Nicole P. Satiembre **March 27, 2024**

Cebu Technological University-Main Campus

Clean Code

Chapter10-Classes

Class Organization. A good rule to follow when organizing classes is to follow the standard Java convention. The class should begin with a list of variables. Public static constants, if any, should come first. Then private static variables, followed by private instance variables. There is seldom a good reason to have a public variable. Public functions should follow the list of variables. We like to put the private utilities called by a public function right after the public function itself.

Encapsulation. We like to keep our variables and utility functions private. If a test in the same package needs to call a function or access a variable, we'll make it protected or package scope.

Classes Should be Small! The title explains itself, classes should be small, and we count responsibilities, not physical lines.

The Single Responsibility Principle. The Single Responsibility Principle (SRP)2 states that a class or module should have one, and only one, reason to change. This principle gives us both a definition of responsibility and guidelines for class size. Classes should have one responsibility—one reason to change. To restate the former points for emphasis: We want our systems to be composed of many small classes, not a few large ones. Each small class encapsulates a single responsibility, has a single reason to change, and collaborates with a few others to achieve the desired system behaviors.

Cohesion. Classes should have a small number of instance variables. Each of the methods of a class should manipulate one or more of those variables. In general, the more variables a method manipulates the more cohesive that method is to its class. A class in which each variable is used by each method is maximally cohesive.

Maintaining Cohesion Results in Many Small Classes. Breaking a large function into many smaller functions often allows us to split several smaller classes out as well.

Organizing for Change. Organizing our classes helps our code to become simple and easy to understand. The risk that one function could break another becomes vanishingly small. When it's time to add the update statements, none of the existing classes need changed. Classes should be open for extension but closed for modification.

Isolating from Change. This passage shows the importance of abstraction. A client class depending upon concrete details is at risk when those details change. We can then use interfaces and abstract classes to help isolate the impact of those details. Dependencies upon concrete details create challenges for testing our system.