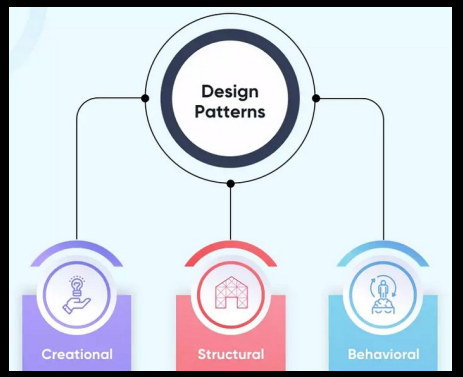


Structural Design Patterns

Ajith de Silva @epita.fr | 2025

Structural Design Patterns :: About

How to assemble objects into LARGE structures

Keep structures

… Flexible

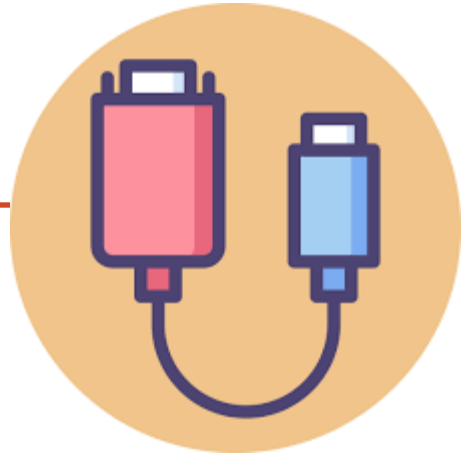
… Efficient

… Maintainable

Structural Design Patterns

1. **Adapter**
   - *… Makes incompatible interfaces to communicate.*
2. **Bridge**
   - *… Split large related class into set of related classed*
   - *… Implementation can be done independently*
3. **Composite**
   - *… Arrange objects into TREE structure and work on it*
4. **Decorator**
   - *… Wrap existing object by attaching new behaviors*
5. **Facade**
   - *… Simplify complex interface,class(es), framework*
5. **Flyweight**
   - *… Share common states between multiple objects*

Adapter Design Pattern :: Problem to solve

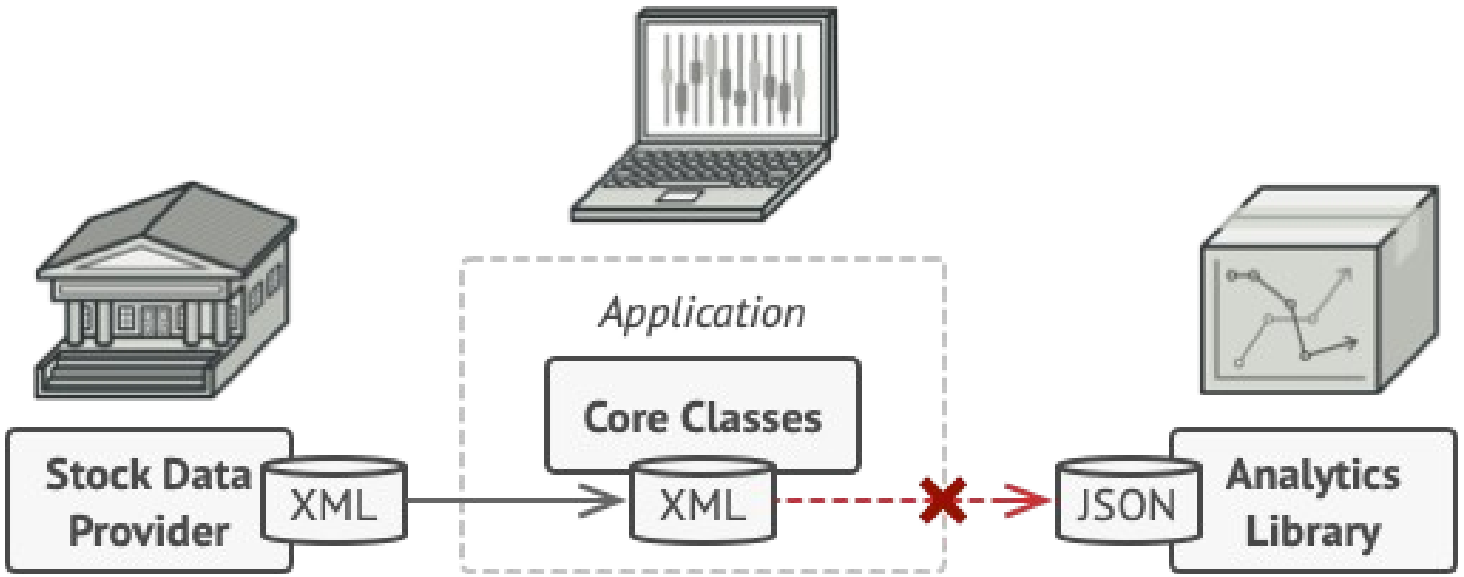


1. **2 external services used by client code.**

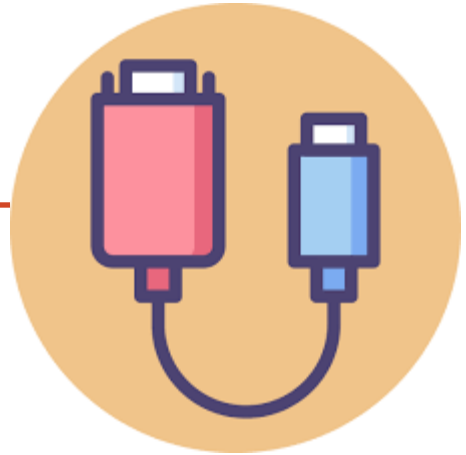
   Both are incompatible

   … XML output from one system

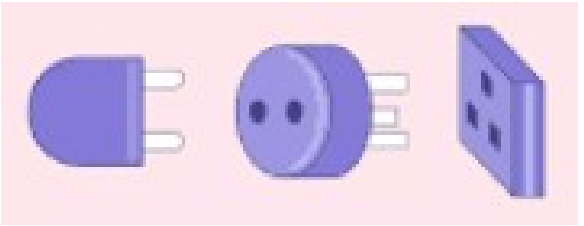
   … JSON input expected by 2nd system



Adapter Design Pattern



1. **In Brief….**
   - **… intermediate/middle object**
   - **… helps to communicate non compatible interfaces**



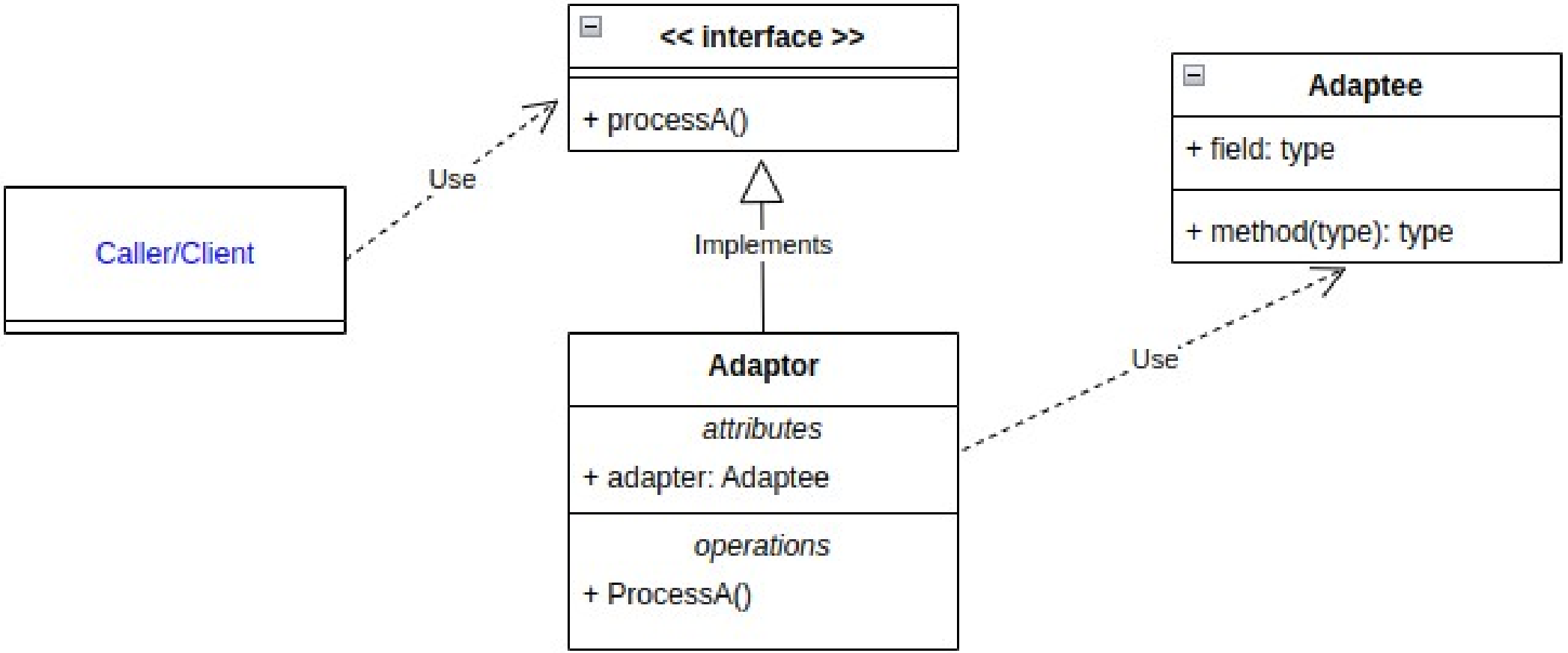
2. **Use existing code/class without modification**

3. **Simplify integration of new components to existing systems**

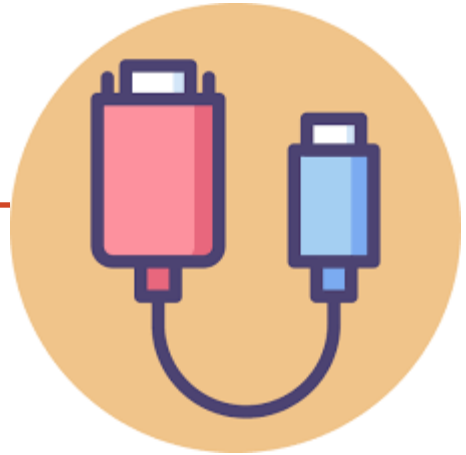
4. **Support multiple targets/interface by changing the adapters**

5. **There will be execution delay since the process is extra work**

Adapter Design Pattern :: UML



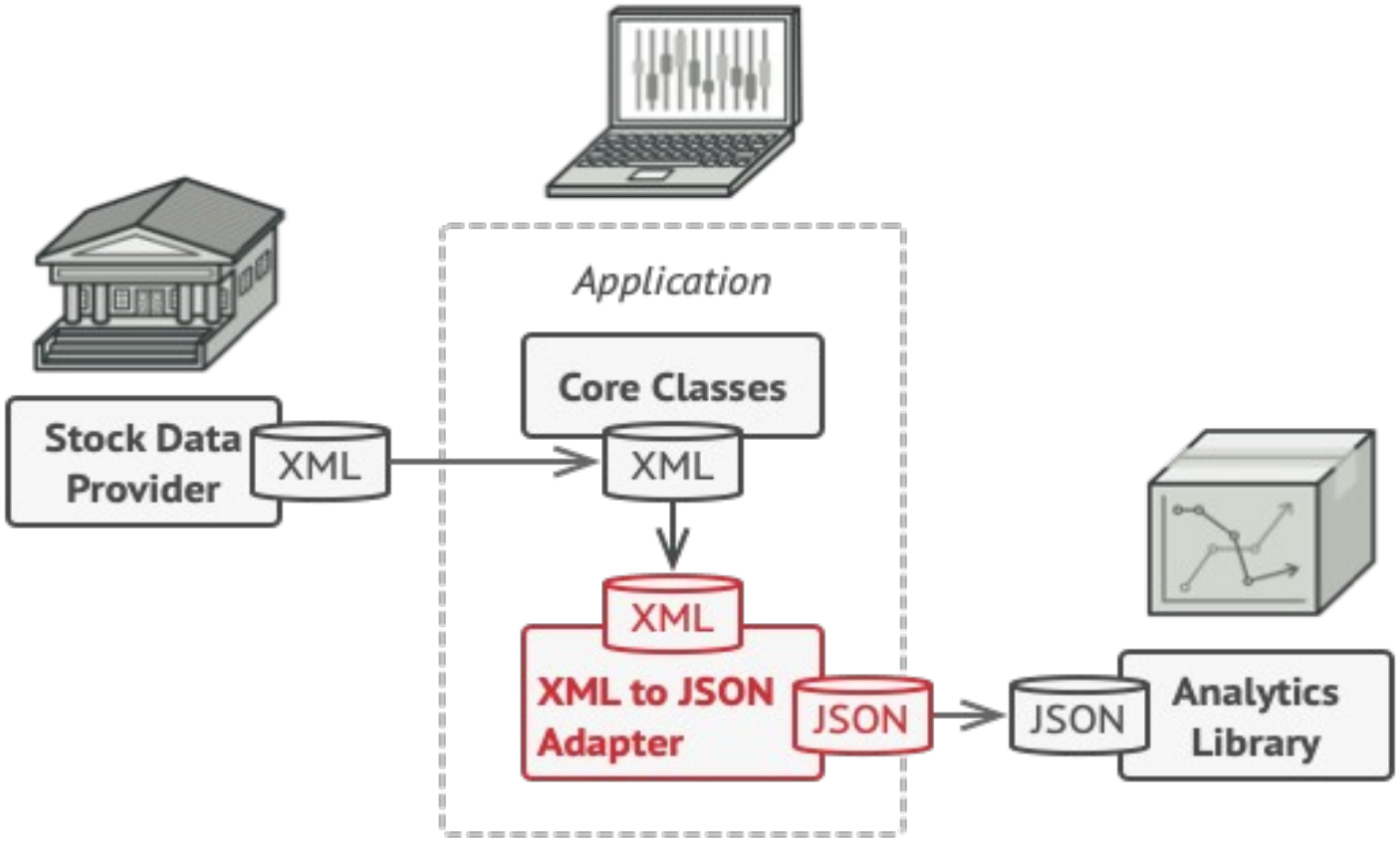
Adapter Design Pattern :: How it works



1. Create a special object that converts

   XML --> JSON

2. Place it in-between 2 systems in your app where applicable
3. Now both 2 external services are able to comminate via the new module (*** ADAPTER)



Adapter Design Pattern :: How to implement

HOW ??

***** Should have service class that you can not change (3rd party/external system)**

Define Abstract/Interface for client

Implement concrete Adapter class by implementing the client interface

Add field to store reference to the service

… initialize it with constructor or pass it via setter or method when using it (FREEDOM)

Implement all required methods

… Will be more complex when you have more parameters

Use adapter by Client's/Calle's code

Adapter Design Pattern :: Practical

1. Check/study the no factory sample
2. Check/study the factory sample
3. Run/Debug and check how it works
3. Extend concrete implementation