Adapter Pattern

Problem – Incompatible Interfaces

- Sometimes in your application, you want to convert the interface of a class into another interface that your clients expect to have because their interfaces does not match.
- Like Port Adapters:

USB-C to USB

Or

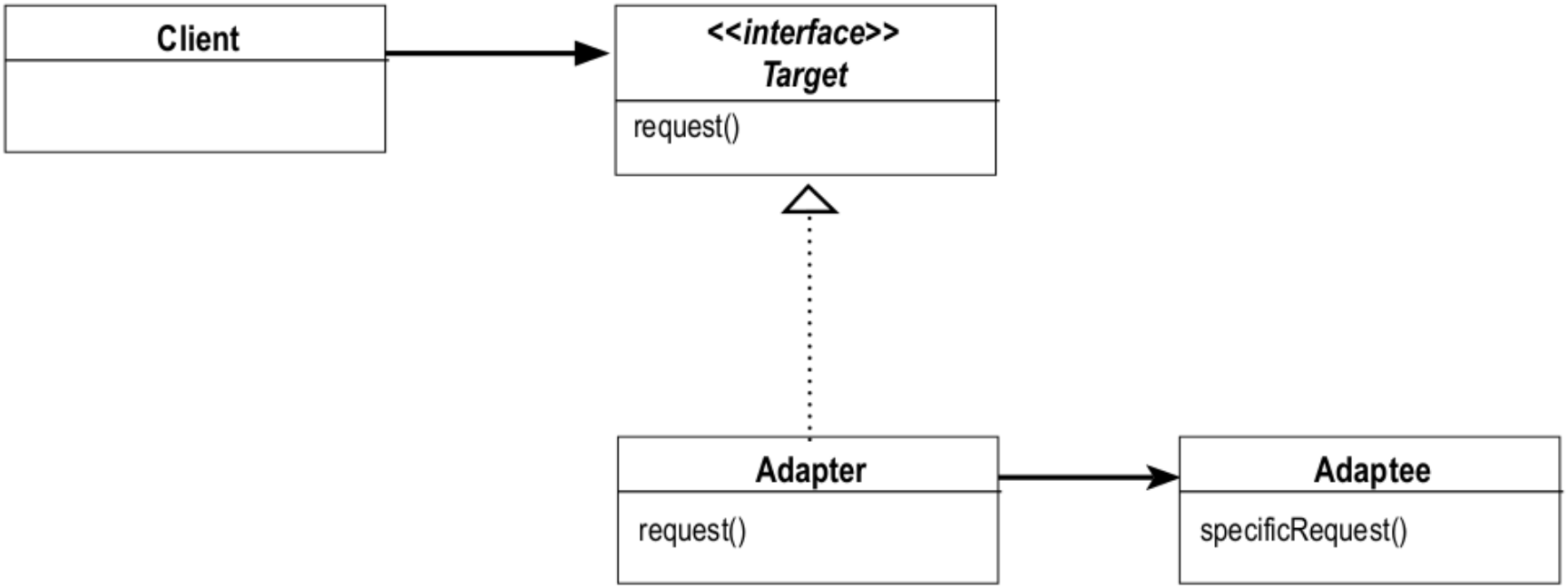
DVI to HDMI

Definition of Adapter Pattern

- The Adapter Pattern converts the interface of a class into another interface that the clients expect to have.

  Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.

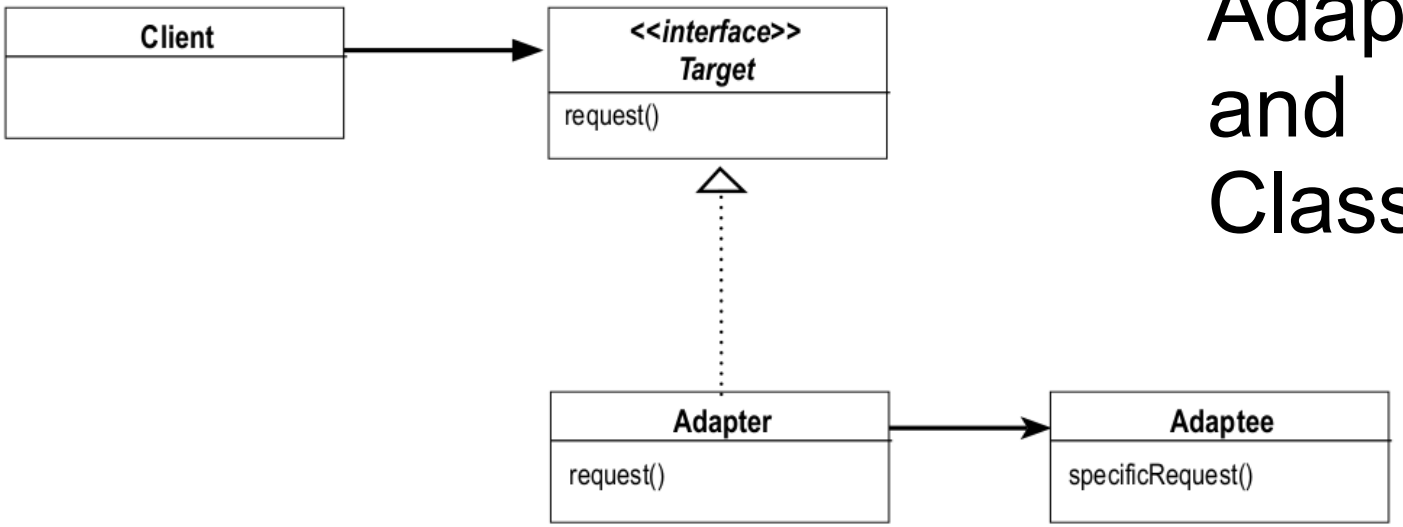
Object Adapter



Simple Adapter

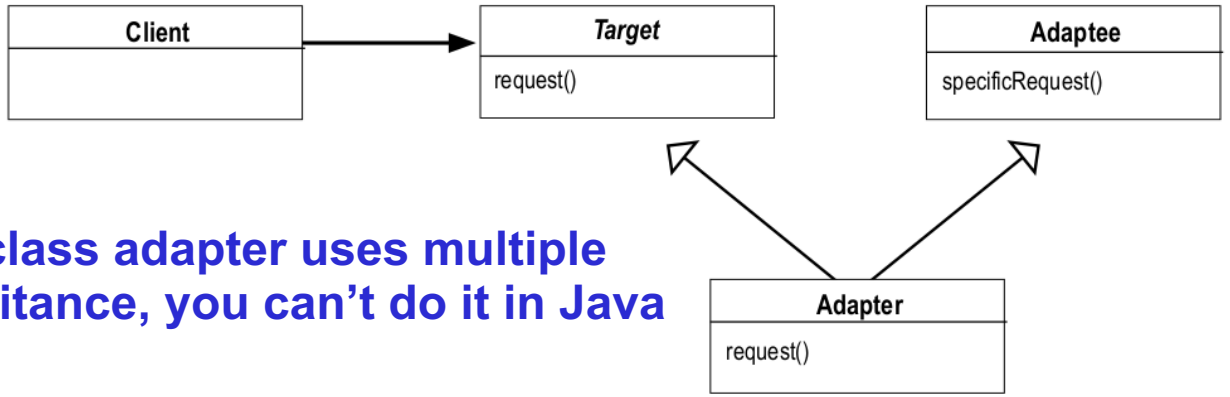
```
public class AdapteeToClientAdapter implements Adapter {

private final Adaptee instance;

public AdapteeToClientAdapter(final Adaptee instance) {
        this.instance = instance;
}

@Override
public void request() {

// here you would call then the  Adaptee's method(s)
// specificRequest
instance.specificRequest();

 }
}
```

Object Adapter



Adapter Object and Class Adapter

Class Adapter



The class adapter uses multiple inheritance, you can't do it in Java

In Class Adapter.

You are committed to one specific adaptee class, but you have a huge advantage because you don't have to reimplement my entire adaptee.

You can also override the behavior of my adaptee if you need to because you are just doing subclassing.

When use the Adapter Pattern

- Use Adapter pattern when
  - **Adapting interfaces.** Existing class and its interface does not match the one you need.
  - **Reusable in Future.** 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.

Consequences of using Adapter Pattern

- Class and object adapters have different trade-offs.

  An object adapter

  - lets **a single Adapter work with many Adaptees.** **T**he Adaptee itself and all of its subclasses. The Adapter can also add functionality to all Adaptees at once.
  - **makes it harder to override Adaptee behavior.** It will require subclassing Adaptee and making Adapter refer to the subclass rather than the Adaptee itself.

Consequences of using Adapter Pattern

A class adapter

- Adapter pattern adapts Adaptee to a concrete Adapter class. As a consequence, a class adapter won't work when we want to adapt a class and all its subclasses.
- lets **Adapter override some of Adaptee's behavior,** since Adapter is a subclass of Adaptee.
- **introduces only one object**, and no additional pointer indirection is needed to get to the adaptee.

Summary of Adapter Pattern

- Adapter Pattern converts the interface of a class into another interface the clients expect.
- Adapter pattern is one of the **Structural Patterns**
- **Related Patterns:**
  - **Decorator** - enhances another object without changing its interface. A decorator is thus more transparent to the application than an adapter is. As a consequence, Decorator supports recursive composition, which isn't possible with pure adapters.
  - **Bridge**
  - **Proxy**