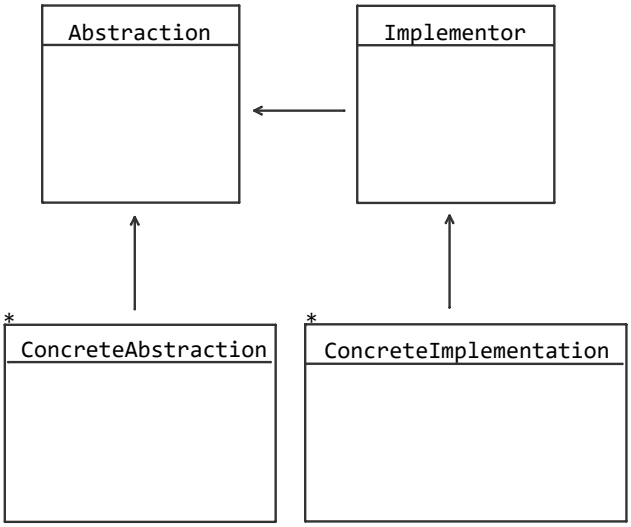
Bridge Pattern

06 June 2021 10:42 PM

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




Bridge pattern is useful when we don't want the implementation to change because of the changes in the abstraction

Bridge Pattern is similar to Adapter Pattern
BUT...the difference is about the intent and why we use it...
Adapter Pattern is used as a precautionary method while the Bridge pattern is used to prevent such situations

Intent behind using this pattern

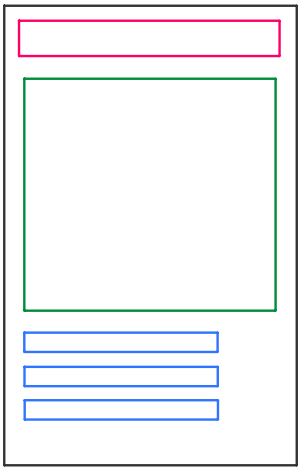
S1={A,B,C,D}
S2={1,2,3,4}

Cartesian Product -> (A,1)(A,2)
(A,3)(A,4)(B,1)(B,2)(B,3)(B,4)
(C,1)(C,2)(C,3)(C,4)(D,1)(D,2)
(D,3)(D,4)

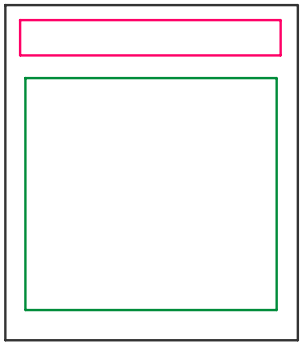
Just like the Cartesian Product we should be able to map any ConcreteAbstraction to any ConcreteImplementation.

Using this pattern the number of Classes created reduces exponentially.

Long Form View



Short Form View



Suppose we have a long form view and a short form view, and we have a set of resources like albums, books, artists, podcasts.

Situation 1: We create all the matches of the classes like
AlbumsLongFormView
AlbumsShortFormView
BooksLongFormView
BooksShortFormView
ArtistsLongFormView
ArtistsShortFormView
PodcastsLongFormView
PodcastsShortFormView
We'll have to create 8 concrete classes
Whereas, if we use Bridge Pattern. We'll have to create only 6 concrete classes:
LongFormView
ShortFormView

AlbumsResource
BooksResource
ArtistsResource
PodcastsResource

This gap increases exponentially as more and more views and resources adds up.

Moreover, the bridge pattern allows flexibility to modify and update a part of code without affecting the other parts.