# Object-Oriented Software Analysis and Design

School of Computer Science
University of Windsor

## Gang of Four's Pattern Catalog

| Creational | Structural | Behavioral |
|---|---|---|
| Abstract Factory | Adapter | Chain of Responsibility |
| Builder | Bridge | Command |
| Factory Method | Composite | Interpreter |
| Prototype | Decorator | Iterator |
| Singleton | Facade | Mediator |
|  | Flyweight | Memento |
|  | Proxy | Observer |
|  |  | State |
|  |  | Strategy |
|  |  | Template Method |
|  |  | Visitor |

## Adapter Pattern: Intent

- Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.
- Also known as
  - Wrapper

## Adapter Pattern: Applicability

- Use the Adapter pattern when
  - you want to use an existing class, and its interface does not match the one you need.
  - you want to create a reusable class that cooperates with unrelated or unforeseen classes, that is, classes that don't necessarily have compatible interfaces.
  - (object adapter only) you need to use several existing subclasses, but it's impractical to adapt their interface by subclassing every one. An object adapter can adapt the interface of its parent class.

## Adapter Pattern: Motivation

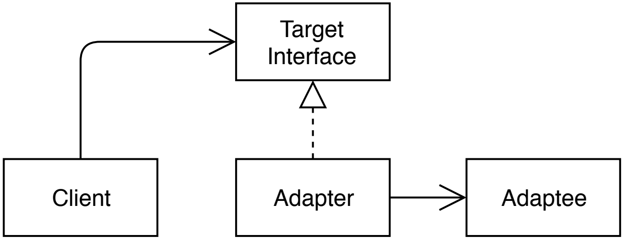
- Sometimes a toolkit class that's designed for reuse isn't reusable only because its interface doesn't match the domain-specific interface an application requires.

## Adapter Pattern

- Physically, an adapter is a device that is used to connect pieces of equipment that cannot be connected directly.
- Software systems may also face similar issues: not all systems have compatible software interfaces. In other words, the output of one system may not conform to the expected input of another system.
- This frequently happens when a pre-existing system needs to incorporate third-party libraries or needs to connect to other systems.

## Adapter Pattern (contd.)

- Translated into a diagram, the pattern looks as below:



## Adapter Pattern (contd.)

- The adapter design pattern consists of several parts.
    - ***A client class.***
        - This class is the part of your system that wants to use a third-party library or external systems.
    - ***An adaptee class.***
        - This class is the third-party library or external system that is to be used.
    - ***An adapter class.***
        - This class sits between the client and the adaptee. The adapter conforms to what the client is expecting to see, by implementing a target interface. The adapter also translates the client request into a message that the adaptee will understand, and returns the translated request to the adaptee. The adapter is a kind of wrapper class.
    - ***A target interface.***
        - This is used by the client to send a request to the adapter.

## Adapter Pattern (contd.)

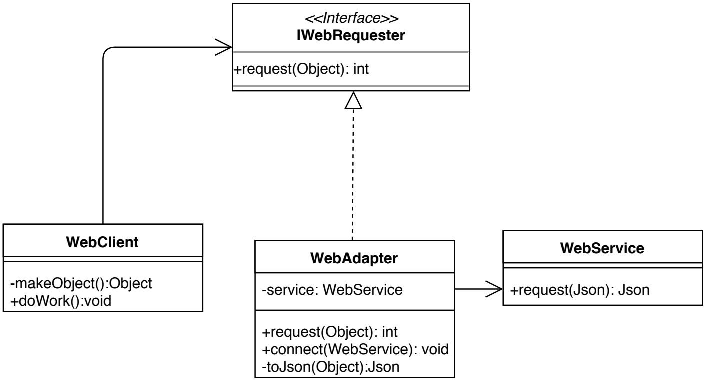
- Implementation of an adapter design pattern can be broken down into steps.

    1. Design the target interface
    2. Implement the target interface with the adapter class
    3. Send the request from the client to the adapter using the target interface

- Let us examine each of these steps using a specific example.

## Adapter Pattern: Example

- Imagine an example where there is a pre-existing web client that we would like to interact with another web service. However, the service only supports JSON objects, and an adapter is needed to convert our Object request into a JSON object.



## Adapter Pattern: Example

- Step 1: Design the target interface

```
//IWebRequester
public interface IWebRequester {
    public int request(Object);
}
```

## Adapter Pattern: Example (contd.)

- Step 2: Implement the target interface with the adapter class

```
//WebAdapter
public class WebAdapter implements IWebRequester
{
    private WebService service;
    public void connect(WebService currentService)
    {
        this.service = currentService;
        /* Connect to the web service */
    }
    public int request(Object request)
    {
        Json result = this.toJson(request);
        Json response = service.request(result);
        if (response != null)
            return 200; // OK status code
        return 500; // Server error status code
    }
    private Json toJson(Object input) {...}
}
```

## Adapter Pattern: Example (contd.)

- Step 3: Send the request from the client to the adapter using the target interface

```
//WebClient
public class WebClient {
    private IWebRequester webRequester;
    public WebClient(IWebRequester webRequester) {
        this.webRequester = webRequester;
    }
    private Object makeObject() { ... } // Make an Object
    public void doWork() {
        Object object = makeObject();
        int status = webRequester.request(object);
        if (status == 200) {
            System.out.println("OK");
        }
        else {
            System.out.println("Not OK");
        }
    return;
    }
}
```

## Adapter Pattern: Example (contd.)

```java
//Program (main)
public class Program
{
    public static void main(String args[])
    {
        String webHost = "Host: https://google.com\n\r";
        WebService service = new WebService(webHost);
        WebAdapter adapter = new WebAdapter();
        adapter.connect(service);
        WebClient client = new WebClient(adapter);
        client.doWork();
    }
}
```

## Adapter Pattern: Example (contd.)

- In the main program, the Web Adapter, the Web Service, and the Web Client needs to be instantiated.
- The Web Client deals with the adapter through the Web Requester interface to send a request.
- The Web Client should not need to know anything about the Web Service, such as its need for JSON objects. The adaptee is hidden from the client by the wrapping adapter class.

## Adapter Pattern (contd.)

- Although it might be tempting to think that a solution is to simply change an interface so that it is compatible with another one, this is not always feasible, especially if the other interface is from a third-party library or external system.
- Changing your system to match the other system is not always a solution either, because an update by the vendors to the outside systems may break part of our system.

## Adapter Pattern (contd.)

- Wrap the adaptee and exposes a target interface to the client.

- Indirectly change the adaptee's interface into one that the client is expecting by implementing a target interface.

- Indirectly translate the client's request into one that the adaptee is expecting.

- Reuse an existing adaptee with an incompatible interface.