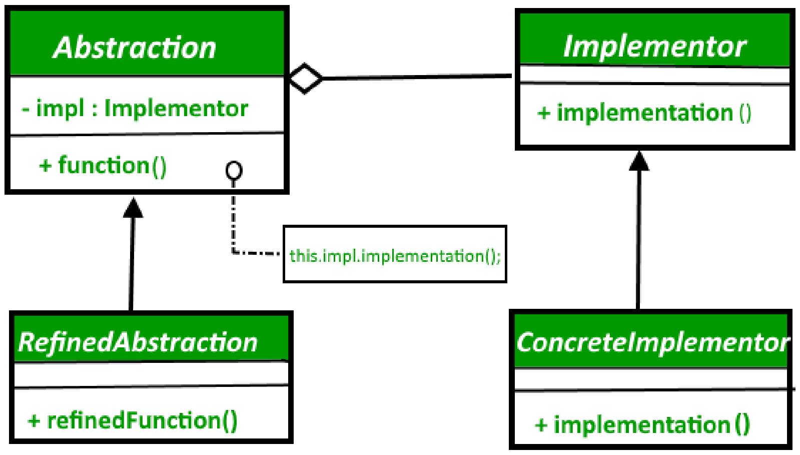
Bridge Design Pattern

Description

Bridge is a Design Pattern that allows you to separate (decouple) an abstraction from its implementation, so the two can vary independently. It is categorized as a structural design pattern as it decouples the implementation class from the abstract class by providing a bridge structure. This “bridge” consists of an interface, making the functionality of the concrete classes independent from the abstraction. In this way, both the abstraction and its implementations can be altered structurally without affecting each other.

The Bridge pattern is an application of the “prefer composition over inheritance” advice.

UML Diagram



- both the Abstraction and the Implementor can be an abstract class or an interface.
- the Abstraction contains a reference to the implementor
- the children of the abstraction are refered as the refined abstractions or concrete classes.
- the children of the Implementor are refined as concrete implementors or implementations.

- since we can change the reference to the implementor in the abstraction, we are able to change the abstraction’s implementor at run-time

When should you use it?

You should use the bridge pattern when:

- you want run-time binding of the implementation
- you want to share an implementation among multiple objects
- you need to map orthogonal class hierarchies (the two have no overlapping methods or data)
- you want to decouple an abstraction from its implementation so that the two can vary independently
- you have a lot of classes resulting from a coupled interface and numerous implementations

Disadvantages

Some throwbacks of this pattern are:

- a set of interfaces with minimal or only one implementations makes it hard to be managed
- since it relies on a “HAS-A” composition, it could chain multiple method calls to perform an operation leading to multiple stack operations and a slight latency (multiple indirection)
- details are hidden from the clients