

SERKUT YILDIRIM commented on this

...



Shirin Monzavi • 1st

Senior .NET Backend Developer | Microservices | ASP.NET Core | Clean Code | Open to Remote or Reloca...
2d •

Design Pattern of the Week: Bridge

Motivation

Decouple an abstraction from its implementation so that both can evolve independently.

Real-World Scenario

When an abstraction can have multiple implementations, the typical approach is inheritance. But this leads to some issues.

Let's say we have an abstract class `Window`, with two implementations: `XWindow` and `PMWindow`. If we use inheritance and want to add a new type like `IconWindow`, we'll need to create `XIconWindow` and `PMIconWindow`. This leads to:

1 ***Class explosion*** – For every new window type, we must implement it for each platform.

2 ***Tight coupling*** – The client is forced to choose a concrete platform (`XWindow` or `PMWindow`), tying the code to a specific implementation.

Solution — Bridge Pattern

The Bridge pattern solves this by splitting the abstraction (`Window`, `IconWindow`, `TransientWindow`, etc.) from the implementation (`WindowImp`, `XWindowImp`, `PMWindowImp`, etc.).

The abstraction holds a reference to the implementation, and they communicate through a defined interface. This bridges the two hierarchies, enabling independent evolution and better flexibility.

Related Patterns

Abstract Factory can help create and configure a particular Bridge.


Adapter lets unrelated classes work together, but it's applied after design.

Bridge is a design-time decision to decouple abstraction from implementation.

Code Example

See it in action:

 GitHub - <https://lnkd.in/dyqftVZC>

 *Your Turn!*

Have you ever used the Bridge Pattern in your projects ?

[#DesignPatterns](#) [#SoftwareEngineering](#) [#DevTips](#) [#BridgePattern](#)



SERKUT YILDIRIM and 37 others

5 comments 1 repo...
