Structural Design Pattern

Adapter

Upcode Software
Engineer Team

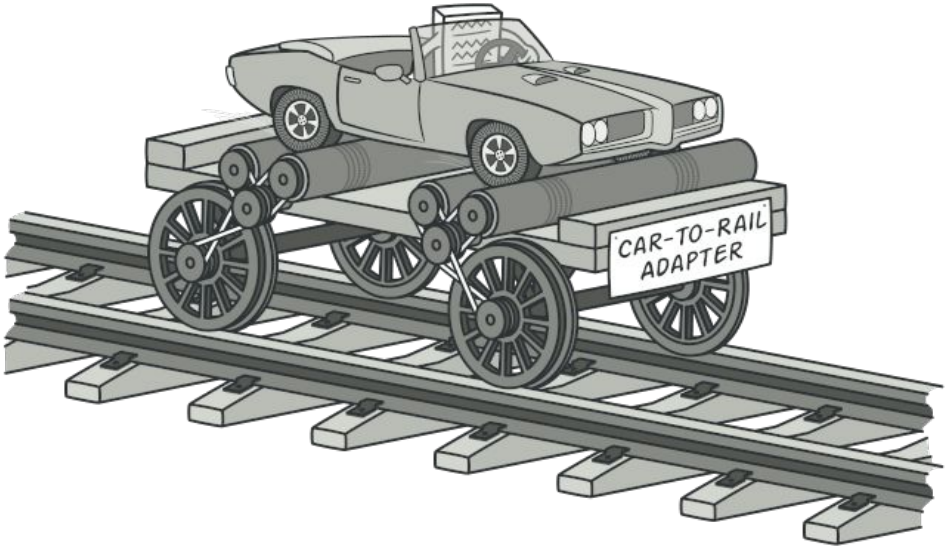
-2023-

CONTENT

1. What is ADAPTER ?
2. WHY needs to ADAPTER ?
3. HOW to use ADAPTER ?
4. WHERE to use ADAPTER ?
5. SOURCE code.

1. What is ADAPTER ?

- **The adapter** (also known as the *Wrapper*) is a structural design pattern that allows your code to communicate with other interfaces that are initially incompatible.



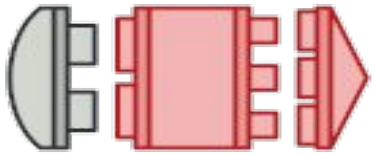
2. WHY needs to ADAPTER ?

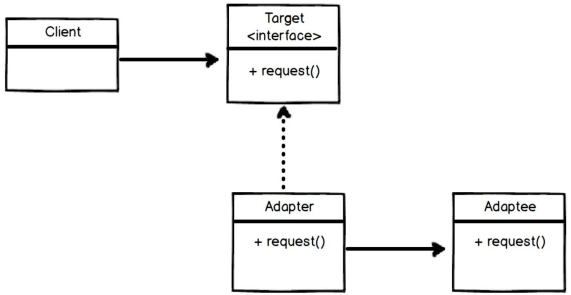
Here are a few reasons why the Structural Adapter Pattern is used:

1. **Legacy system integration:** When integrating a new system with an existing legacy system, there may be a mismatch between the interfaces of the two systems. The Adapter Pattern allows the new system to communicate with the legacy system by adapting its interface to match the expected interface of the legacy system.
2. **Reusability and compatibility:** Different components or modules may have been developed independently with different interfaces. The Adapter Pattern allows these components to be reused by adapting their interfaces to fit together. It promotes compatibility and avoids the need to modify existing code to make it work with other components.
3. **Interface standardization: I**n some cases, it may be necessary to standardize the interface of a class or component to conform to a specific contract or interface definition. The Adapter Pattern can be used to adapt the existing interface to the required standard interface without modifying the original code.

2. WHY needs to ADAPTER ?

4. **Separation of concerns:** The Adapter Pattern helps to separate the concerns of different components. It encapsulates the logic required to adapt the interface of one component into a separate adapter class, keeping the core components clean and focused on their primary responsibilities.
5. **Testing and mocking:** Adapters can be useful in unit testing scenarios. By creating an adapter that mimics the behavior of a real component, you can isolate the system under test and mock the interactions with external dependencies.





3. HOW to use ADAPTER ?

To use the Adapter pattern, you can follow these steps:

1. **Identify the interfaces:** Determine the incompatible interfaces that need to be adapted. This includes the target interface that your client code expects and the existing interface that needs to be adapted.
2. **Create the target interface:** Define the interface that your client code expects. This is the interface that the adapter will implement.
3. **Create the adapter class:** Create a new class that implements the target interface. This adapter class will act as a bridge between the existing class and the client code. The adapter class should contain an instance of the existing class.

3. HOW to use ADAPTER ?

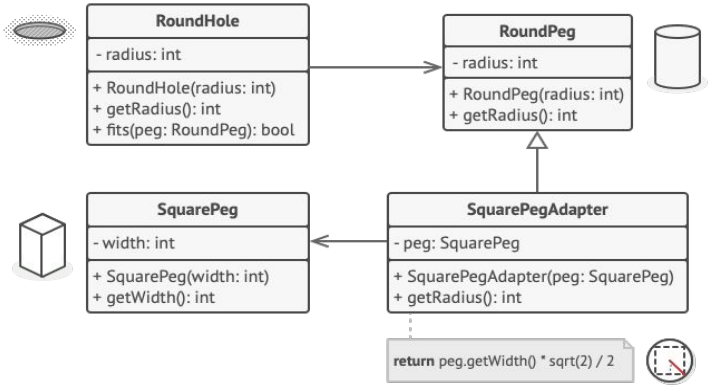
To use the Adapter pattern, you can follow these steps:

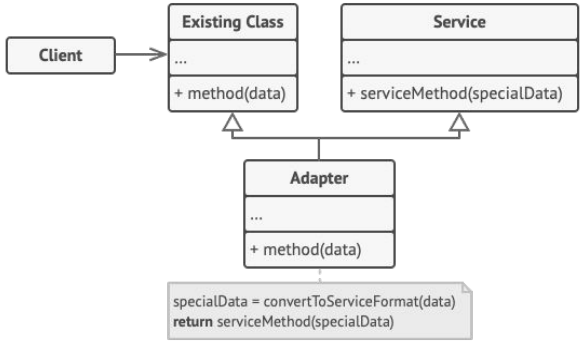
4. **Implement the adapter methods:** In the adapter class, implement the methods defined by the target interface. These methods should delegate the calls to the corresponding methods in the existing class, adapting the input and output as necessary.
5. **Connect the client code:** Replace the direct usage of the existing class in the client code with the adapter class. The client code will now interact with the adapter class through the target interface.

3. WHERE to use ADAPTER ?

The Adapter pattern is used when we want to convert the interface of an existing class into another interface that the client expects:

1. Integrating with third-party libraries
2. Reusing legacy code
3. Supporting different versions or implementations
4. Mocking and testing





5. SOURCE code

1. Adapter Design Pattern (Github)
2. Adapter Design Pattern Computer Types (refactoring.guru)

Summary

Overall, the Adapter pattern is handy in situations where you need to bridge the gap between incompatible interfaces, integrate external code, or decouple client code from specific implementations. It promotes flexibility, reusability, and modularity in your application design.

Reference

1. Dive into Design Pattern (page 150 - 162)
2. Website (www.refactoring.guru)
3. Adapter Pattern in TypeScript
4. Adapter Design Pattern in detail (YouTube)

Thank you!

Presented by **Sanjar Suyunov**

(sanjarsuyunov1304@gmail.com)