**UNIVERSITY OF THE ARMED FORCES – ESPE**
**HW18 – Software Engineering Principles**

**Name:** Joel Alexander Zeas Clavijo
**Date:** 01th February 2022
**NRC:** 7490
**Career:** Telecommunications

## Introduction to the SOLID principles

To achieve quality software and guarantee its correct operation, the following points can be taken into account that help us to carry out this:

-

The class must be encapsulated and each class must be responsible for only part of the system operation. This principle is related to the principle of cohesion, which focuses on a single function and is related to encapsulation.

-

A class system is open if it is available for extension and closed if it is available for use by another class, for modification. This principle helps reduce complexity and increase extensibility. Thanks to this, you can get:
-Each abstraction should focus on a single purpose.
-Concrete classes inherit from these abstractions.

-

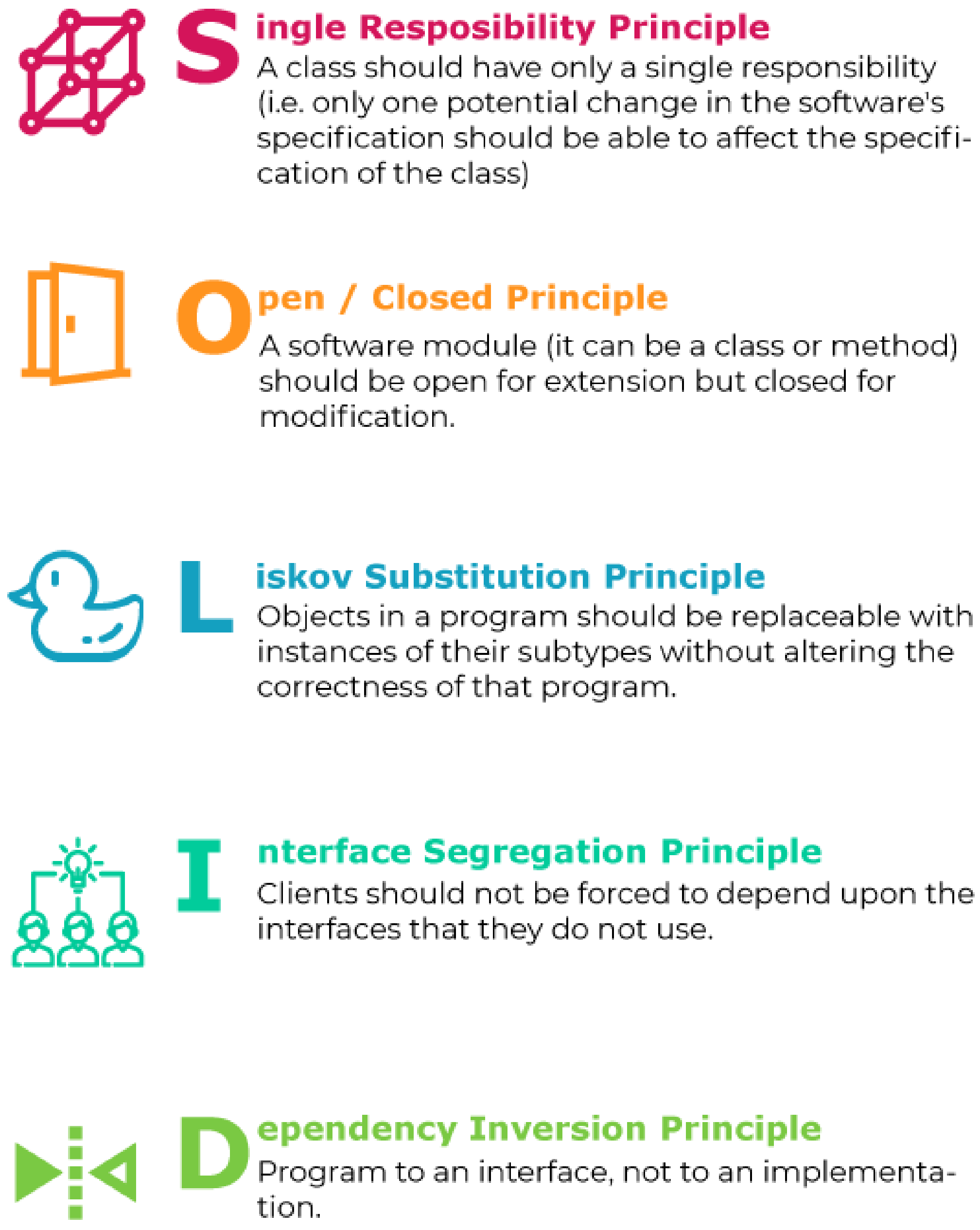
It is used in OOP and says that each class that inherits from another can be used as its parent without needing to know the differences between them, resulting in:
-A Product is a base class using the "save an object to a file" method.
-Implementation of specializations: Widget of some product fulfills a single purpose or a single function within the system.

-

Avoid the implementation of methods to classes that do not need it, to avoid problems that can lead to unexpected errors and unwanted dependencies and thus code can be reused and improvements implemented.

- D | Dependency inversion principle

This principle helps us understand that high-level modules should not depend on low-level modules, since both should depend on abstractions. The abstractions must not depend on the details, but the details must depend on the abstractions.



**S**ingle Resposibility Principle
A class should have only a single responsibility (i.e. only one potential change in the software's specification should be able to affect the specification of the class)

**O**pen / Closed Principle
A software module (it can be a class or method) should be open for extension but closed for modification.

**L**iskov Substitution Principle
Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.

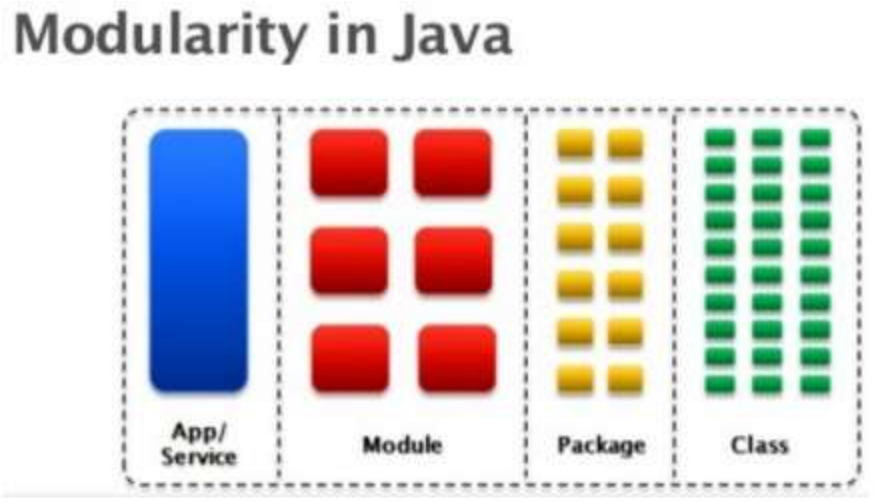
**I**nterface Segregation Principle
Clients should not be forced to depend upon the interfaces that they do not use.

**D**ependency Inversion Principle
Program to an interface, not to an implementation.

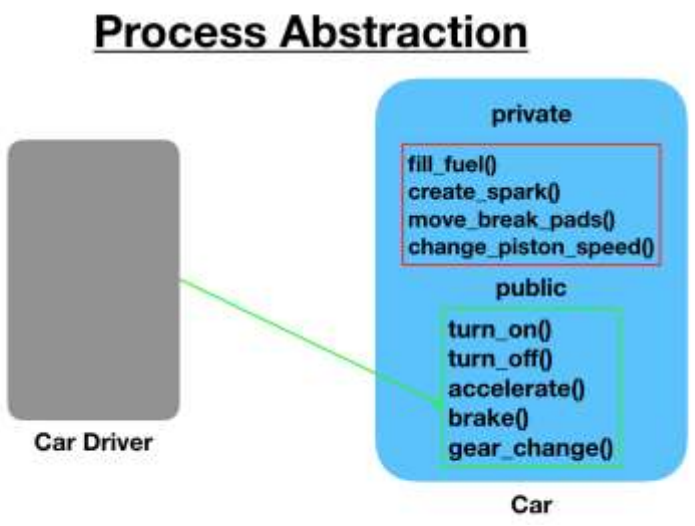
# Modularity, abstraction and encapsulation

These mentioned methods help to improve the functionality, development and understanding of a code, among them it can be highlighted that:

**Modularity.-**
Is the ability to ignore the details of the parts in order to focus attention on a higher level of a problem. It is based on the principle of "divide and conquer", in this procedure something similar is done, since a system is broken down into subprograms known as modules or smaller parts.



**Abstraction.-**
Happens when something is wrapped in a protective layer or shielded from anything that might harm it. They are the specific characteristics of an object, those that distinguish it from other types of objects and that manage to define conceptual limits regarding who is making said abstraction of the object.



**Encapsulation.-**

Is the act of packing or protecting data or attributes with methods. It is called encapsulation to the hiding of the state, that is, of the member data, of an object so that only it can be changed by the operations defined for that object, or the programmer alone can do it.