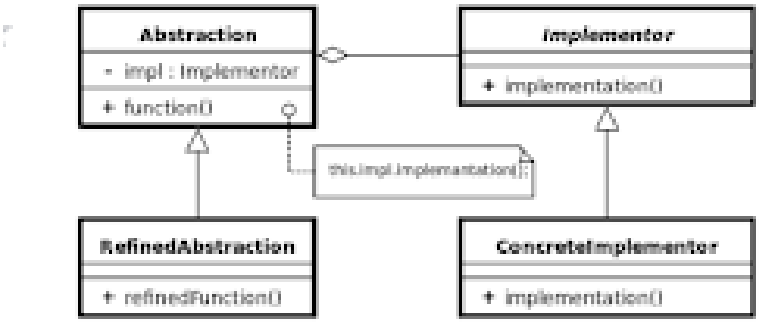
Bridge Pattern

Intent of the DP

The Bridge design pattern is a structural design pattern used for decoupling abstraction from implementation. It is useful in cases when both the class(the abstraction) and what it does(the implementation) vary often. It can be seen as a two-layered abstraction.

Explanation

UML diagram



Abstraction - defines the abstract class and keeps an Implementor reference(for example a vehicle class)
RefinedAbstraction - extends the Abstraction class(there can be more of those, for example bike and bus extend vehicle)
Implementor - defines the interface for implementation classes(for example a workshop)
ConcreteImplementor - implements a behaviour for Implementor(for example asemble and produce for workshop)

Solid principles

It supports the Single responsibility principle as it creates as an implementations has a single task.
It supports the Dependency Inversion principle as objects here depend upon abstractions, not concretions.
It supports the Open-Closed principle as it is open to extension by adding different implementations of the behavior, but closed to changing the core.
More detailed explanations and examples:
https://www.tutorialspoint.com/design_pattern/bridge_pattern.htm
https://www.geeksforgeeks.org/bridge-design-pattern/
https://dzone.com/articles/design-patterns-bridge

Related patterns

The Bridge pattern is a composite of the Template and Strategy patterns.

Strategy pattern

The UML class diagram for the Strategy pattern is very similar to the diagram for the Bridge pattern. However, these two design patterns aren't the same in their intent. While the Strategy pattern is meant for behavior, the Bridge pattern is meant for structure.

Template pattern

In Template pattern, an abstract class exposes defined way(s)/template(s) to execute its methods. The part which bridge takes from template is the abstraction and refined abstraction, but bridge implements the behaviour separately.

Common situations of use

The Bridge pattern is commonly used as a way to have an extra interface between different systems.

Some examples:

In Java:

- JDBC - communicates with a database via a Driver Bridge
- Swing - uses a Bridge to communicate with the Operating System's UI

Similar patterns

Adapter

At first sight, the Bridge pattern looks a lot like the Adapter pattern in that a class is used to convert one kind of interface to another. However, the intent of the Adapter pattern is to make one or more classes' interfaces look the same as that of a particular class. The Bridge pattern is designed to separate a class's interface from its implementation so that the implementation can be modified without changing the client code.