CMPE 202

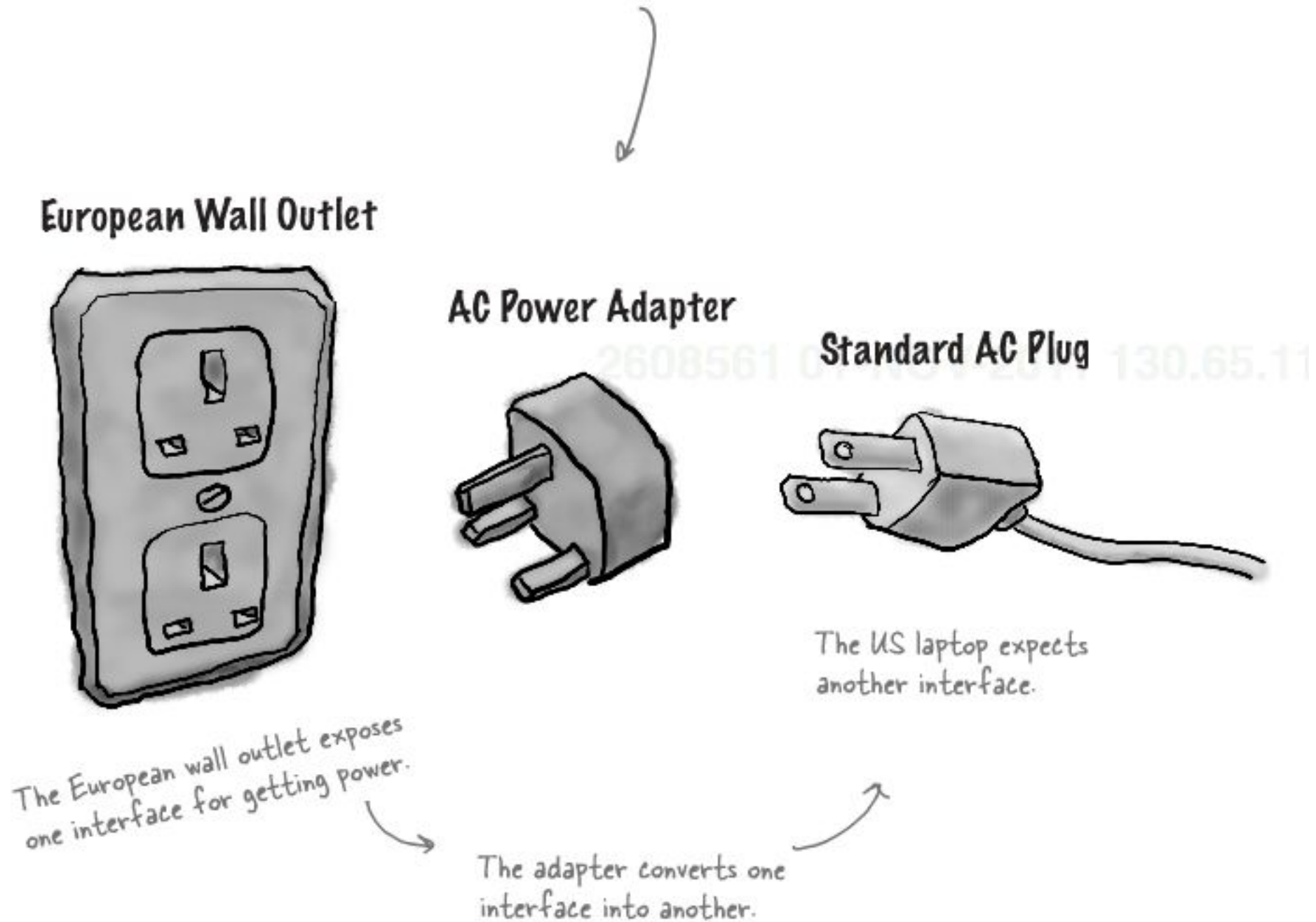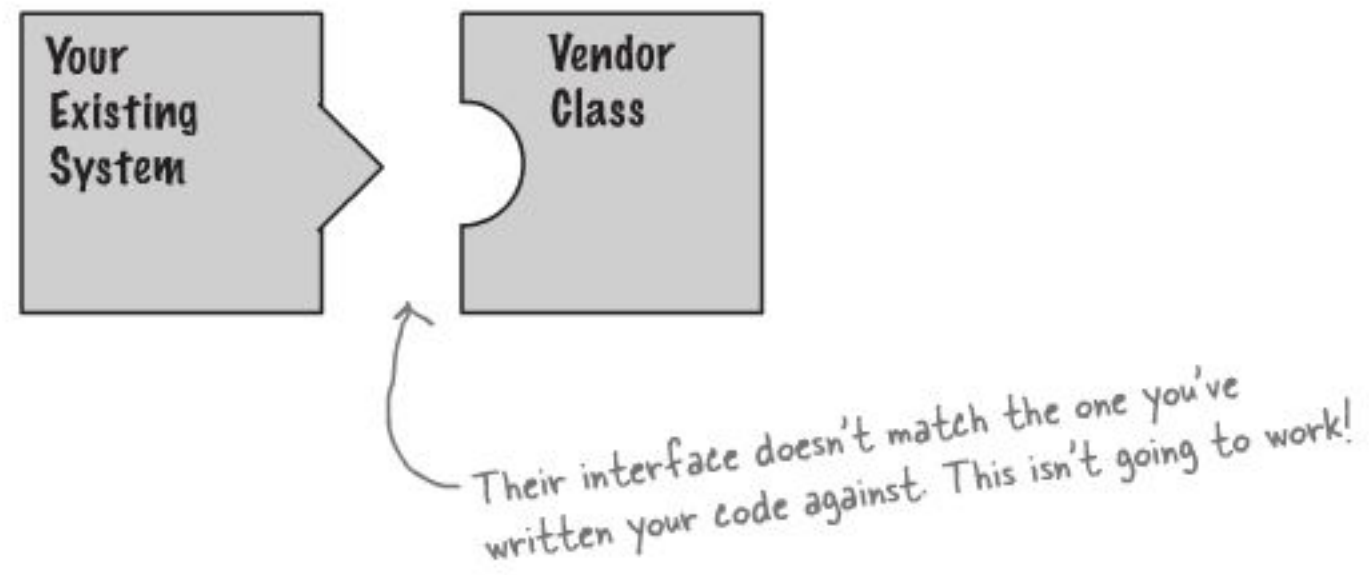## Gang of Four Design Patterns

Adapter

Adapters all around us

You'll have no trouble understanding what an OO adapter is because the real world is full of them. How's this for an example: Have you ever needed to use a US-made laptop in a European country? Then you've probably needed an AC power adapter...

European Wall Outlet

AC Power Adapter

Standard AC Plug

The European wall outlet exposes one interface for getting power.

The adapter converts one interface into another.

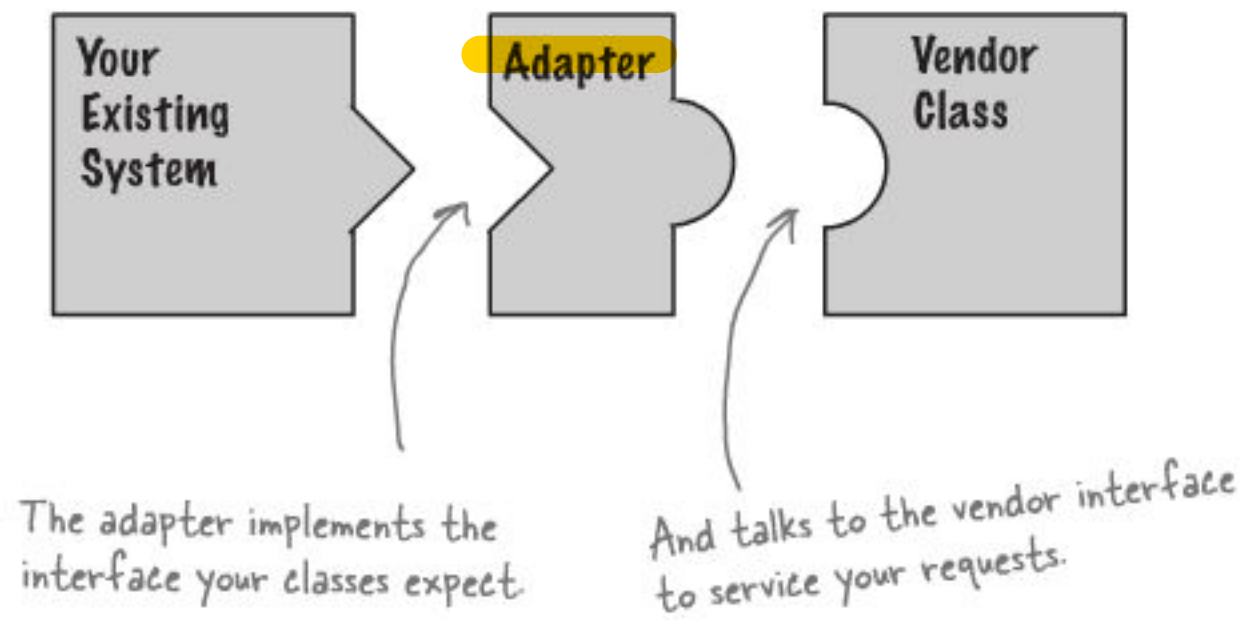The US laptop expects another interface.

Object oriented adapters

Say you've got an existing software system that you need to work a new vendor class library into, but the new vendor designed their interfaces differently than the last vendor:

Your Existing System → Vendor Class

Their interface doesn't match the one you've written your code against. This isn't going to work!

Okay, you don't want to solve the problem by changing your existing code (and you can't change the vendor's code). So what do you do? Well, you can write a class that adapts the new vendor interface into the one you're expecting.

Your Existing System → **Adapter** → Vendor Class

The adapter implements the interface your classes expect.

And talks to the vendor interface to service your requests.

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

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.

Participants
- **Target** (Interface)
  - defines the domain-specific interface that Client uses.

- **Client**
  - collaborates with objects conforming to the Target interface.

- **Adaptee** (Interface)
  - defines an existing interface that needs adapting.

- **Adapter**
  - adapts the interface of Adaptee to the Target interface.

Collaborations
- Clients call operations on an Adapter instance. In turn, the adapter calls Adaptee operations that carry out the request.
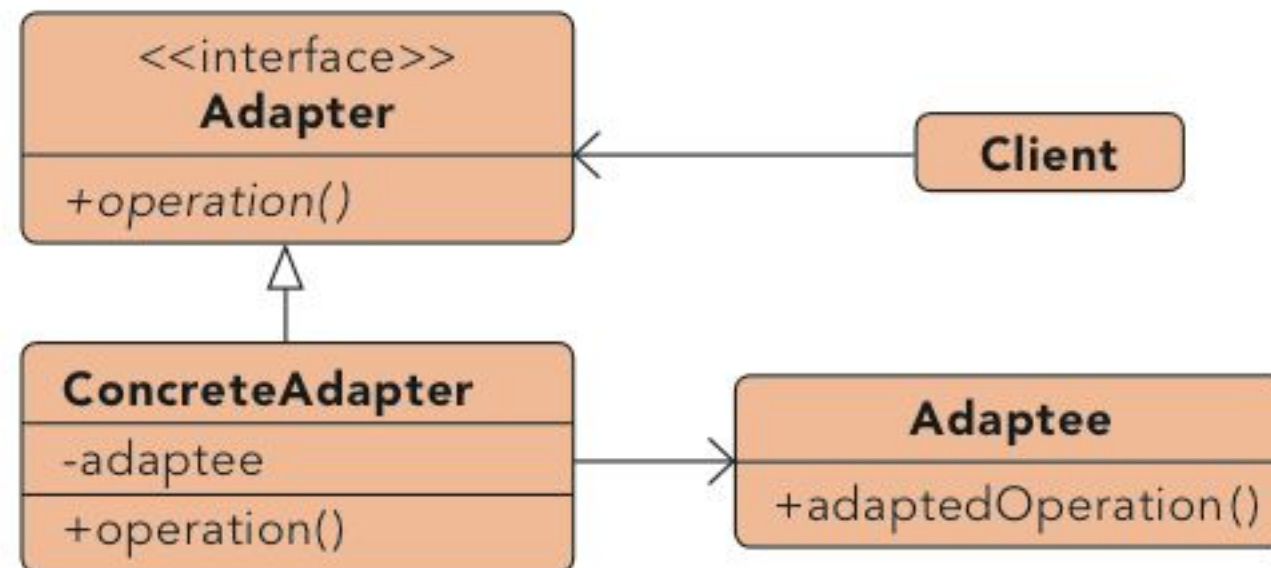
Purpose

Permits classes with disparate interfaces to work together by creating a common object by which they may communicate and interact.

Use When

- A class to be used doesn't meet interface requirements.
- Complex conditions tie object behavior to its state.
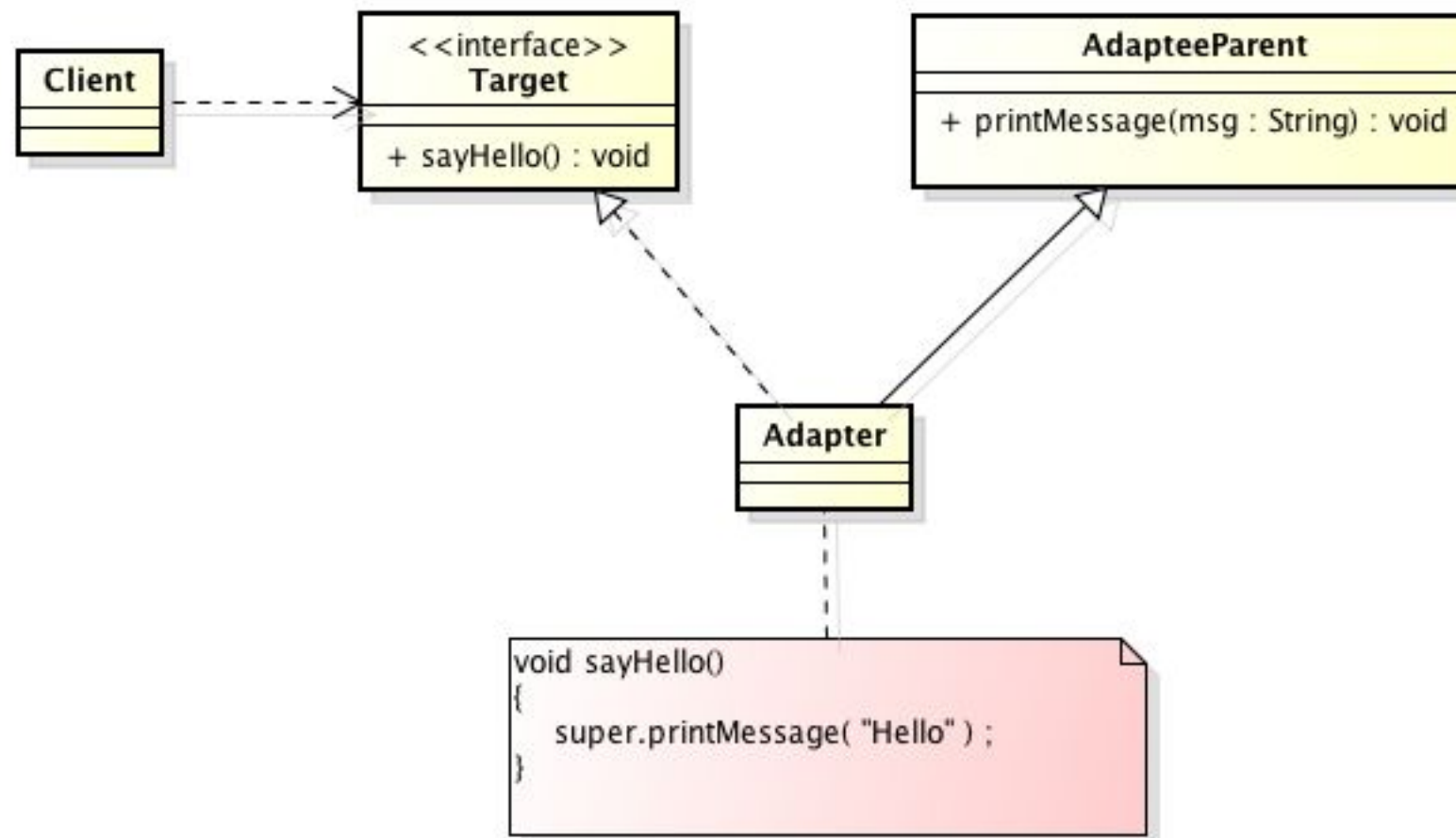- Transitions between states need to be explicit.

```java
public class Client {

    public void runTest()
    {
        Target obj = new Adapter() ;
        obj.sayHello() ;
    }

}
```

```java
public class AdapteeParent {

    public void printMessage(String msg) {
        System.out.println( msg );
    }

}
```



```java
public class Adapter extends AdapteeParent implements Target {

    /**
     * @see adapter.classAdapter.Target#sayHello()
     */
    public void sayHello() {
        super.printMessage( "Hello" );
    }

}
```

```java
public class Client {

    public void runTest()
    {
        Target obj = new TargetObject() ;
        obj.sayHello() ;
    }

}
```

```java
public class TargetObject implements Target {

    private AdapteeObject adaptee;

    public TargetObject()
    {
        adaptee = new AdapteeObject() ;
    }

    /**
     * @see adapter.objectAdapter.Target#sayHello()
     */
    public void sayHello() {
        adaptee.printMessage("Hello");
    }

}
```
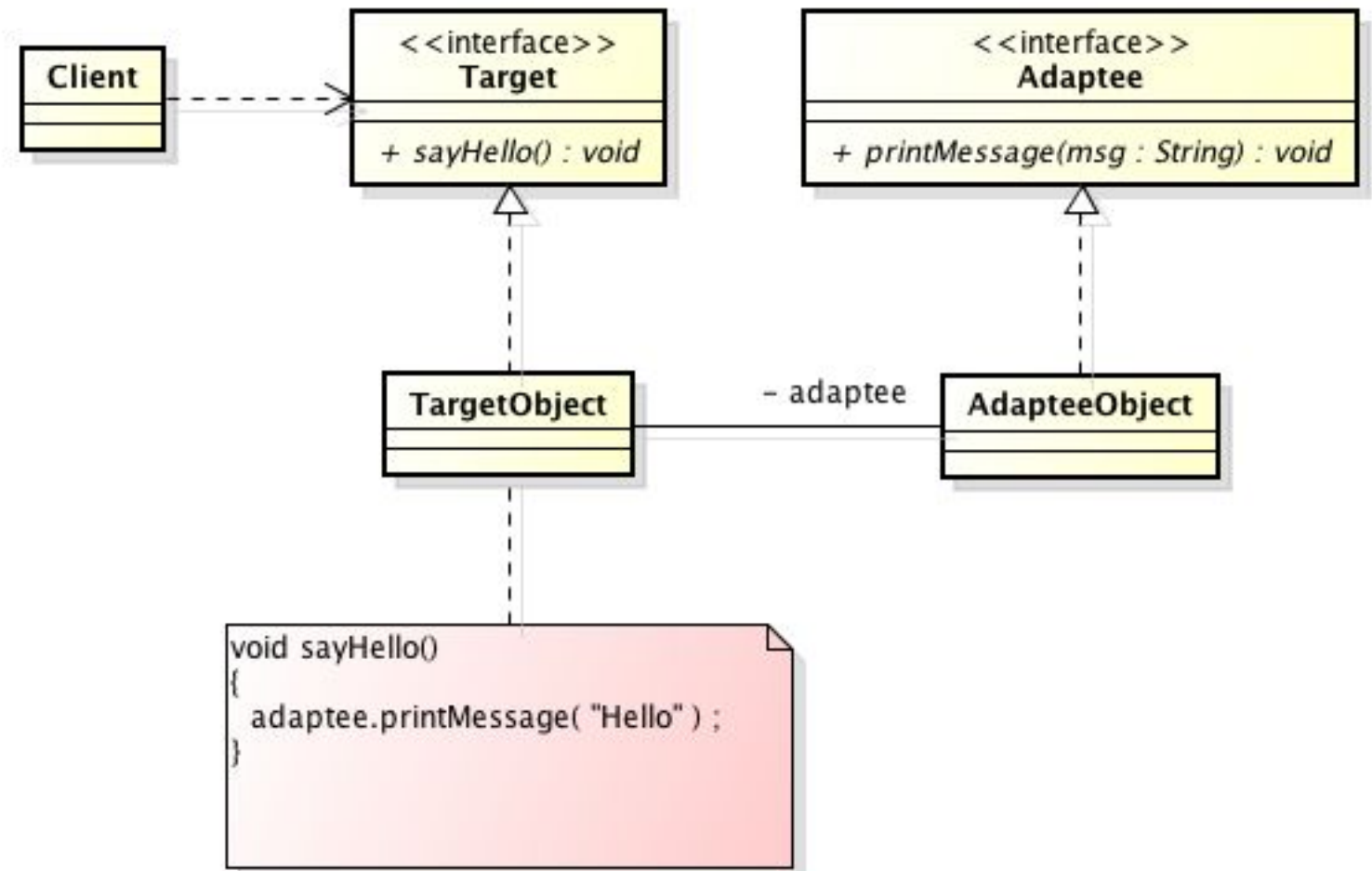
```
Client ----> <<interface>>
                 Target
             + sayHello() : void
```

```
<<interface>>
    Adaptee
+ printMessage(msg : String) : void
```

```
TargetObject ── – adaptee ── AdapteeObject
```

```
void sayHello()
{
  adaptee.printMessage( "Hello" ) ;
}
```

```java
public class AdapteeObject implements Adaptee {

    private TargetObject targetObject;

    /**
     * @see adapter.objectAdapter.Adaptee#printMessage(java.lang.String)
     */
    public void printMessage(String msg) {
        System.out.println( msg ) ;
    }

}
```