INTRODUCTION TO THE SOLID PRINCIPLES

OVERVIEW OF THE SOLID PRINCIPLES

• SOLID is a mnemonic acronym for five principles

• Single Responsibility Principle

• Open/Closed Principle

• Liskov Substitution Principle

• Interface Segregation Principle

• Dependency Inversion Principle

DESIGN PROBLEM

It consists of a detailed study of the problem. The input data, output data and the description of the problem must be identified.

SINGLE RESPONSIBILITY PRINCIPLE

The Single Responsibility Principle is the first of the five that make up SOLID.

The Single Responsibility principle tells us that an object must do only one thing. It is very common, if we do not pay attention to this, that we end up having classes that have several logical responsibilities at the same time.

OPEN/CLOSED PRINCIPLE

This principle tells us that a software entity should be open to extension but closed to modification.

• A class is open if it is still available for extension

• A class is closed if it is available for use by other class, and therefore should not be modified

INTERFACES, ABSTRACT CLASSES, PURE VIRTUAL CLASSES

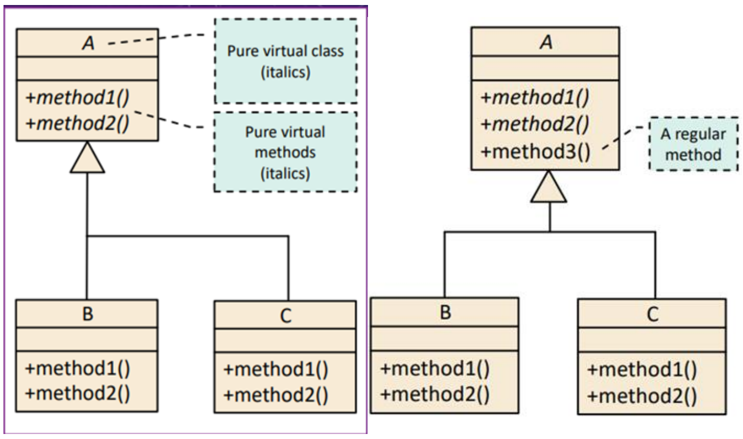
• Review: Inheritance allows a specialization (a derived class) to re-use the generalization's (a base class's)

• An interface is like a base class, but only allows for method declarations

• In UML, the names of abstract and pure virtual classes and methods are written in italics

• An abstract and a pure virtual class (C++) may include data members and some method implementations

• The modern Open/Closed Principle encourages developers to



OPEN/CLOSED PRINCIPLE

•Inheritance

•Aggregation

• Parameterization

• Following the Open/Closed Principle can help developers

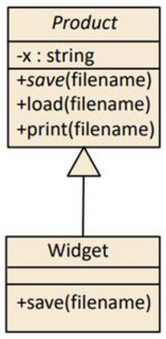
LISKOV SUBSTITUTION PRINCIPLE

El principio de sustitución de Liskov nos dice que si en alguna parte de nuestro código estamos usando una clase, y esta clase es extendida, tenemos que poder utilizar cualquiera de las clases hijas y que el programa sigue siendo válido.

Esto nos obliga a asegurarnos de que cuando extendemos una clase no estamos alterando el comportamiento del padre.

FOLLOWING THE LISKOV SUBSTITUTION PRINCIPLE

This principle comes to disprove the preconceived idea that classes are a direct way of modeling reality, and that care must be taken with that modelling.



LISKOV SUBSTITUTION PRINCIPLE

• Following the Liskov Substitution Principle can help developers

• Increase Reuse

• Increase Extensibility

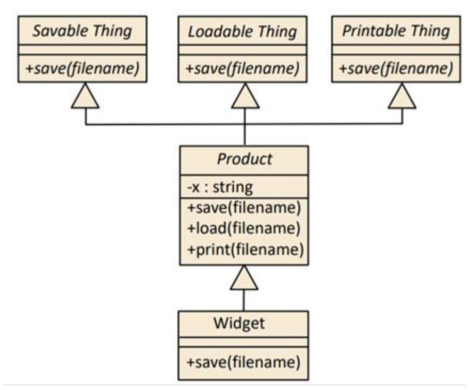
• Increase Maintainability

INTERFACE SEGREGATION PRINCIPLE

The principle of interface segregation means that no class should depend on methods that it does not use.

Therefore, when creating interfaces that define behaviors, it is important to make sure that all classes that implement those interfaces will need and be able to add behaviors to all methods. Otherwise, it is better to have several smaller interfaces.

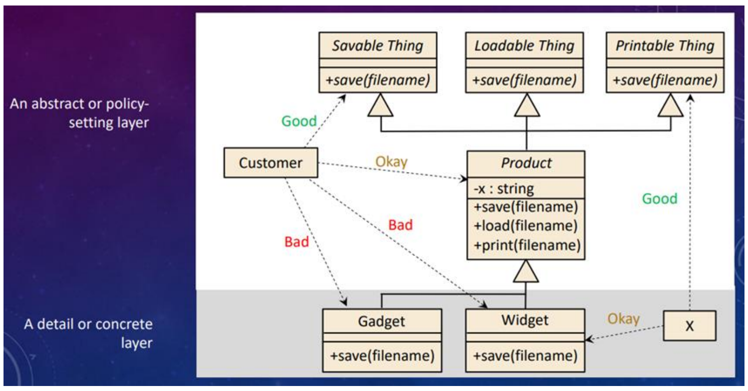
INTERFACE SEGREGATION PRINCIPLE



DEPENDENCY INVERSION PRINCIPLE

Interfaces help us decouple modules from each other. This is so because if we have an interface that explains the behavior that the module expects to communicate with other modules, we can always create a class that implements it so that it meets the conditions.

DEPENDENCY INVERSION PRINCIPLE


An abstract or policy-setting layer
A detail or concrete layer
Savable Thing
+save(filename)
Loadable Thing
+save(filename)
Printable Thing
+save(filename)
Good
Customer
Okay
Product
-x : string
+save(filename)
+load(filename)
+print(filename)
Good
Bad
Bad
Gadget
+save(filename)
Widget
+save(filename)
Okay
X