REPUBLIQUE TUNISIENNE
*** * ***
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
*** * ***
UNIVERSITE DE CARTHAGE
*** * ***
INSTITUE NATIONALE DES SCIENCES APPLIQUEES ET DE TECHNOLOGIE

Compte rendu du TP n° 1 en Architectures Logicielles

Filière : Génie Logiciel 4

Sujet :

Principes SOLID

Réalisé par : Yosra DRIDI

Professeur : Mme Hela Chikhaoui

Année Universitaire 2021/2022

Introduction

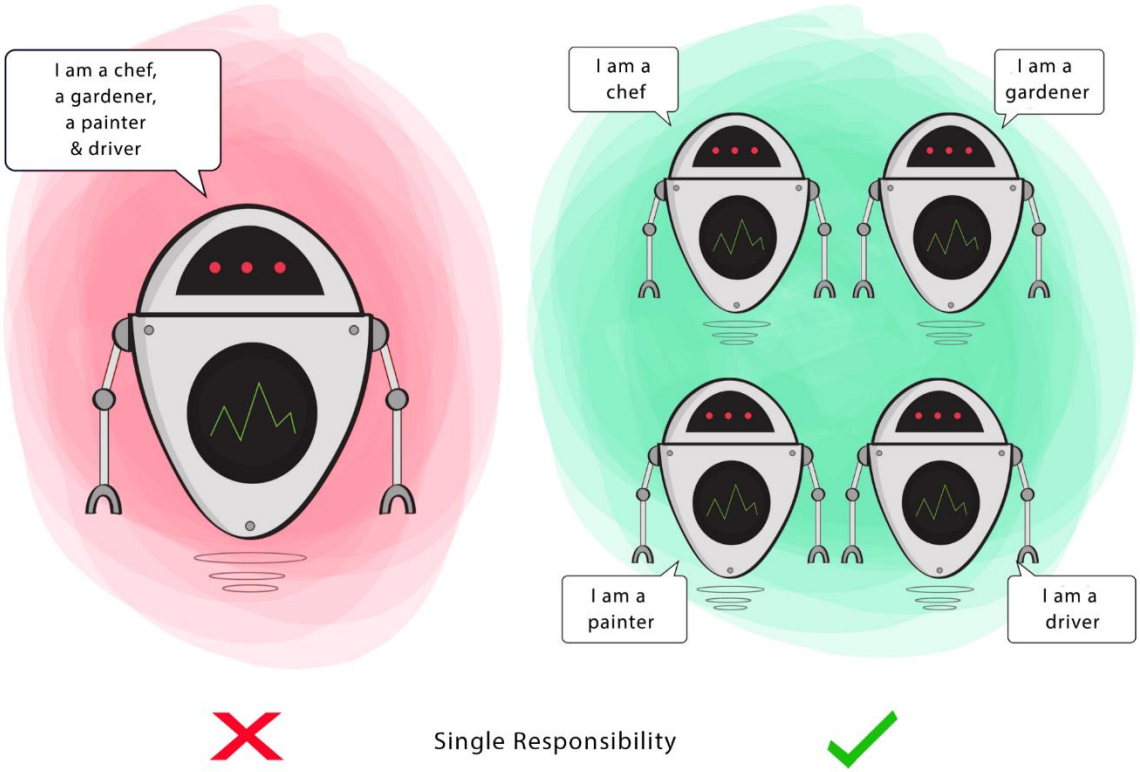
SOLID aims at creating understandable, readable, and testable code that many developers can collaboratively work on.

SOLID stands for:

- **S** – **S**ingle **R**esponsiblity **P**rinciple (**SRP**)
- **O** – **O**pen **C**losed **P**rinciple (**OCP**)
- **L** – **L**iskov **S**ubstitution **P**rinciple (**LSP**)
- **I** – **I**nterface **S**egregation **P**rinciple (**ISP**)
- **D** – **D**ependency **I**nversion **P**rinciple (**DIP**)

I. Single Responsability Principle (SRP)

This principle aims to separate behaviours so that if bugs arise as a result of your change, it won't affect other unrelated behaviours.



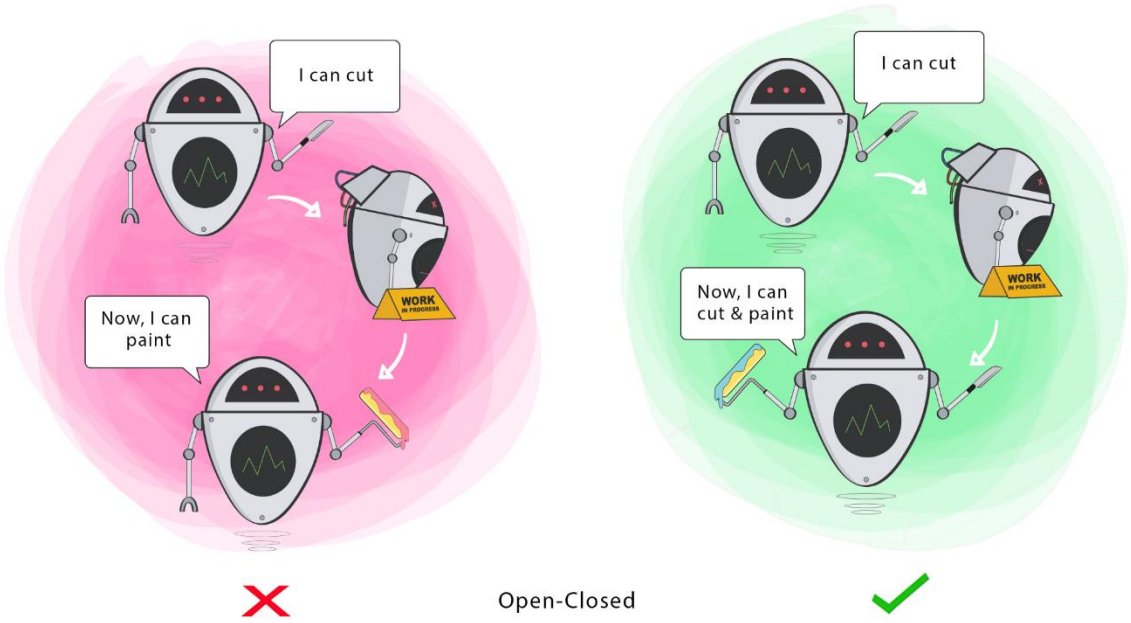
Further Improvements :

- For looser coupling, we added interfaces (formatter, analyzer, ..) => DIP

- We added `getCarName()` to the car formatter interface (since users that wish formatting a single car name will be forced to put that car in a list) => `getCarsNames()` calls `getCarName()`.
- Cars manager is not adding any fonctionalities => may be we should add some methods that a manager should take care of unless this manager is only a façade.

II. Open Closed Principle (OCP)

This principle aims to extend a Class's behaviour without changing the existing behaviour of that Class. This is to avoid causing bugs wherever the Class is being used.

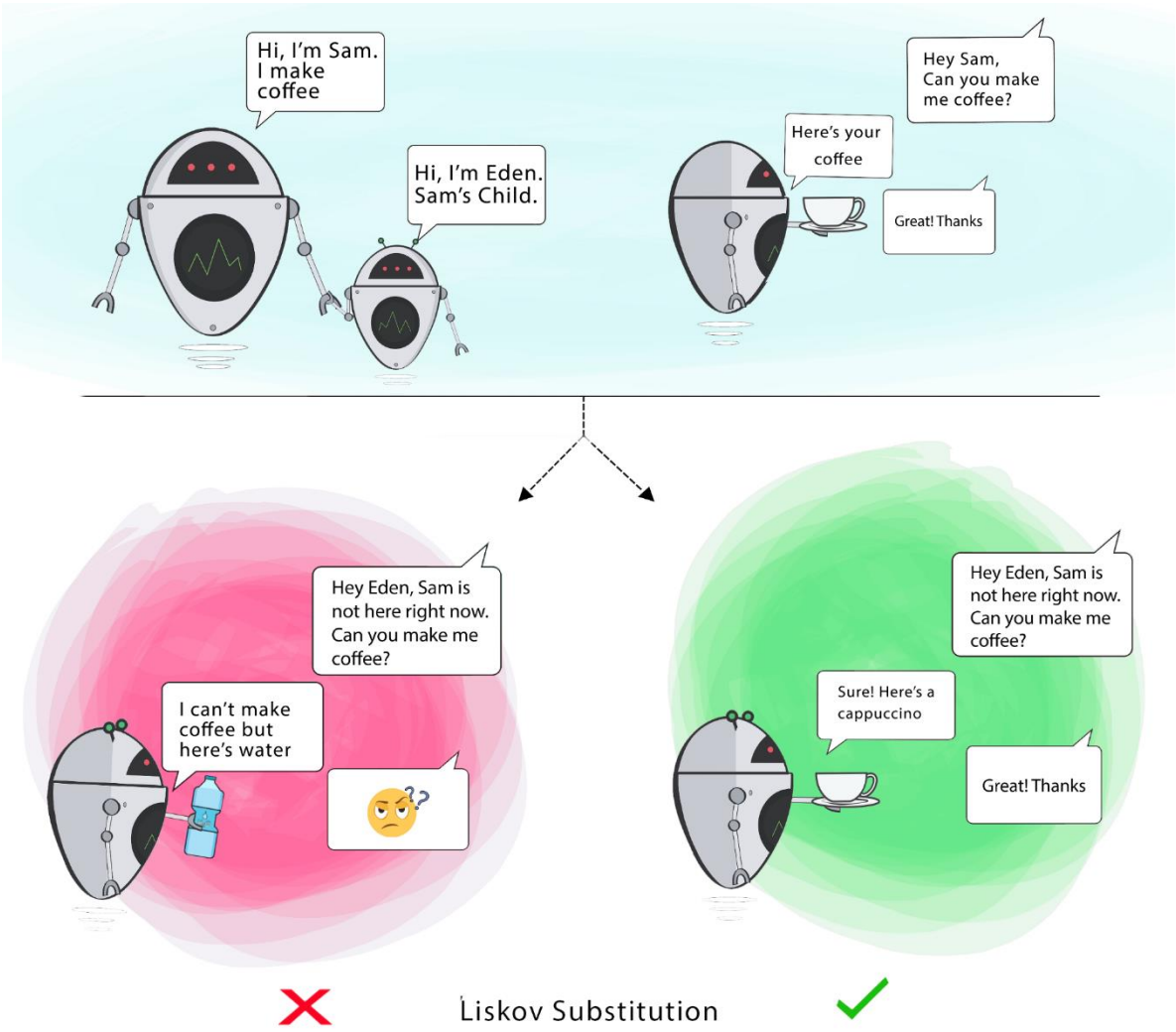


Further Improvements :

- We used an interface of a resource manager that takes care of allocating and freeing resources slots. Two manager implements this interface : space resource manager and time resource manager.

III. Liskov Substitution Principle (LSP)

This principle aims to enforce consistency so that the parent Class or its child Class can be used in the same way without any errors.

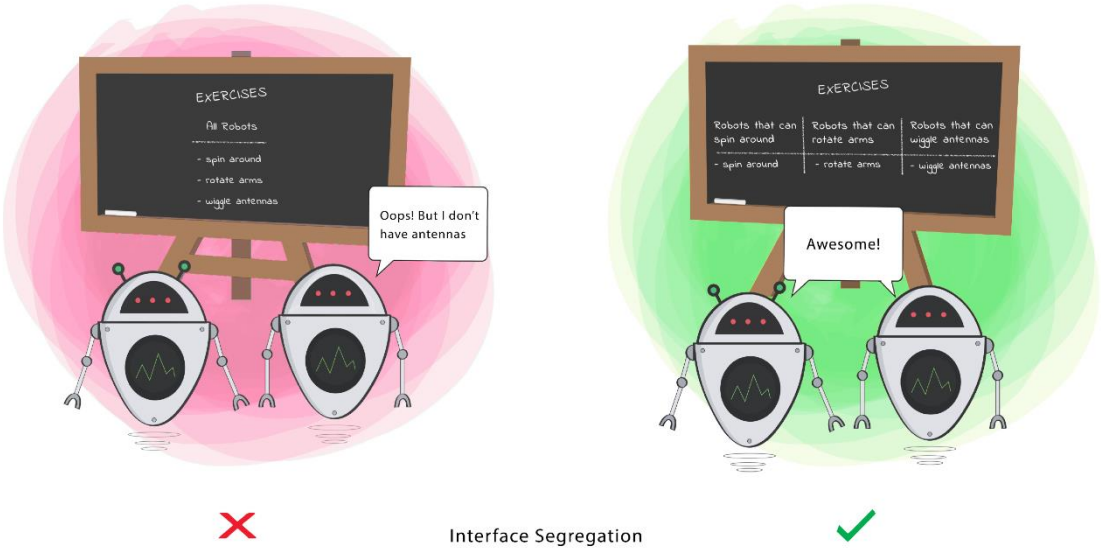


Further Improvements :

- An electronic duck cannot quack or swim when it is turned Off. But, since a duck is supposed to always quack and swim, then to solve the problem we turn On the electronic duck whenever its quack and swim methods are called.

IV. Interface Substitution Principle (ISP)

This principle aims at splitting a set of actions into smaller sets so that a Class executes ONLY the set of actions it requires.



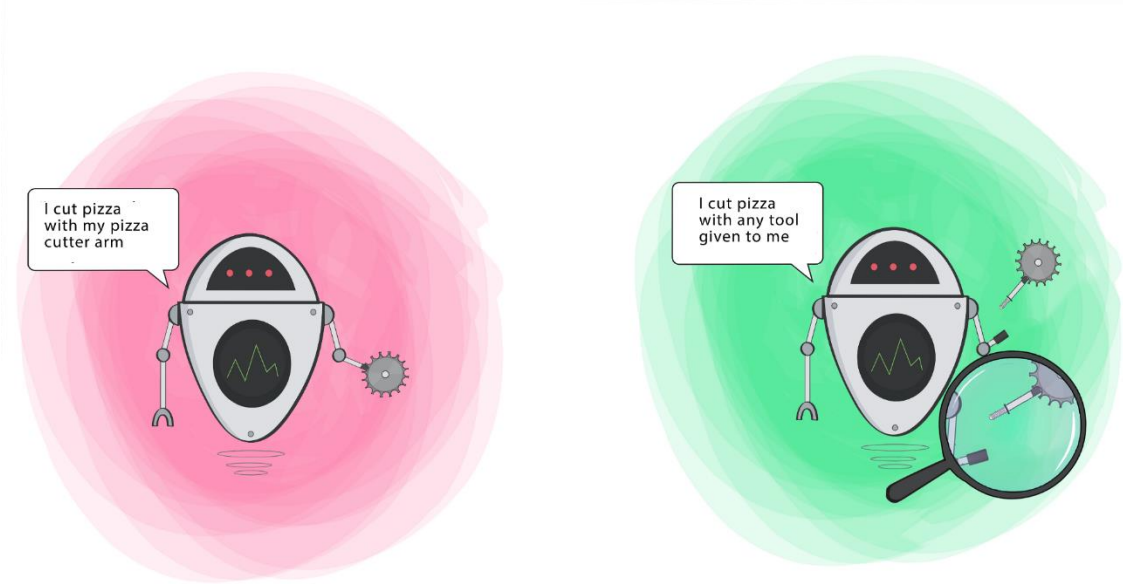
Interface Segregation

- **Further Improvements :**

- A door always has the attributes : opened and locked. Furthermore, we can always open, close, lock and unlock a door, that is why we repeat the same code for these methods in every kind of door => we use and abstract class of the Door instead of an interface where we put the implementation of the 4 methods and the 2 attributes.

V. Dependency Inversion Principle (DIP)

This principle aims at reducing the dependency of a high-level Class on the low-level Class by introducing an interface.



Dependency Inversion

Conclusion

Tout au long de ce TP, on a pu appliquer les principles SOLID pour avoir un code plus propre.