SOLID Principles

UPskill

Digital Skills & Jobs

SOLID

- **S**ingle Responsibility Principle (SRP)
- **O**pen/Closed Principle (OCP)
- **L**iskov Substitution Principle (LSP)
- **I**nterface Segregation Principle (ISP)
- **D**ependency Inversion Principle (DIP)

Single Responsibility Principle (SRP) SOLID

- SRP states that:
  - A class should have a single responsibility
  - There should never be more than one reason for a class to change
- It is equivalent to **High Cohesion** pattern
- If a class has more than one responsibility, it can become hard to understand, maintain, and modify

Open/Closed Principle (OCP) (1/2)

sOLID

- OCP states that:
  - Classes should be Open for Extension
  - But Closed for Modification
- OCP promotes software extensibility and maintainability
- It is equivalent to **Protected Variations** pattern
  - *Example*: Algorithms / External Services
  - *Solution*: Create a stable interface (recall previous presentations)

Open/Closed Principle (OCP) (2/2)

SOLID

- OCP states that:
  - Classes should be Open for Extension
  - But Closed for Modification

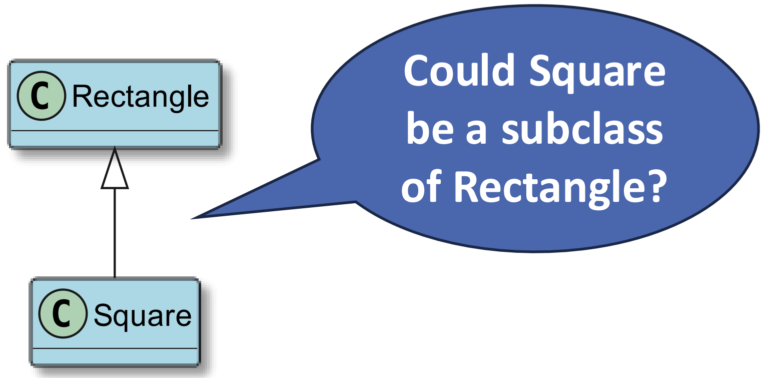
If the code works,
do not change it.
Extend it!

- OCP promotes software extensibility and maintainability

- It is equivalent to **Protected Variations** pattern
  - *Example*: Algorithms / External Services
  - *Solution*: Create a stable interface (recall previous presentations)

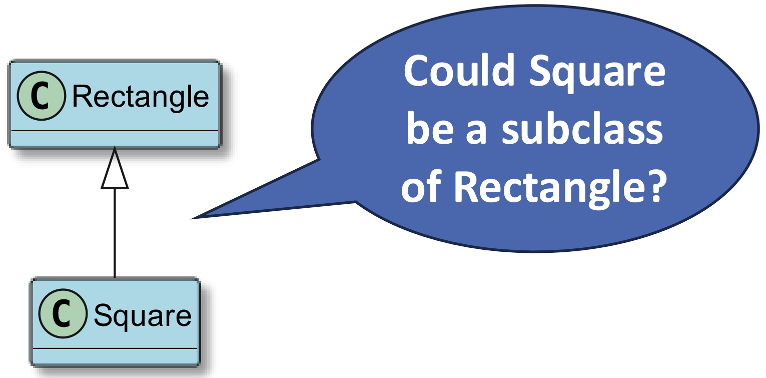
Liskov Substitution Principle (LSP) (1/2) SOLID

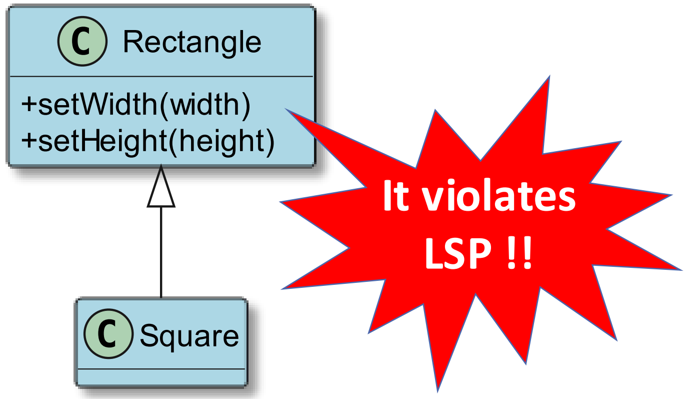
- LSP states that any instance of a derived/sub class should be substitutable for an instance of its base/super class without affecting the correctness of the program



Liskov Substitution Principle (LSP) (2/2) SOLID

- LSP states that any instance of a derived/sub class should be substitutable for an instance of its base/super class without affecting the correctness of the program



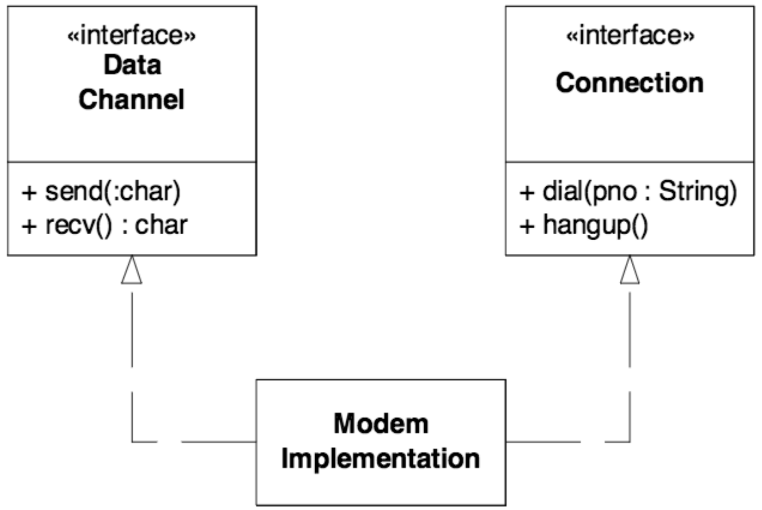


- The behavior of **Square** is not consistent with the behavior of **Rectangle**

- Following the LSP helps ensure that class hierarchies are designed correctly, which facilitates code maintenance and extension

Interface Segregation Principle (ISP) SOLID

- ISP states that no client should be forced to depend on methods it does not use
- Multiple client-specific interfaces are better than a single “fat” and/or “general-purpose” interface
- Split “fat” interfaces into several interfaces – “fat” interfaces are not cohesive
- ISP is a combination of:
  - **SRP**
  - **Polymorphism**
  - **High Cohesion**

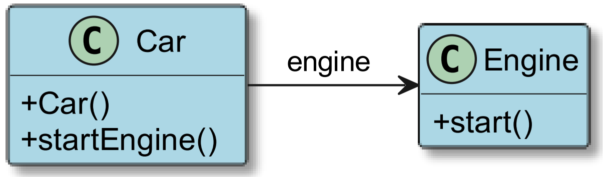


Dependency Inversion Principle (DIP) (1/4) SOLID

- DIP states that:
  - High-level modules should not depend on low-level modules
  - Both should depend on abstractions rather than concrete implementations

Dependency Inversion Principle (DIP) (2/4) SOLID

- DIP states that:
  - High-level modules should not depend on low-level modules
  - Both should depend on abstractions rather than concrete implementations



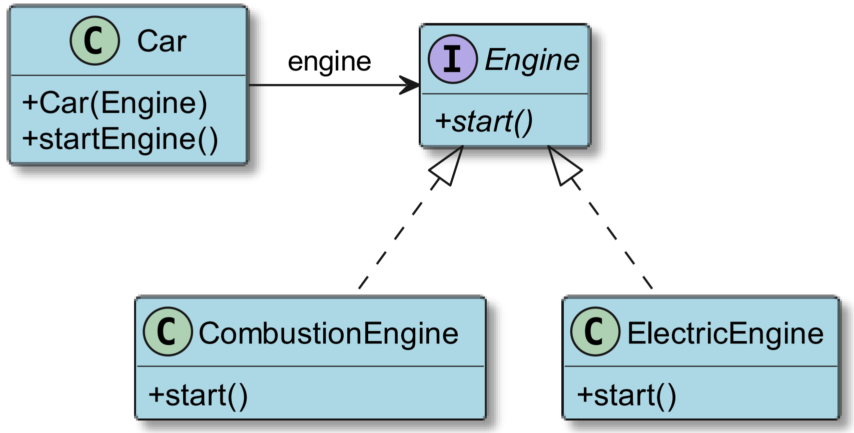
```
public class Car {
    private Engine engine;
    public Car() {
        engine = new Engine();
    }
    public void startEngine() {
        engine.start();
    }
}

public class Engine {
    public void start() {
        // Engine starting...
    }
}
```

The Car class directly creates an instance of the Engine class, making it **tightly coupled to the implementation**.

It violates DIP !!

Dependency Inversion Principle (DIP) (3/4) SOLID



It adopts DIP !!

```
interface Engine {
    void start();
}

public class CombustionEngine implements Engine {
    public void start() {
        // Combustion engine starting...
    }
}

public class ElectricEngine implements Engine {
    public void start() {
        // Electric engine starting...
    }
}

public class Car {
    private Engine engine;
    public Car(Engine engine) {
        this.engine = engine;
    }
    public void startEngine() {
        engine.start();
    }
}
```

The Car class depends on the Engine interface, making it possible to **easily change the implementation**.

Dependency Inversion Principle (DIP) (4/4) SOLID

- OOP Good Practice:
  - **"You should code to interfaces, not implementations."**

- Programming to interfaces promotes a lower/weaker coupling and makes applications more extensible, testable and flexible

Summary

- The SOLID principles are related to GRASP
- SOLID and GRASP are not laws
  - See them as tools
  - Combine them to design and code better systems

References & Bibliography

- Larman, Craig; Applying UML and Patterns; Prentice Hall (3rd ed.); ISBN 978-0131489066
- Ashutosh Krishna (2023). What is SOLID? Principles for Better Software Design. Available on: https://www.freecodecamp.org/news/solid-principles-for-better-software-design
- William Durand (2013). From STUPID to SOLID code! Available on: https://williamdurand.fr/2013/07/30/from-stupid-to-solid-code/