.

Cons:

1. Not powerful enough to my opinion.
   i.e. the client of this API cannot do everything that the library could have done for him.
   e.g. - Get the available amount for some book: if he want to know if it's worth adding to his loan request that might delay because of that book (if it's 0).
   - Would change – add methods that complete the client's ability to implement more possible uses of the SifriTaub that the library could have implemented for him (not do every thing! Just complete his initial set of capablities) like replace getBookDescription with "bookInformation" which returns book desc **and** it's current available amount.
2. Inconsistent naming - [user/loanRequest]information are both info querying by id as apposed to getBookDescription which also queries for info using id and I would expect it to be called "bookDescription" ("Consistency is more important than readability")
   - Would change – either use the verbose "get" prefix or not – consistently
3. No interface – causing the the SifriTaub client to be coupled (and exposed) to this implementation rather then an API only
   - Would change – add a SifriTaub interface file and create a separated implementation
4. No use of compile time type-checking. E.g. addBookToCatalog(), register() and any other method that receives Int as one of it's parameters and throws an error if it has a negative value is doing a run-time check that could have been done by the compiler instead.
   - Would change – accept an Unsigned Int and prevent client misuse at compile time.