5. Adapter & Facade

Structural Design Patterns

- Structural design patterns focusses on organising and structuring classes & objects to create more structural & maintainable software systems.
  - What class will be there
  - What are the attributes that should be there in a class
  - How different classes communicate with each other

Adapter Design Pattern

- In real world,
  - Some of the code is written in our software system.
  - Some of the code is outsourced to third parties. We use multiple third party libraries, SDK's, API's in our system.
  - Example: EntityFramework, FluentValidation, Lombok, Auto-mapper etc.,

- Problems in using third party libraries in our code directly,
  - If we plan to change the particular third party provider? Changing from Entity framework to Dapper
  - If the third party services goes out of maintenance?

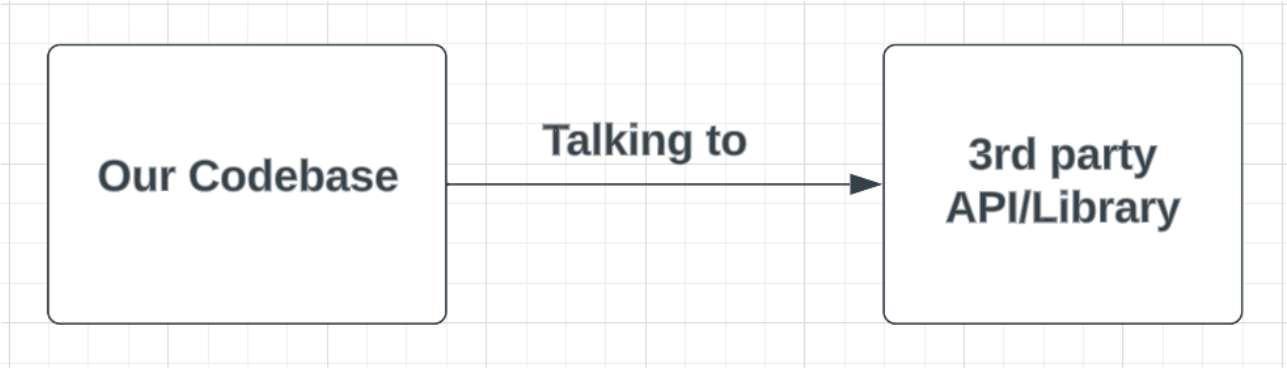
- If our codebase is directly talking to 3rd party libraries, it involves a lot of tight coupling between our codebase & the 3rd party library. It can affect maintainability.
- Whenever we are connecting to a 3rd party API, Never connect directly, always use an Interface in between. To follow the Dependency Inversion principle, which says Never code to concrete classes, code to interface.

How to use an Adapter?

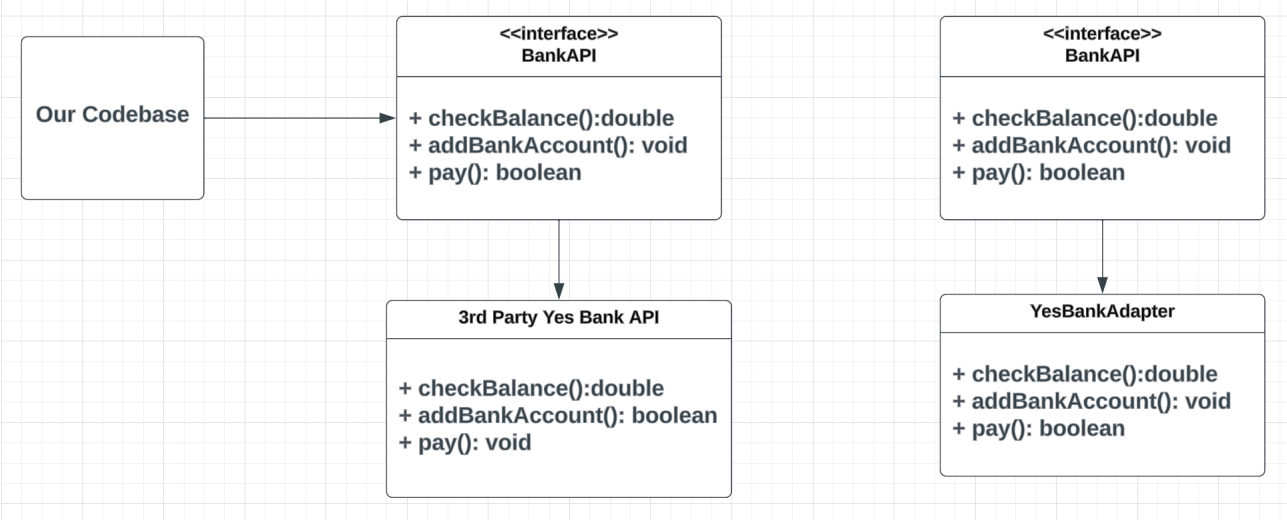
- Whenever we connect to a 3rd party API, create an interface & decide the methods or logics that we are going to perform using the 3rd party library. Have a concrete implementation for the interface which performs the actual logic using the 3rd party library.

- Even if move to a different 3rd party service provider in the future, the existing codes won't be affected, the only thing to do is, create a new subclass from the interface.

Avoid this



Use an adapter in between



- The concrete class of **YesBankAdapter** in our code base will be using the actual external 3rd party Yes bank API's. Those intermediate layer between the 3rd party Yes bank API's & our code base -> YesBankAdapter is the adapter design pattern.

When to use Adapter design pattern?

- Whenever we are talking to 3rd parties (SDK's, Libraries, API's)

Facade Design Pattern

- Whenever we find a method/class doing too much of work, instead of doing the work in the class, create a helper class to do that actual work.