**SOFTWARE ENGINEERING BASICS**

**SOLID PRINCIPLES**

To reinforce the definitions, strongly advise you to take a look to examples written in C# or Python.

1. **Single Responsibility Principle (SIP)**

SIP states that every class should have only one job or responsibility. That does not mean that a class has to have only one method. A class can have more than one method but the important thing is to have ONLY ONE responsibility.

[For more details.](https://realpython.com/solid-principles-python/)

1. **Open-Closed Principle**

Open closed principle means that classes, modules, functions should be open for extension and closed for modification. Following OCP will make your codebase loosely coupled.

[baeldung open-closed principle example](https://www.baeldung.com/java-open-closed-principle#:~:text=2.,be%20extended%2C%20but%20not%20modified).

1. **Liskov Substitution Principle**

LSP states that in an object-oriented program, if we substitute a superclass object reference with an object of any of its subclasses the program should not break.

Google it! :D

1. **Interface Segregation Principle (ISP)**

Segregation means keeping your stuff separated. In this context, ISP indicates that you should separate your interfaces. Therefore, a class or a subclass doesn't have to implement the method or methods that it does not need.

Please see the coding examples!

1. **Dependency Inversion Principle**

Dependency inversion is the strategy of depending upon interfaces or abstract function and classes rather than upon concrete function and classes.

Simply put, when components of our system have dependencies, we don’t want to directly inject a component’s dependency into another. Instead, we should use a level of abstraction between them.

DIP states that high level modules should not depend on low-level modules; they should depend on abstractions. Secondly abstractions should not depend upon details; details should depend on abstractions.

The idea is that we isolate our class behind a boundary formed by abstractions it depends on. If all details behind those abstractions change, then our class is still safe. This helps keep coupling low and makes our design easier to change. DIP also offers to test things in isolation, details like databases are plugins to our system.

[java tech dependency inversion example](https://javatechonline.com/solid-principles-the-dependency-inversion-principle/)