Bridge Pattern

> Decouple an abstraction from its implementation so that the two can vary independently.

- Bridge provides a way to extend two sides of a relationship, independent of each other.
- The force behind the Bridge pattern is OCP.
- The principle used in the solution of the bridge pattern is DIP.

When to use Bridge Pattern?

- When you want to avoid a permanent binding between an abstraction and its implementation — especially when implementation may be selected or switched at runtime.
- Both the abstractions and their implementations should be extensible by subclassing.
- Change in the implementation of an abstraction should not impact the clients — no recompilation of client code.
- In C++, you want to hide the implementation from the .h file.
- Avoids proliferation of classes.

Consequences of using Bridge

- Decoupling interface and implementation — may be configured at runtime — may even be changed.
- Eliminates compile time dependency on implementation.
- Encourages layering — resulting in better system.
- Improved extensibility.
- Shields clients from implementation details.

Bridge vs. Other Patterns

- Abstract Factory can create and configure a particular Bridge.
- Different from Adapter Pattern:
  - Adapter → making unrelated classes work together — usually applied to systems after redesign.
  - Bridge → lets abstraction and implementation vary independently — used up-front in design.