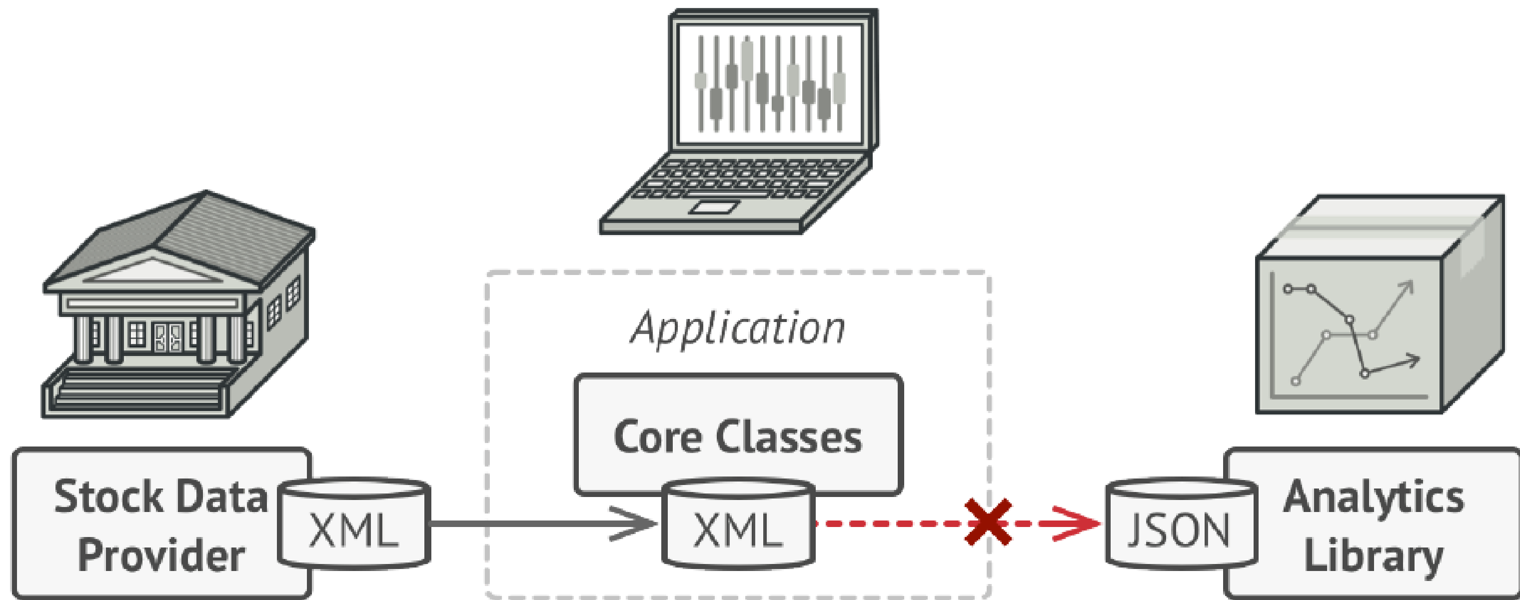
Adapter

also known as: wrapper

- **Adapter** is a structural design pattern that allows objects with incompatible interfaces to collaborate

Problem

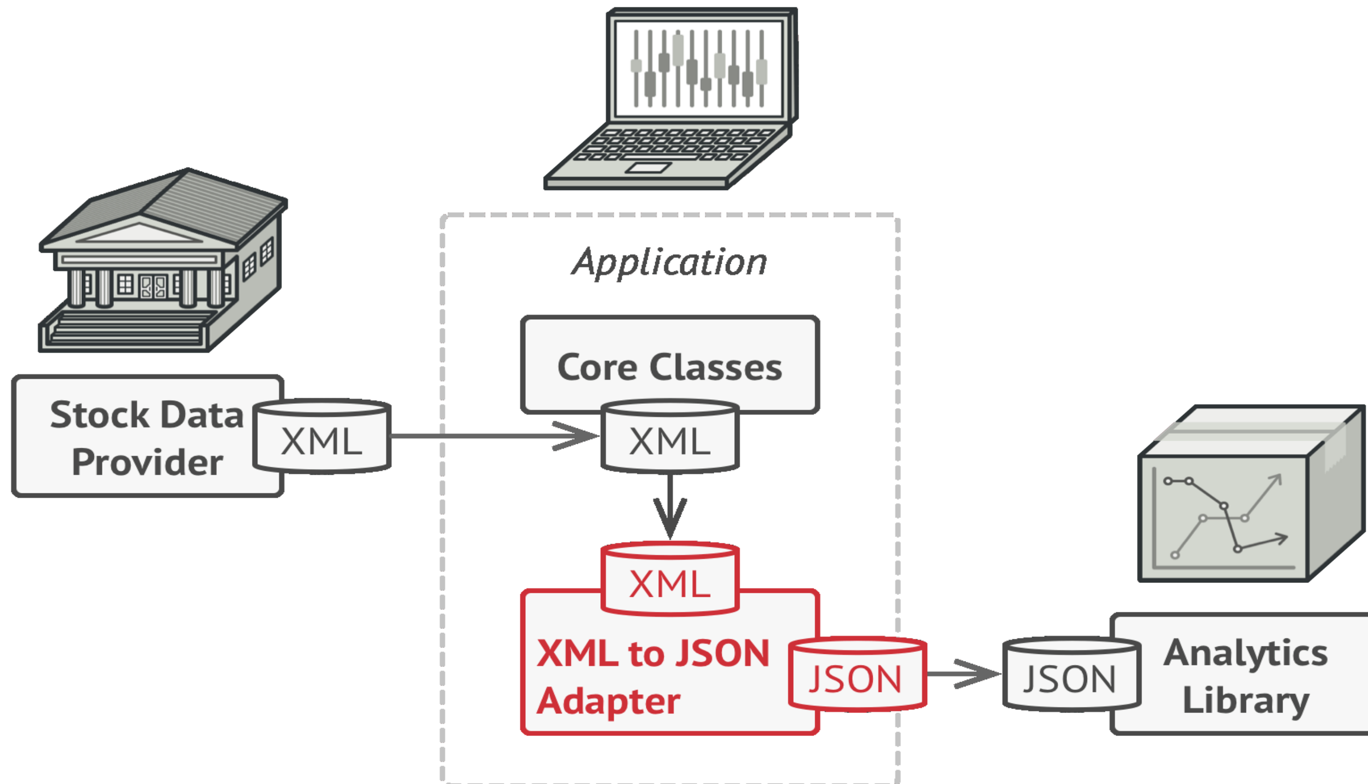


We can't use the analytics library "as is" because it expects the data in a format that's incompatible with our app

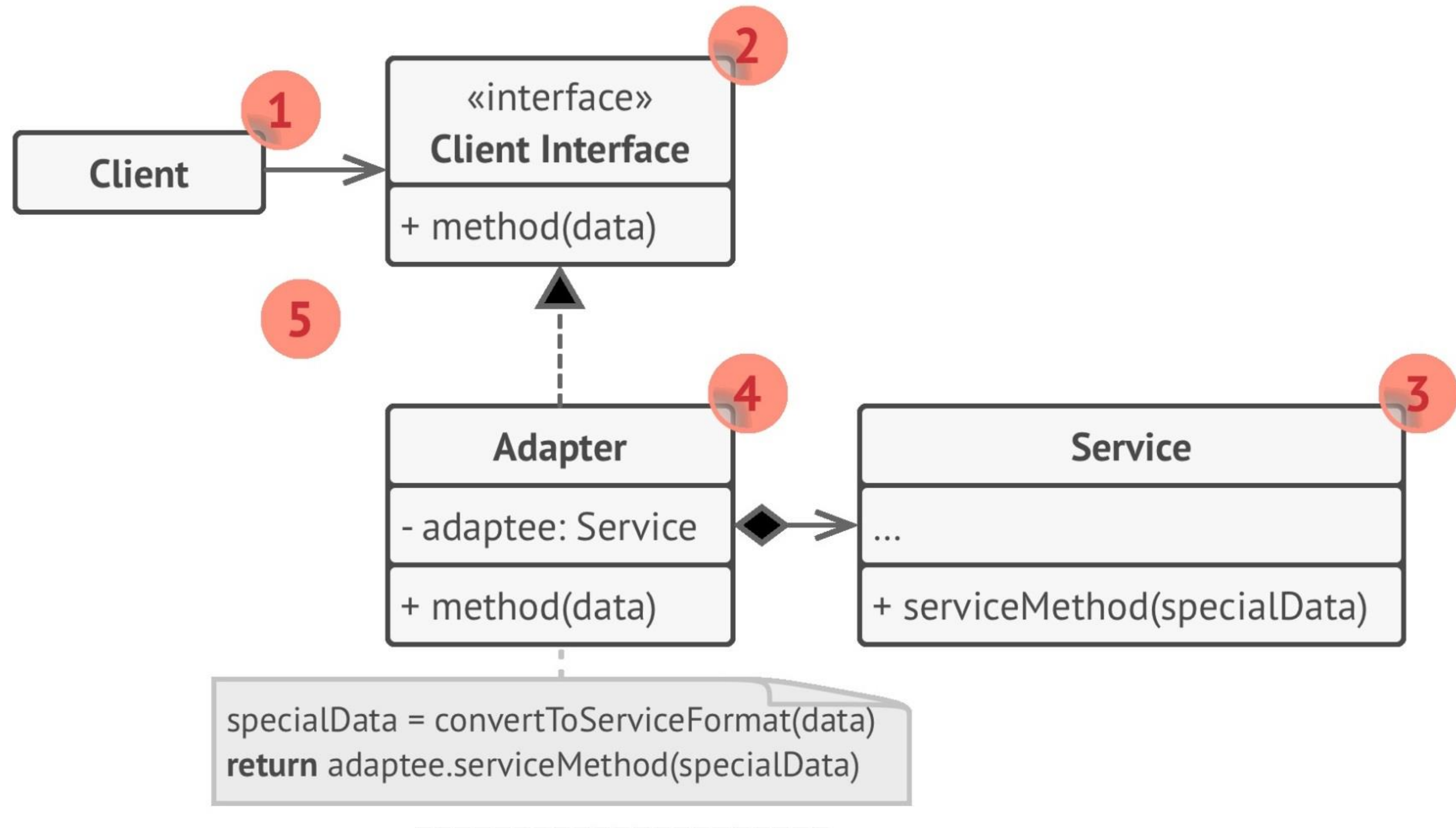
Solution

- We can create an *adapter*. This is a special object that converts the interface of one object so that another object can understand it.
- Adapters can not only convert data into various formats but can also help objects with different interfaces collaborate.
- The adapter gets an interface, compatible with one of the existing objects.
- Using this interface, the existing object can safely call the adapter's methods.
- Upon receiving a call, the adapter passes the request to the second object, but in a format and order that the second object expects.

Contd.



Structure

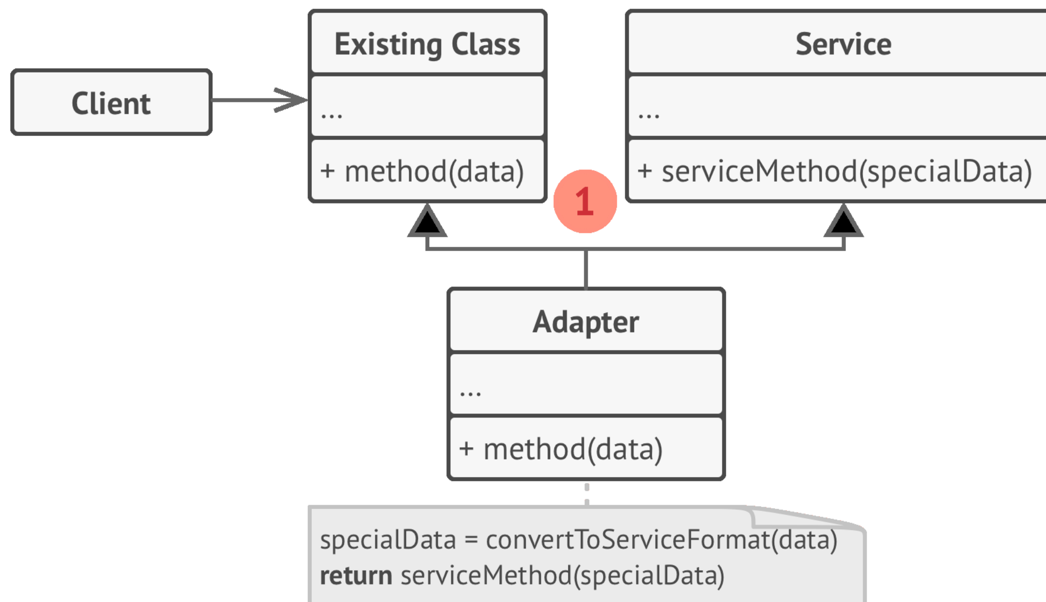


Contd.

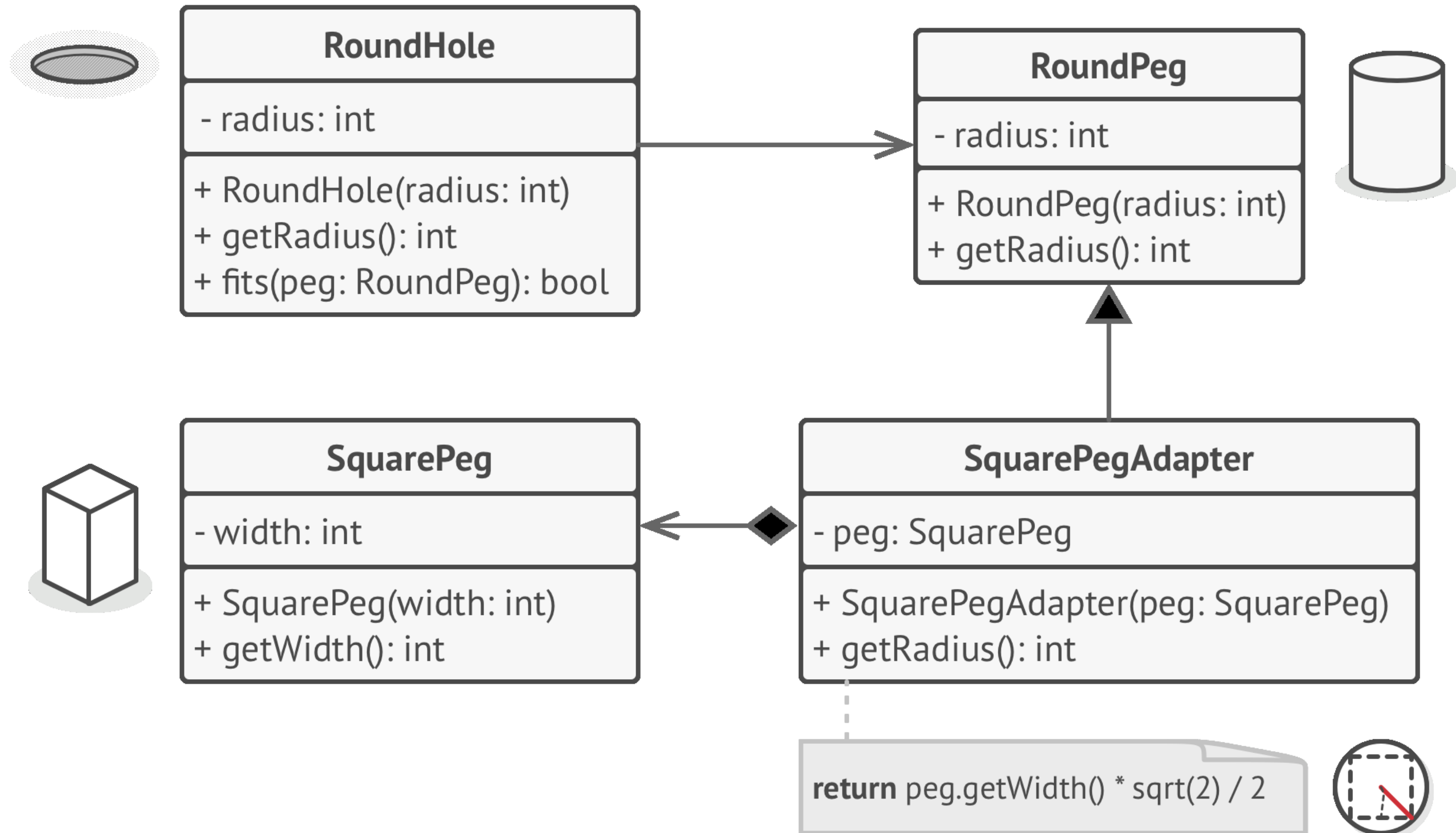
- The **Client** is a class that contains the existing business logic of the program.
- The **Client Interface** describes a protocol that other classes must follow to be able to collaborate with the client code.
- The **Service** is some useful class (usually 3rd-party or legacy). The client can't use this class directly because it has an incompatible interface.
- The **Adapter** is a class that's able to work with both the client and the service: it implements the client interface, while wrapping the service object. The adapter receives calls from the client via the adapter interface and translates them into calls to the wrapped service object in a format it can understand.
- The client code doesn't get coupled to the concrete adapter class as long as it works with the adapter via the client inter-face. For this, we can introduce new types of adapters into the program without breaking the existing client code.
- This can be useful when the interface of the service class gets changed or replaced: you can just create a new adapter class without changing the client code.

Contd.

- The Class Adapter doesn't need to wrap any objects because it inherits behaviors from both the client and the service.
- The adaptation happens within the overridden methods



Pseudo code



Adapting square pegs to round holes.

raktimchakraborty27@gmail.com,
www.oodesign.com

Application

- The Adapter class is used when we want to use some existing class, but its interface isn't compatible with the rest of the code.
- This pattern is used when we want to reuse several existing subclasses that lack some common functionality that can't be added to the superclass.

Pros and Cons

- *Single Responsibility Principle*. We can separate the interface or data conversion code from the primary business logic of the program.
- *Open/Closed Principle*. We can introduce new types of adapters into the program without breaking the existing client code, as long as they work with the adapters through the client interface.
- The overall complexity of the code increases because you need to introduce a set of new interfaces and classes. Sometimes it's simpler just to change the service class so that it matches the rest of your code.