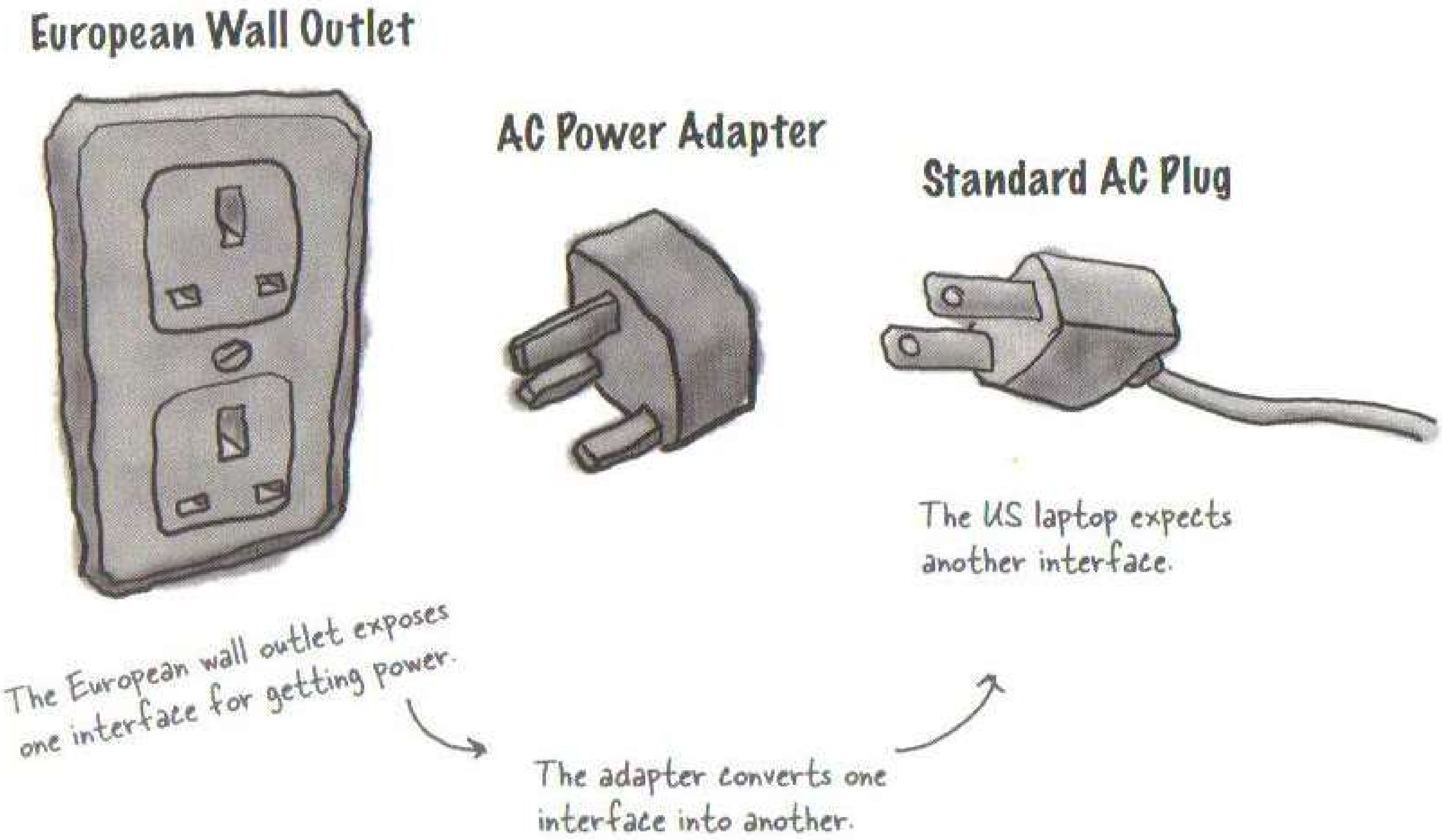
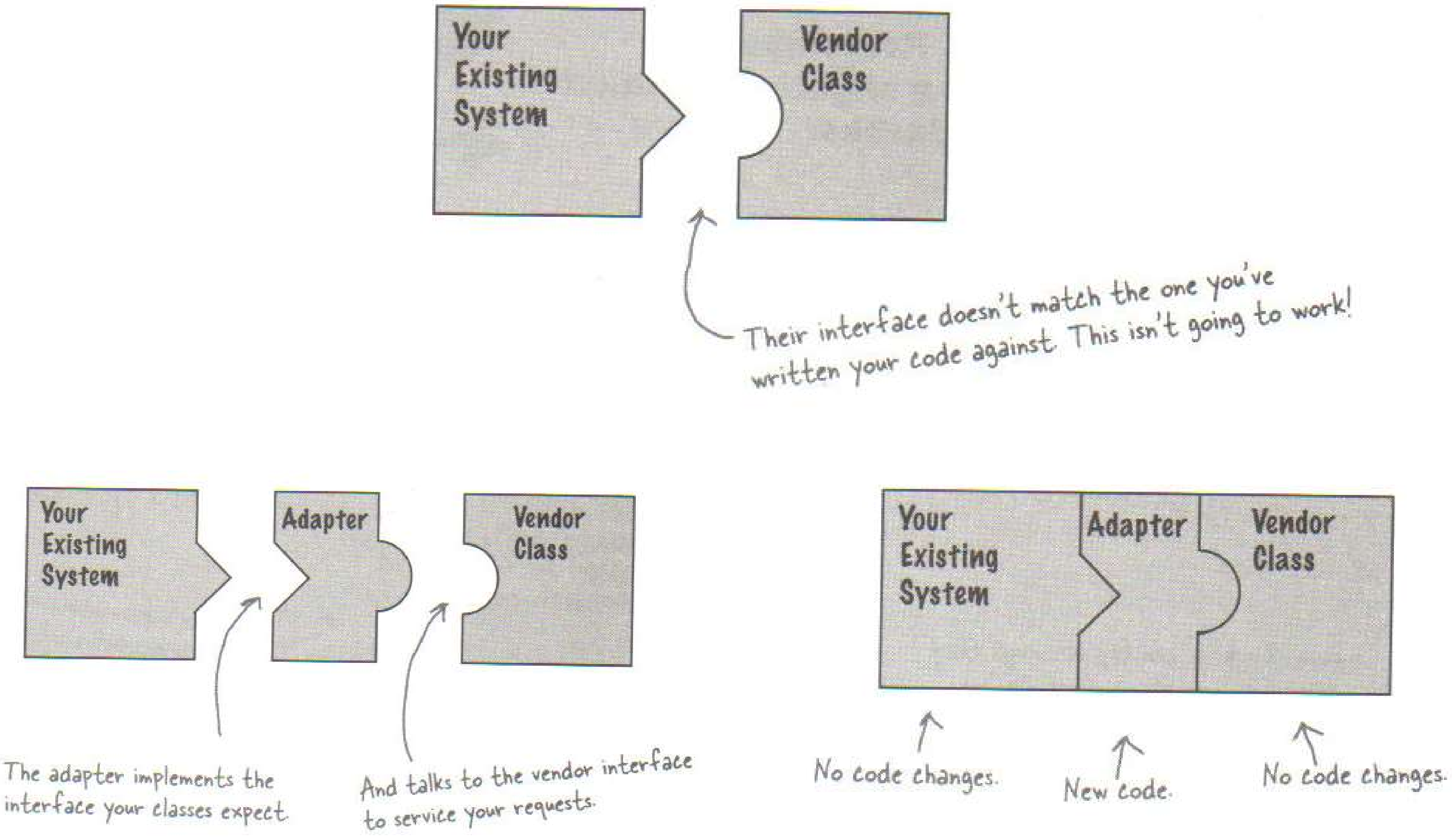
Design Patterns: Adapter

Adapters in real life


European Wall Outlet
AC Power Adapter
Standard AC Plug
The US laptop expects another interface.
The European wall outlet exposes one interface for getting power.
The adapter converts one interface into another.

Object-Oriented Adapters



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

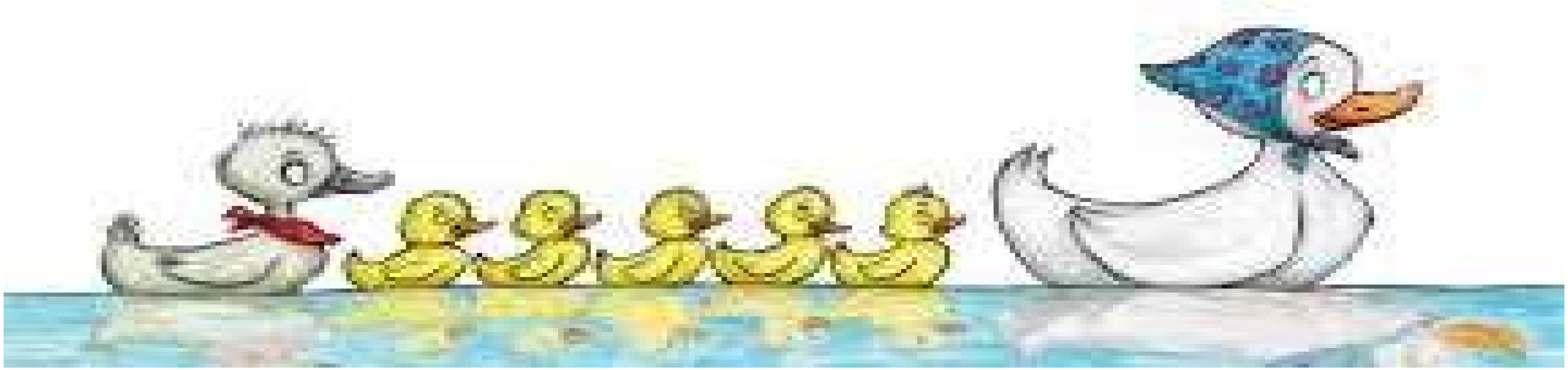
The adapter implements the interface your classes expect.

And talks to the vendor interface to service your requests.

No code changes.

New code.

No code changes.



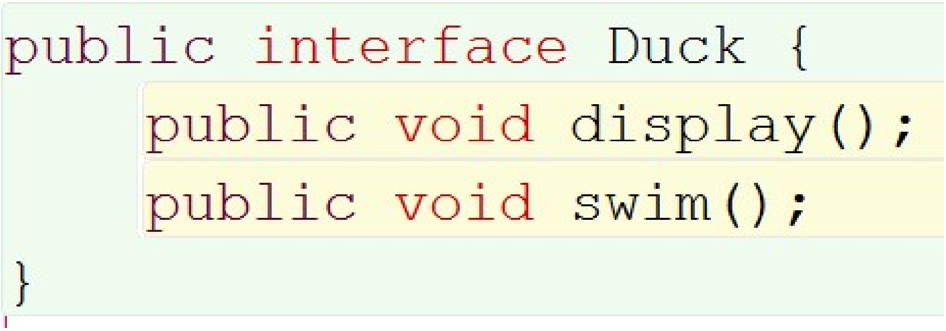
```
public interface Duck {
            public void display();
            public void swim(); }
```

```
public interface Swan{
    public void show();
    public void swim(); }
```

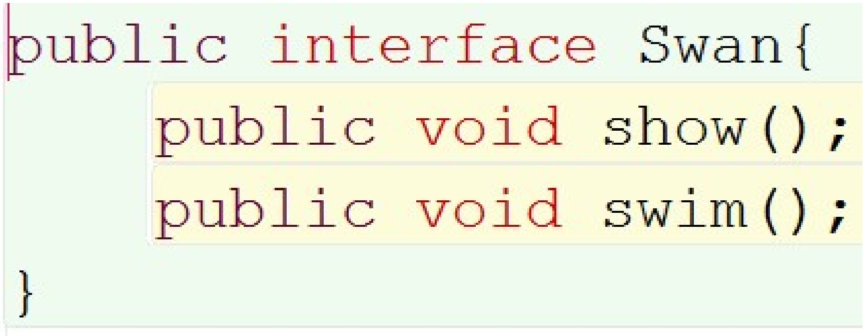
```
public class Duckling implements Duck {
    public void display() {
      System.out.println(“I’m a pretty duckling");
    }
    public void swim() {
              System.out.println("I'm learning…");
    }
}
```

```
public class HuglyDuckling implements Swan{
    public void show() {
       System.out.println(“I’m large and hugly");
    }
    public void swim() {
        System.out.println("I'm swimming!”");
    }
}
```

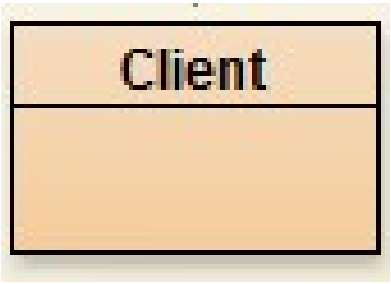
Two hierarchies and the need to deal with objects in a uniform way

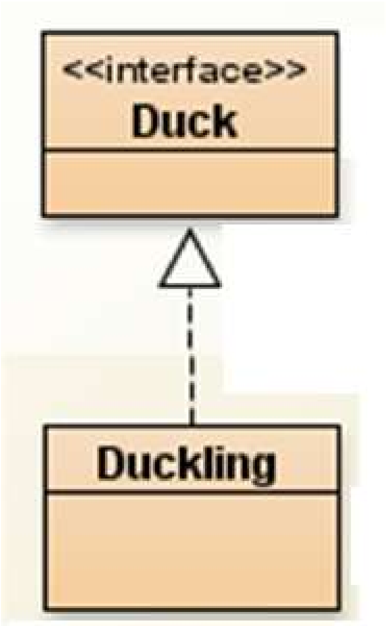


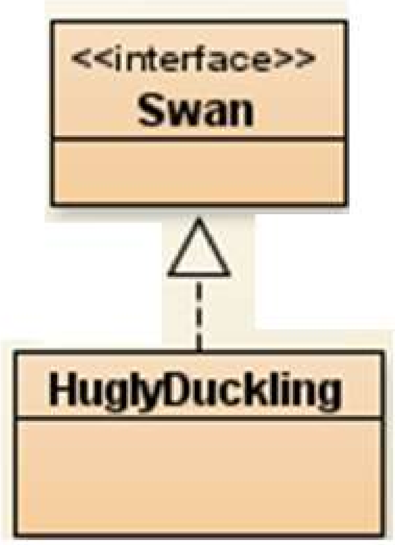
```
public interface Duck {
    public void display();
    public void swim();
}
```



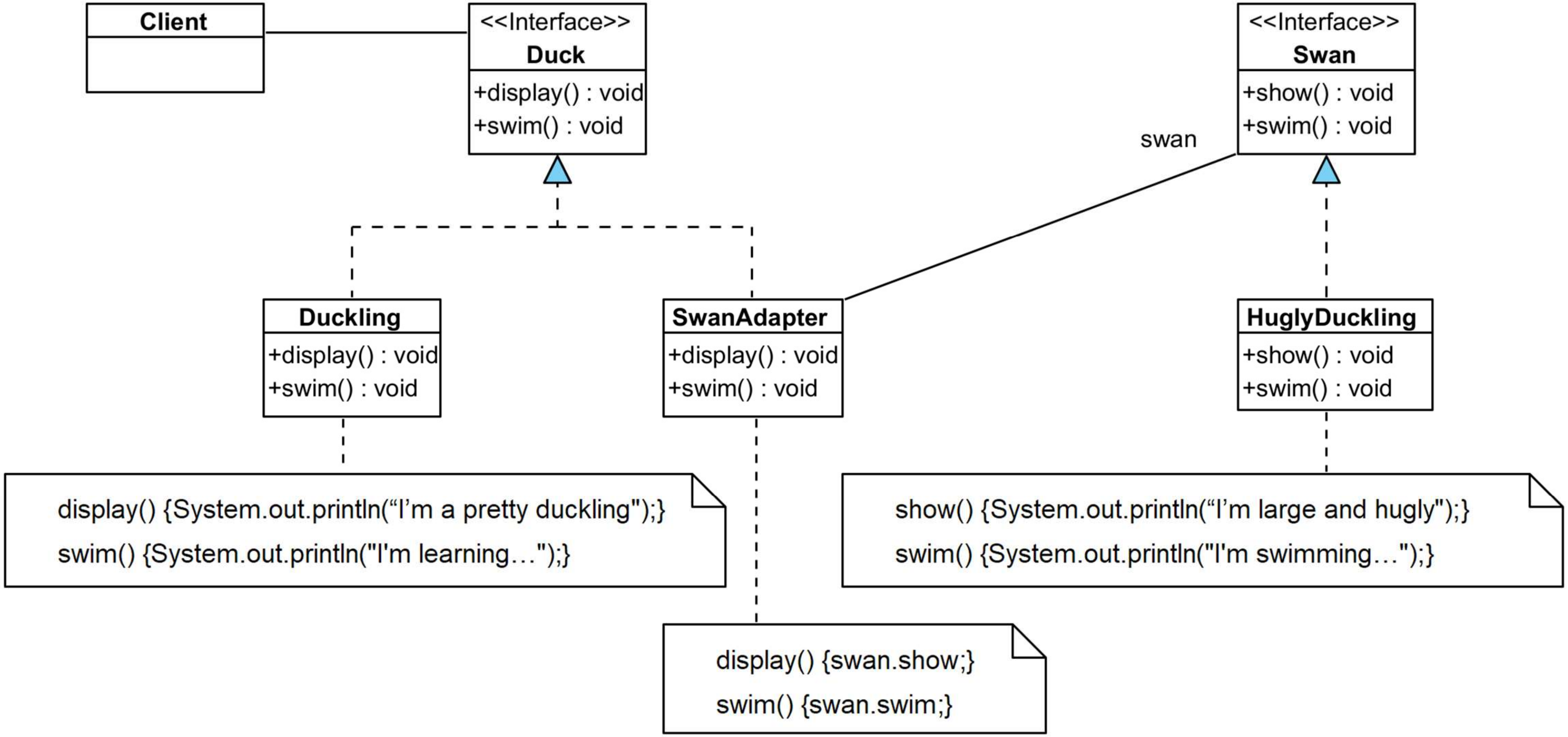
```
public interface Swan{
    public void show();
    public void swim();
}
```







Hugly Duckling with Adapter

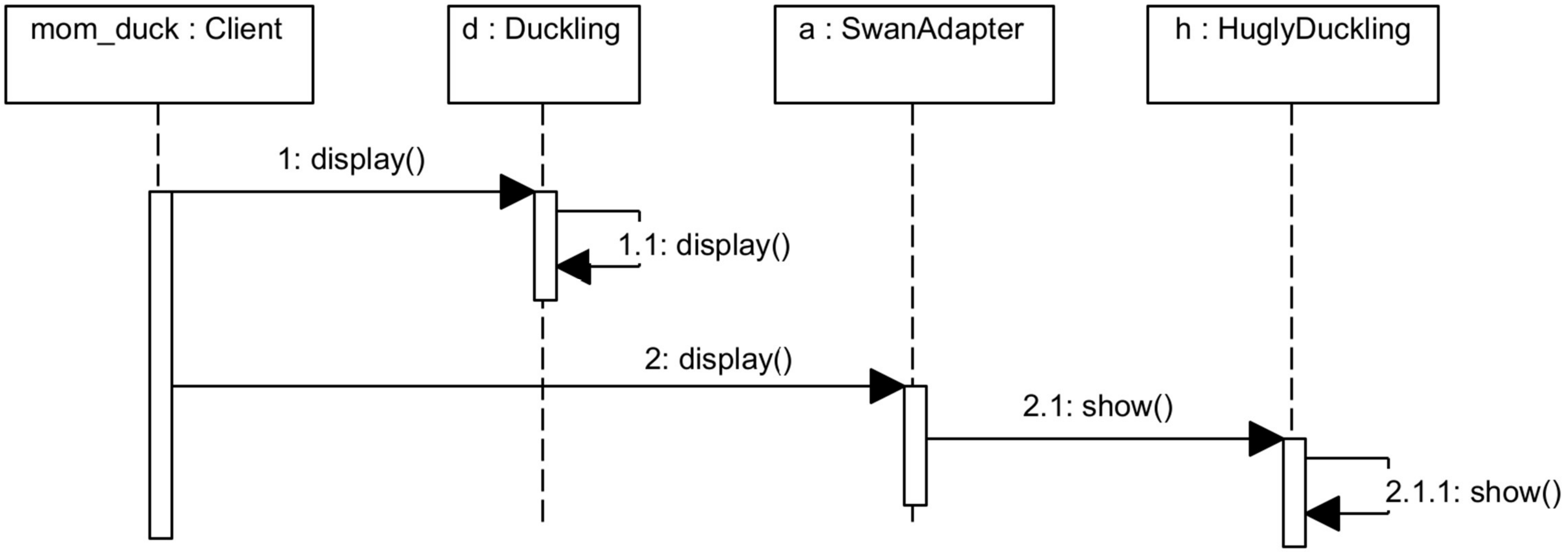


```
display() {System.out.println(“I’m a pretty duckling");;}
swim() {System.out.println("I'm learning…");}
```

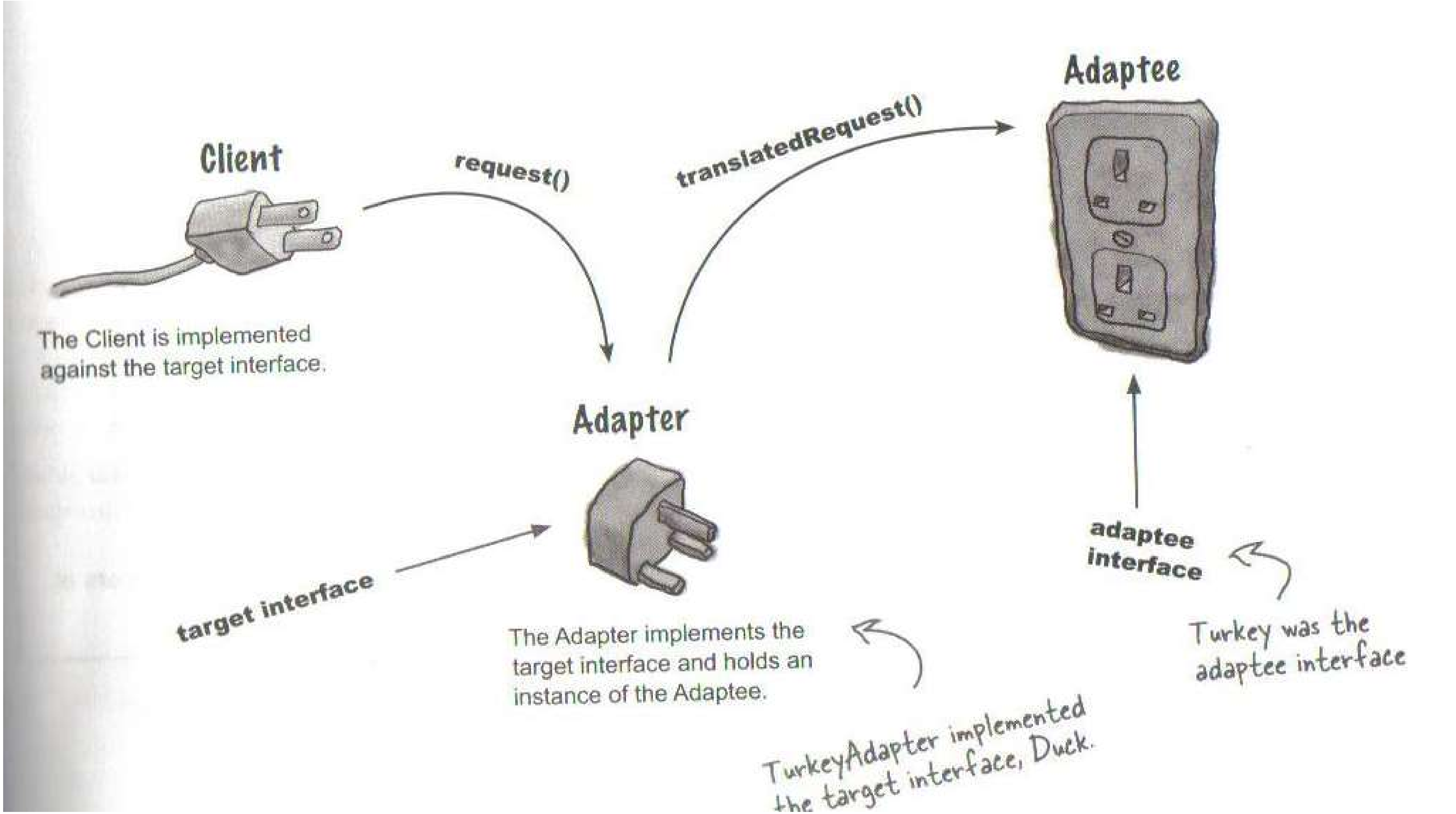
```
show() {System.out.println(“I’m large and hugly");}
swim() {System.out.println("I'm swimming…");}
```

```
display() {swan.show;}
swim() {swan.swim;}
```

Hugly Duckling with Adapter



Adapter Pattern explained



The Client is implemented against the target interface.

The Adapter implements the target interface and holds an instance of the Adaptee.

Turkey was the adaptee interface

TurkeyAdapter implemented the target interface, Duck.

Adapter Pattern defined

The Adapter Pattern converts the interface of a class into another interface the clients expect. Adapter lets classes work together that couldn’t otherwise because of incompatible interfaces.

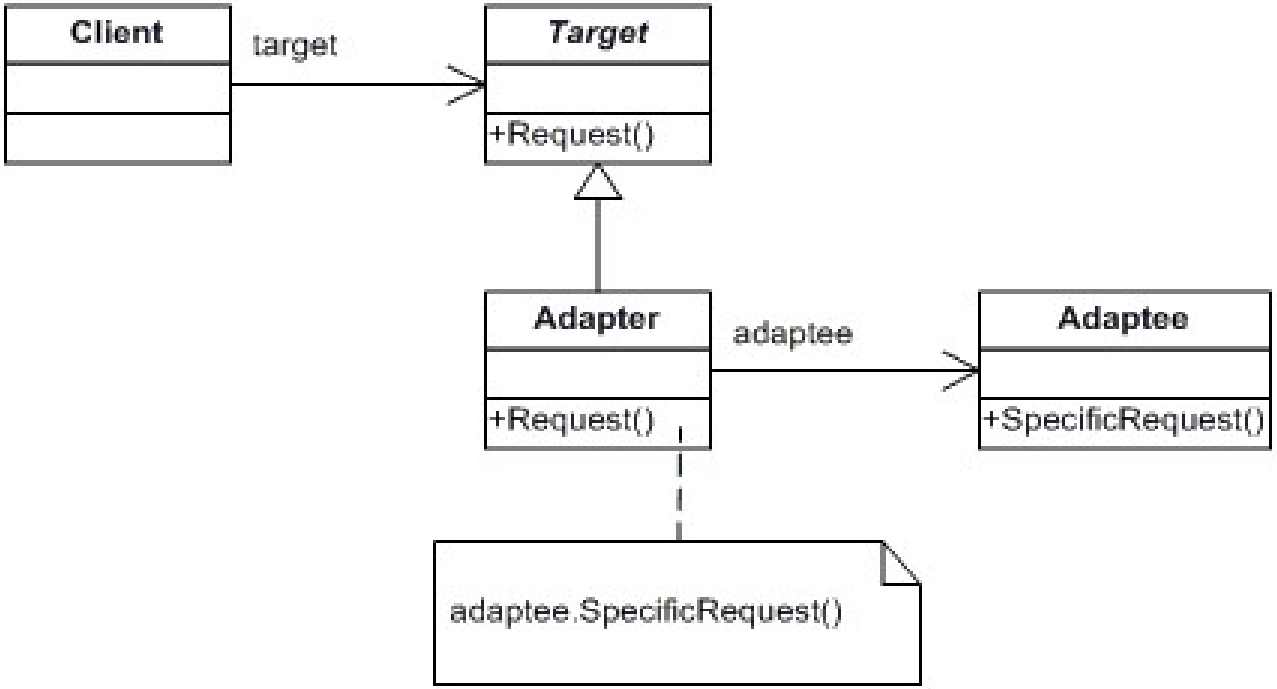
Adapter pattern

Delegation is used to bind an Adapter to an Adaptee

Interface inheritance is used to specify the interface of the Adapter class.

Target and Adaptee (usually called legacy system) pre-exist the Adapter.

Target may be realized as an interface in Java.



Participants

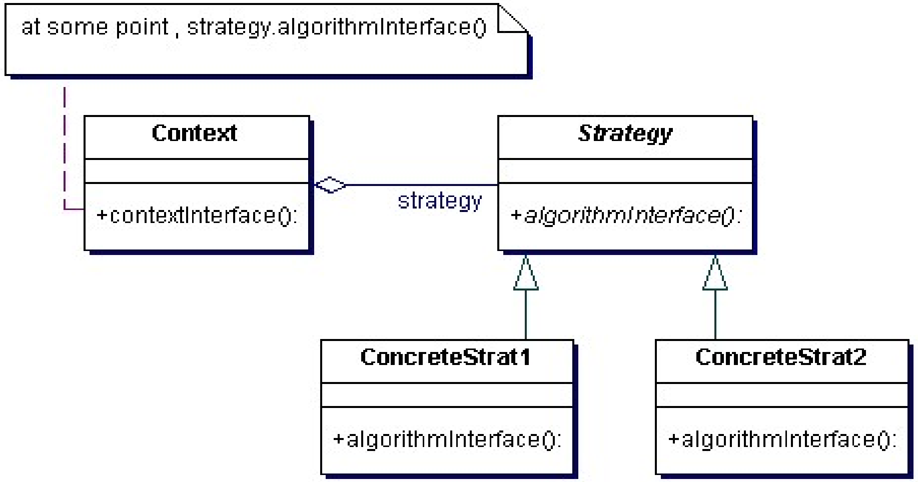
Target: Defines the application-specific interface that clients use.

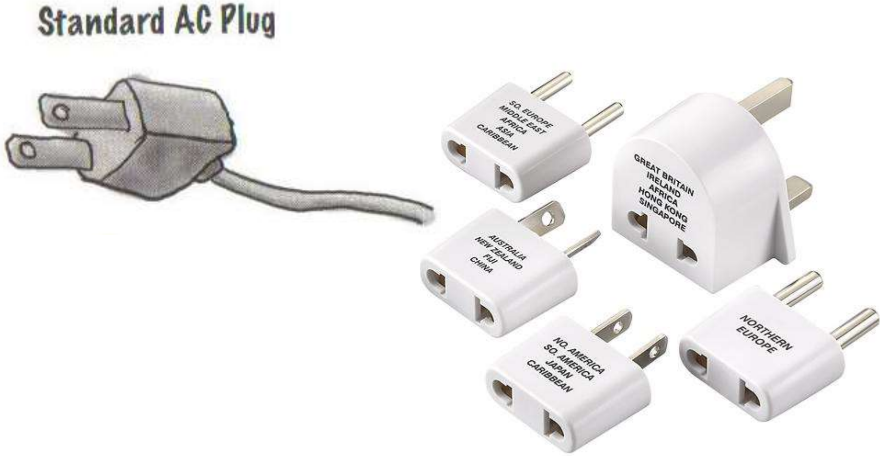
Client: Collaborates with objects conforming to the target interface.

Adaptee: Defines an existing interface that needs adapting.

Adapter: Adapts the interface of the adaptee to the target interface.

Adapter Pattern and Strategy Pattern





The adapter can play the role of a concrete strategy: if we have several modules implementing the same functionality and we wrote adapters for them, we have a set of adapters that implement the same interface.

We can hence replace the adapters objects at run time.

Homework: DPHomework4

Extend the ugly duckling example to adapt turkeys too.

Desing a solution that combines Adapter and Strategy

Façade

Another GoF pattern

Forwards requests to many adaptees