Nicole P. Satiembre **March 25, 2024**

Cebu Technological University-Main Campus

Clean Code

Chapter 6

6. Objects and Data Structures.

There is a good reason why we need to understand the role of objects and Data Structures especially when keeping our private variables private.

Data Abstraction. This passage shows an example of two listings that represent the data of a point on the Cartesian plane, where one exposes its implementation and the other completely hides it. This demonstrates the importance of access policy. Hiding implementation is not just putting a layer of functions between the variables. Hiding implementation is about abstractions. This means that variables in a class will not simply be manipulated through getters and setters, rather, it exposes abstract interfaces that allow its users to manipulate the essence of the data, without having to know its implementation.

Data/Object Anti-Symmetry. Objects hide their data behind abstractions and expose functions that operate on that data. Data structure exposes their data and has no meaningful functions. In Procedural code, it will be hard to add new data structures because all of the functions must change. On the other hand, object-oriented code makes it hard to add new functions because all the classes must change. There will be systems where OOP is much more applicable however it is not the only approach that is in there, in other scenarios, procedural codes are much more helpful especially if we want to add new functions as opposed to data types. Experienced programmers understand the mistake of treating everything as an object. There will be times when we just need to use simple data structures with procedures operating on them.

The Law of Demeter. A module should not know about the innards of the objects it manipulates. An object should not expose its internal structure through accessors because it exposes its internal structure. The method should not invoke methods on objects that are returned by any of the allowed functions.

Train Wrecks. Chains of calls are generally considered to be sloppy style and should be avoided. An example would be a containing module that knows that the ctxt object contains options, which contain a scratch directory, which has an absolute path. If they are objects, then this violates the Law of Demeter, on the other hand, if they are just data structures with no behavior, then they naturally expose their internal structure, and so Demeter does not apply.

Hybrids. There are times when hybrid structures are implemented where there are half object and half data structures. This is very confusing and we should avoid it because they make it hard to add new functions and also make it hard to add new data structures.

Hiding Structure. Objects are supposed to hide their internal structure, issues could arise when objects expose their internal structure. This shows the importance of encapsulation and an object-oriented design. Instead, it suggests a more object-oriented approach where objects should encapsulate behavior rather than exposing their internal state. By delegating tasks to objects and letting them perform actions rather than querying their internal details, code becomes more modular, easier to maintain, and less prone to unexpected changes.

Data Transfer Objects. In object-oriented programming, data structures can take the form of classes with public variables (DTO) or private variables manipulated by getters and setters (beans). DTOs are commonly used for communication between different layers of an application, while beans offer a degree of encapsulation but may not always provide significant benefits beyond appeasing purists. Both forms serve important roles in handling and representing data within software systems.

Active Record. Active Records are special forms of DTOs. They are data structures with public (or bean-accessed) variables, but they typically have navigational methods like save and find. Typically these Active Records are direct translations from database tables or other data sources. We should treat the Active Record as a data structure and create separate objects that contain the business rules and that hide their internal data.

Conclusion. We must understand the proper use of Data structures and OOPs. Objects expose behavior and hide data which makes it easy to add new kinds of objects without changing existing behaviors, however, this also makes it hard to add new behaviors to existing objects. On the other hand, Data structures expose data and have no significant behavior. This makes it easy to add new behaviors to existing data structures but makes it hard to add new data structures to existing functions.