Software Design H.W 0 – Dry Part

Sahar Cohen, 206824088
Yuval Nahon, 206866832

Q.1:

When talking about class design, the lecturer gives an example of a misuse of inheritance in java's own library API. The example is the generic **stack** class, which is a subclass of the **vector**. This serves as an example of misuse of inheritance because clearly, we cannot say "stack **is a** vector": these two data structures are fundamentally different. For example, a vector supports random indexing whereas a stack by its vanilla definition should not support it – this breaks the "is a" relationship.

Q.2:

Checked exceptions are exceptions that are checked at compile time, so if some method **could** throw an exception of this kind: it **must** specify it explicitly (most throwables in Java, for example, are checked).

The lecture shows us a disadvantage of these exceptions that forces implementation to impact the API itself. The given example is an API about phone numbers that has some method(s) that throw sequel exceptions (which are checked in Java). We might want to re-implement this API using some non-sequel datastore, but herein lies a problem: the clients who use the API at this point are already trying to catch sequel exceptions. If we change the implementation now: our exceptions will be of some other checked type. It is most definitely a bad design to emulate sequel exceptions from the new exceptions because of ambiguity and plain **wrong** communication of the exception, so there is no "easy fix" to this problem.

The design solution here is make sure that the checked exceptions we declare in the API are on the same layer of abstraction as the rest of the API. In other words, all implementations of this API **should** throw these exceptions, and this should derive directly from the purpose of the API. Implementation-dependent checked exceptions are not welcome in APIs.