

# SOLID: Liskov Substitution Principle

Explore the Liskov Substitution Principle and how it guides the use of inheritance.

We'll cover the following ^

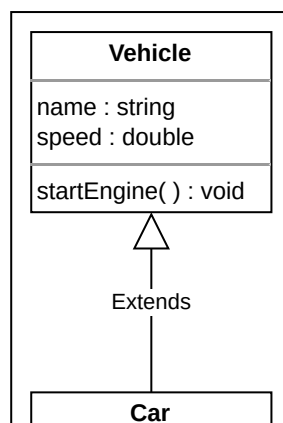
- Introduction
- The Vehicle class
  - Violation
  - Solution
- Conclusion

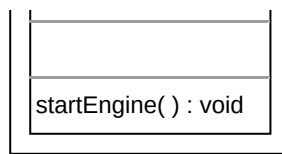
## Introduction

The **Liskov Substitution Principle (LSP)** is one of the fundamental design principles of object-oriented design. The LSP helps guide the use of inheritance in design so that the application does not break. It states that the objects of a subclass should behave the same way as the objects of the superclass, such that they are replaceable. This rule generally applies to abstraction concepts like inheritance and polymorphism.

## The Vehicle class

Let's construct a simple class called **Vehicle** that has some attributes and methods and a subclass **Car** that extends it as shown below:



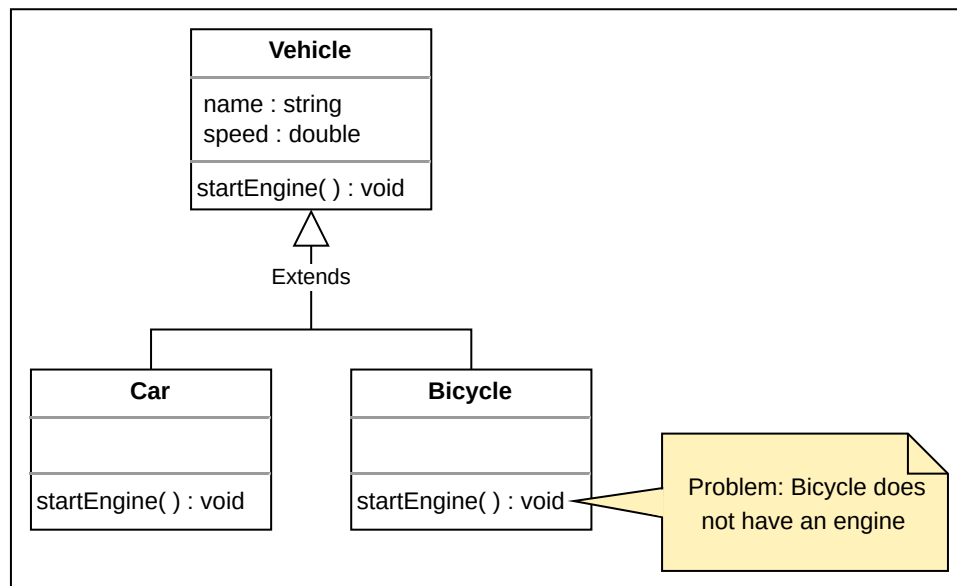


The vehicle superclass

So far, this implementation seems right since a car IS A vehicle, and the `startEngine()` method will override the superclass method. However, it's not as simple as it looks.

## Violation

Let's add a **Bicycle** subclass in this system and see what happens:

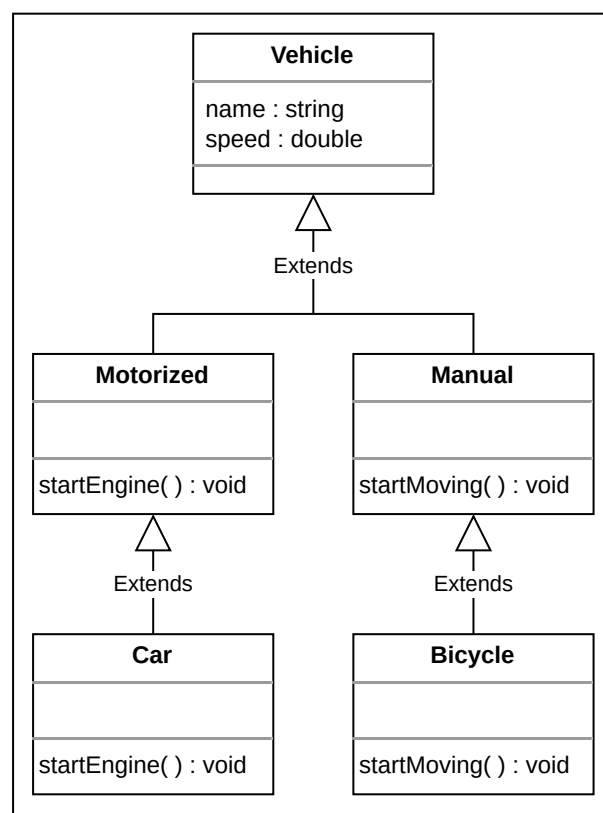


Violation of the LSP

This results in a problem. A bicycle is a vehicle, but it does not have an engine. Therefore, the **Bicycle** class should not be allowed to override the `startEngine()` method.

## Solution

A possible fix to this issue would be to add two subclasses of **Vehicle** that classify the vehicles as motorized vehicles and manual vehicles as follows:



An example of the LSP implementation

With this implementation, we have satisfied the LSP.

- **Car** is substitutable with its superclass, **Motorized**, and **Bicycle** is substitutable with its superclass, **Manual**, without breaking the functionality.
- Their methods can also override the methods of the superclass.

## Conclusion

The LSP is an important principle that should be extended to the level of system architecture. A small violation of the substitutability of classes can cause the system to break down, which is why we should always be on the lookout for violations. A few benefits of the LSP are provided below:

- It avoids the generalization of concepts that may not be needed in the future.
- It makes the code maintainable and easier to upgrade.

Now that we have learned about the Liskov Substitution Principle, let's look at the Interface Segregation Principle in the next lesson.

← Back

SOLID: Open Closed P...

Next →

SOLID: Interface Segr...

☐

Mark as  
Completed

---