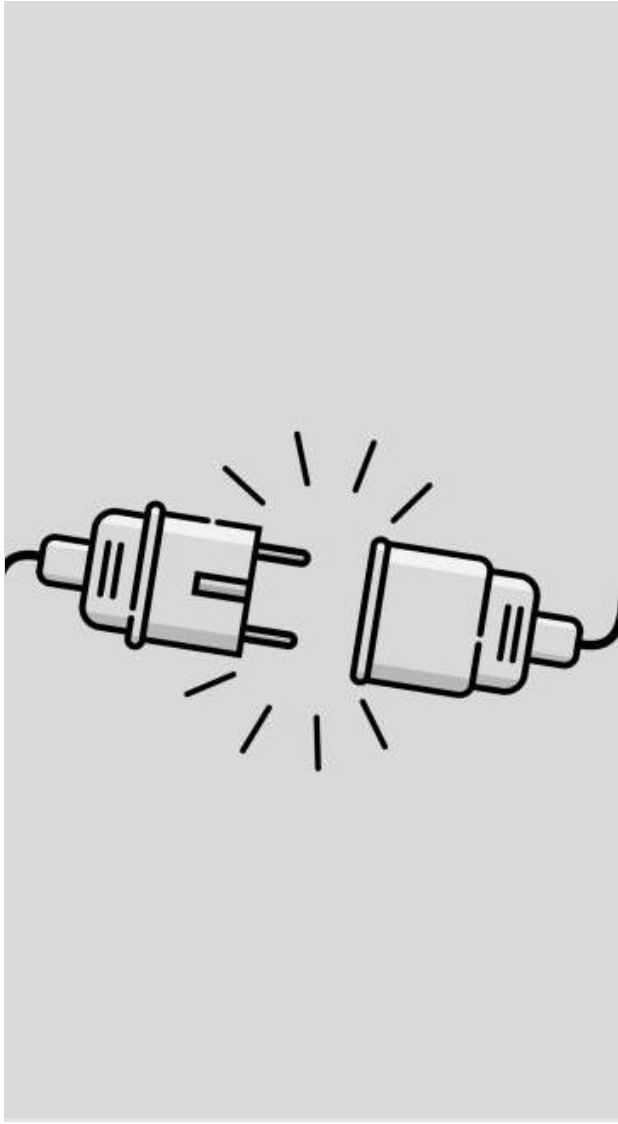
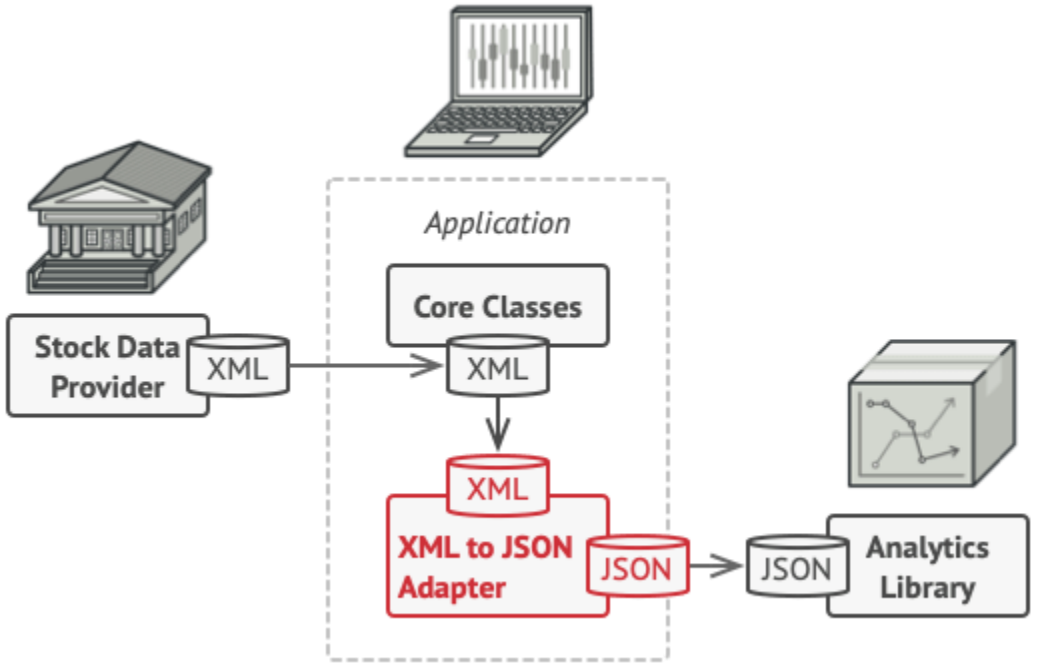
Adapter Design Pattern

Reese Dominguez

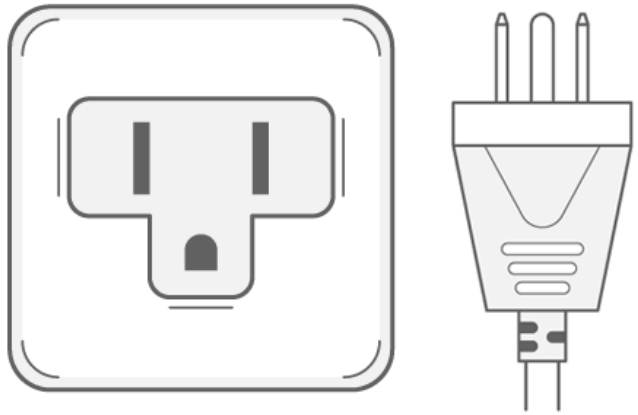


What is it?

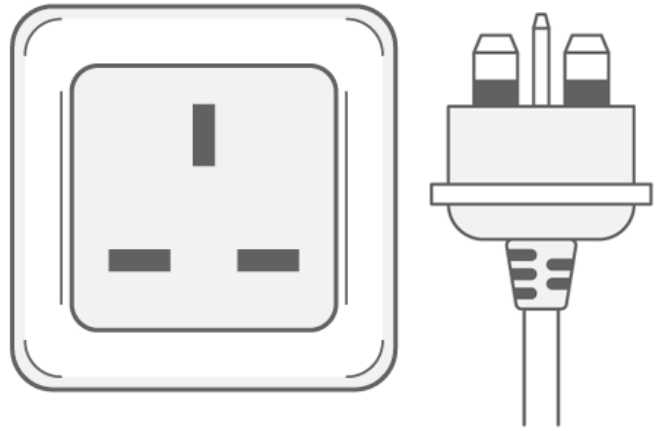
- Structural design pattern
- Allows two incompatible interfaces to work with each other without modifying the source code
- There are two ways to implement it
  - Class and Object adapter



What is it?



North American outlet

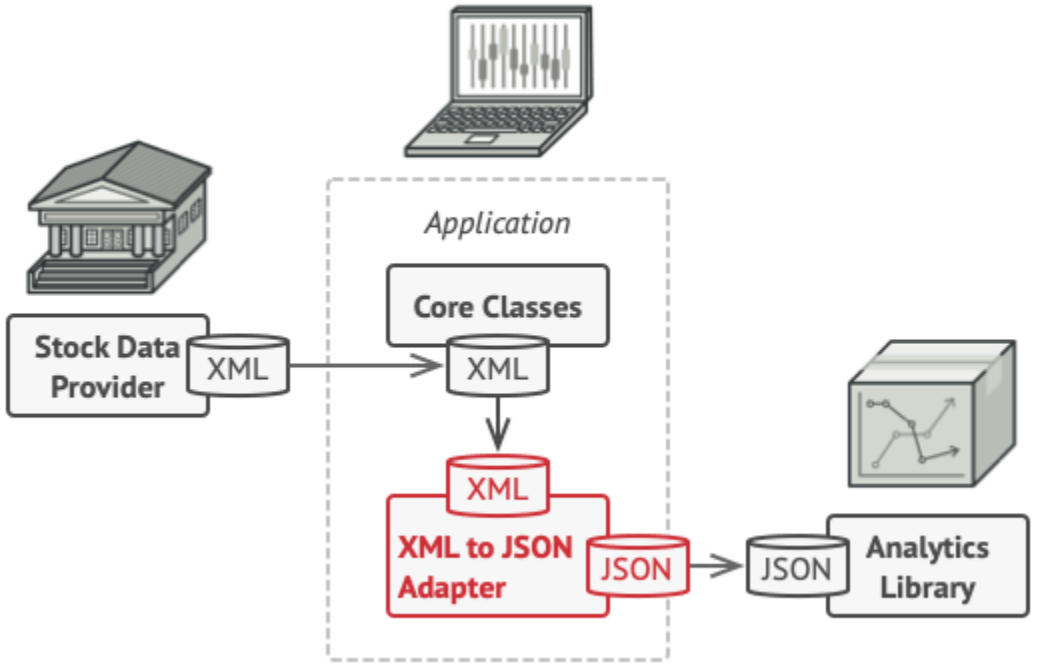


Singapore outlet

- The two plugs are incompatible
- You would need a travel adapter

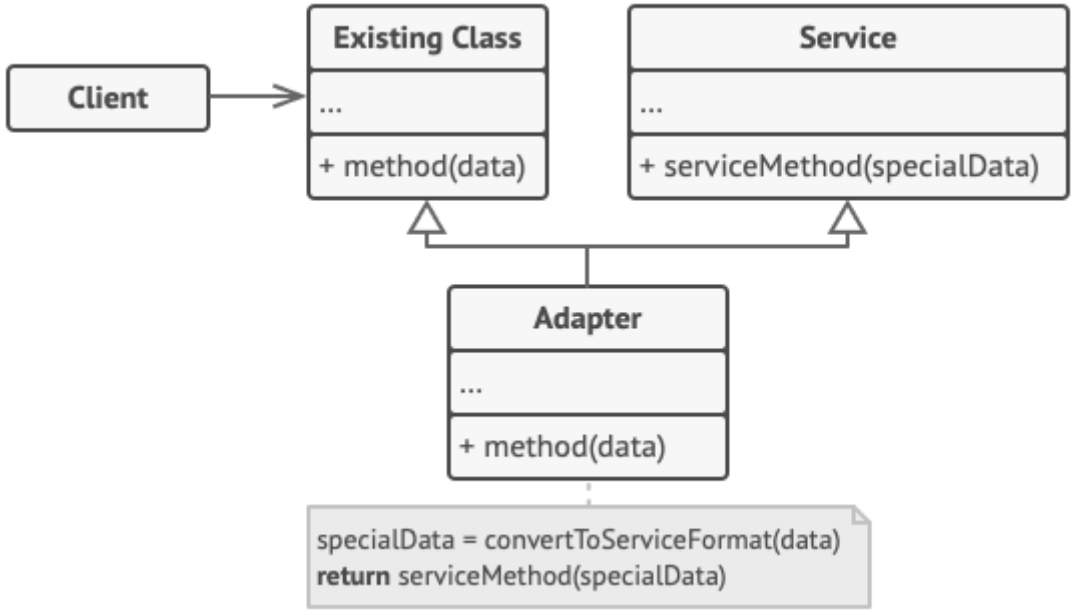
What is it?

- Structural design pattern
- Allows two incompatible interfaces to work with each other without modifying the source code
- There are two ways to implement it
  - Class and Object adapter



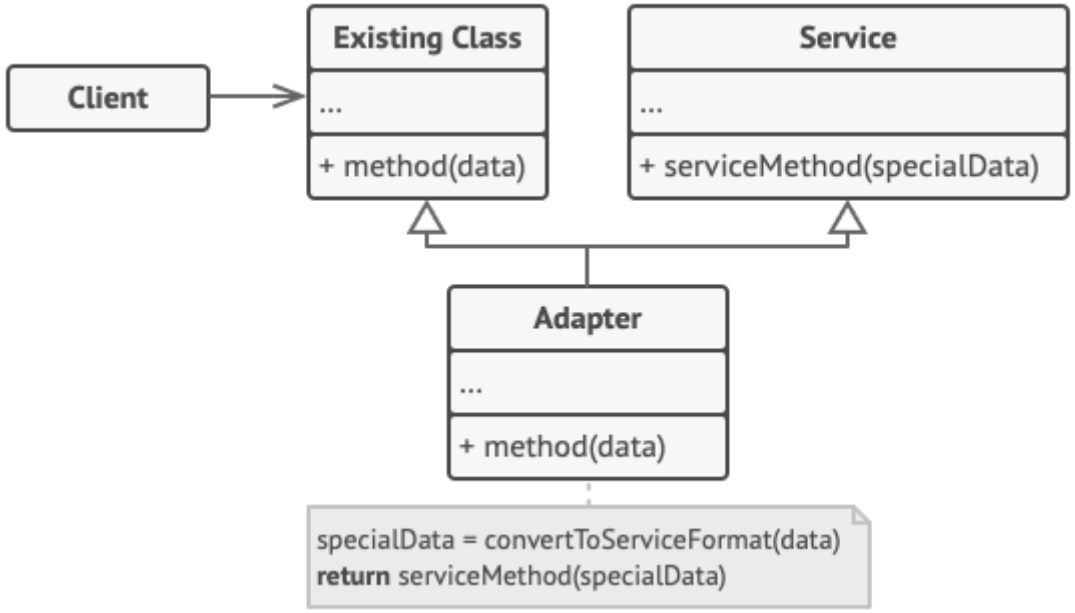
Implementation: Class

- Multiple inheritance needed
  - Adapter inherits from both the client and the service
  - Adapter can be used in place of client class
  - May not be possible in Java (unless one of them is an interface)
- Overridden methods in the adapter handle the adaptation



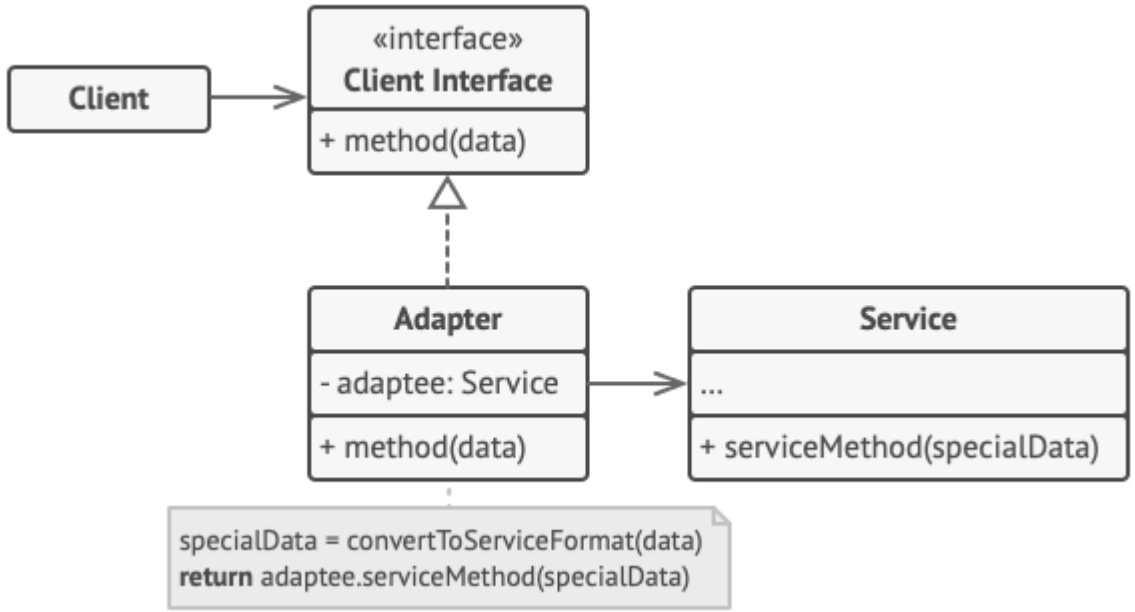
Implementation: Class

- **Benefit:** You don't have to write everything
  - Can override some of the service's/class' behaviour
- **Downside:** You need to rewrite the same adapter for each subclass of the service class, if any



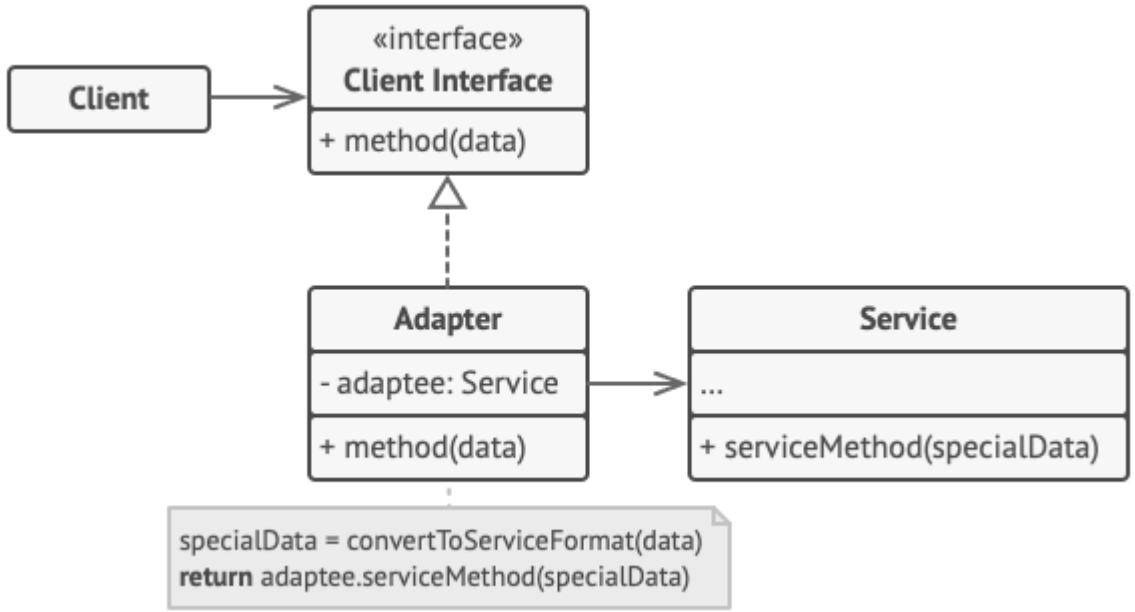
Implementation: Object

- Adapter implements/inherits the Client, wraps service
  - Adapts by delegating calls to the incompatible service to work with the client



Implementation: Object

- **Benefits:**
  - Adapts all the subclasses of the foreign service, if any
  - Loose coupling: the existing code isn't affected as much if the adapter exists or not
- **Downside:** Since it's a wrapper, you can't directly override service behaviour
  - You'll have to create a subclass with the overridden behaviour



When to use it

- Compatibility with 3rd-party/legacy/otherwise incompatible classes
- Add functionality to subclasses that can't be added to the superclass