Bridge ID

Student:

Izabela Kuźniar

Teacher:

Andrea Corradini

Course:

Software Design Patterns

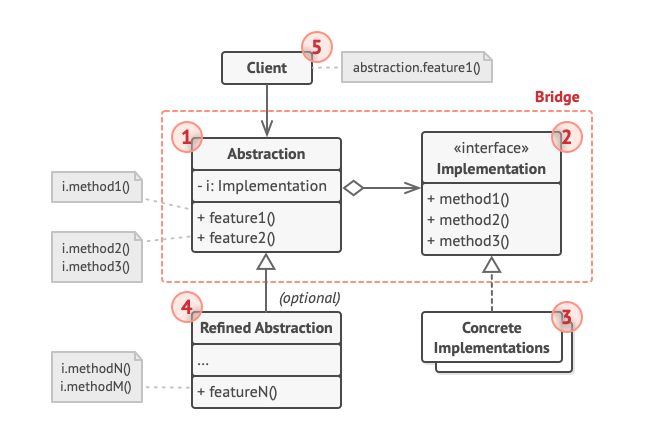
# Name and category

Bridge is a structural pattern.

# Intent:

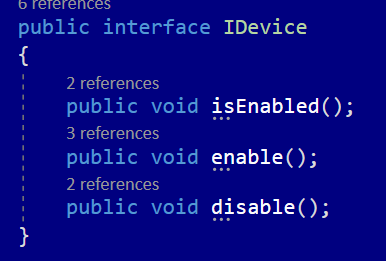
Introduces loosely coupled classes. Separates classes into two hierarchies: abstraction and implementation.

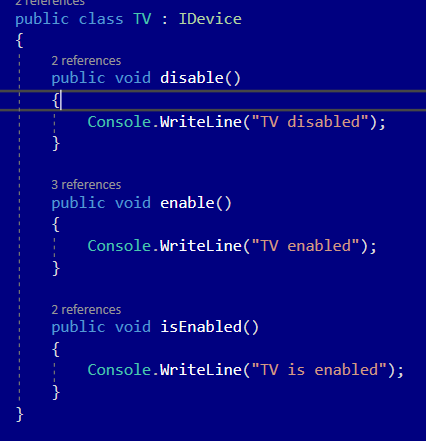
# Structure as a UML class diagram

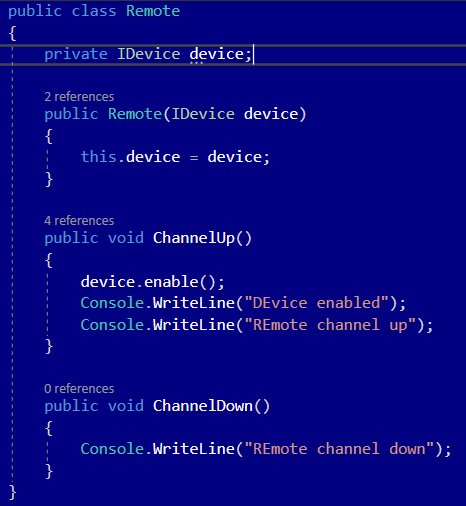


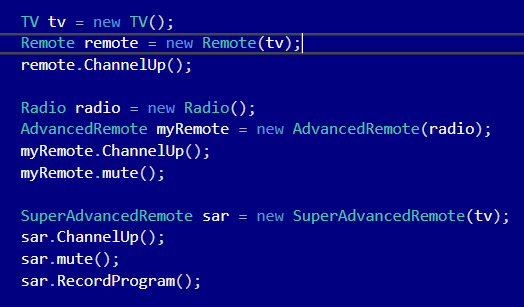
# Implementation:

To implement Bridge Pattern we need an Implementation interface, in my case it was IDevice. Then we could create many types of devises inheriting from this interface, such as TV or Radio. Then we needed and Abstraction that would hold the implementation. I created Remote class that could work with any IDevice. Then we could develop more and more advanced classes from the Remote class that would inherit it, such as AdvancedRemote and Super AdvancedRemote.









# Consequences:

Benefits:

* Ability to create platform-independent classes and apps
* Client-code works with high-level abstraction. Is not exposed to platform details.
* Open/Closed Principle. Ability to introduce new abstractions and implementations independently from each other.
* Single Responsibility Principle. Ability to focus on high-level logic in the abstraction and on the platform details in the implementation.

Drawbacks:

* Possibility of making the code even more complicated by applying it to highly cohesive class.

# Known uses

* Used in JWT Windows to separate components such as buttons.
* JDBC: Java to SQL DB driver

# Related patterns

1. Bridge is usually designed up-front, letting you develop parts of an application independently of each other. On the other hand, Adapter is commonly used with existing app to make some otherwise-incompatible classes work together nicely.
2. Bridge, State, Strategy (and to some degree Adapter) have very similar structures. Indeed, all of these patterns are based on composition, which is delegating work to other objects. However, they all solve different problems. A pattern isn’t just a recipe for structuring your code in a specific way. It can also communicate to other developers the problem the pattern solves.
3. You can use Abstract Factory along with Bridge. This pairing is useful when some abstractions defined by Bridge can only work with specific implementations. In this case, Abstract Factory can encapsulate these relations and hide the complexity from the client side.
4. You can combine Builder with Bridge: the director class plays the role of abstraction, while different builders act as implementations.