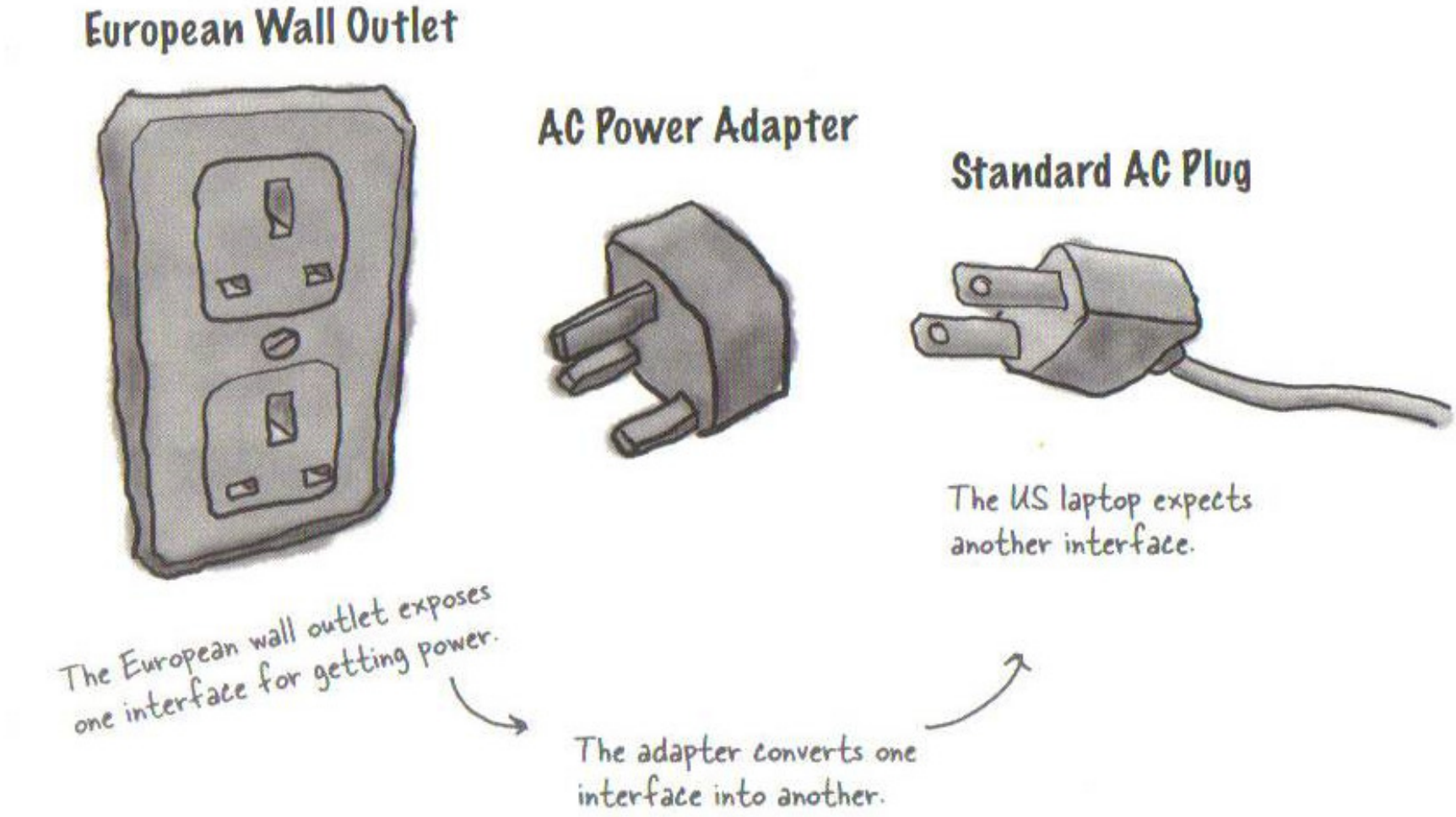
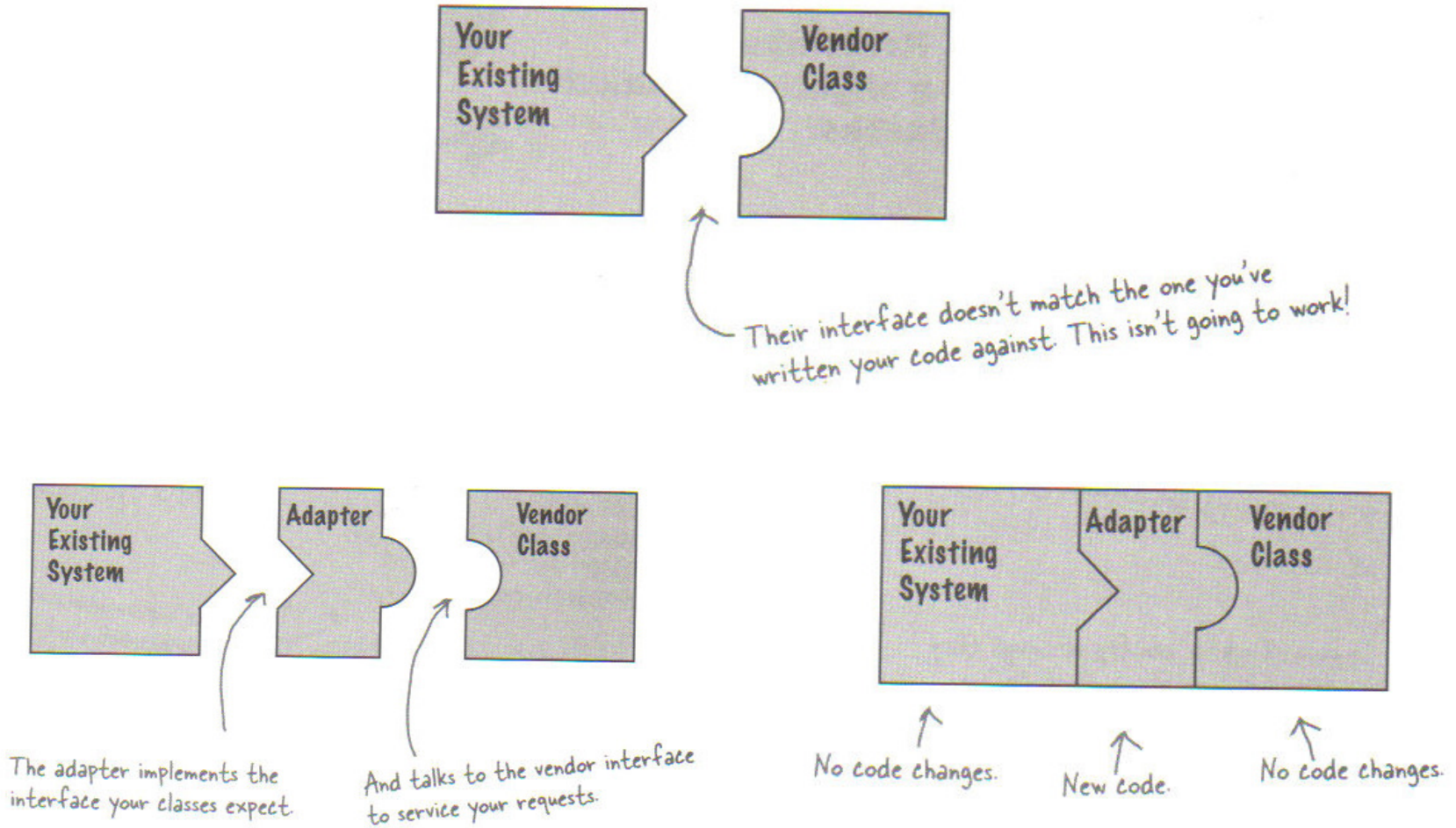
# Tecniche di Progettazione: Design Patterns

GoF: Adapter

# Adapters in real life (anglo-centric….)



# Object-Oriented Adapters

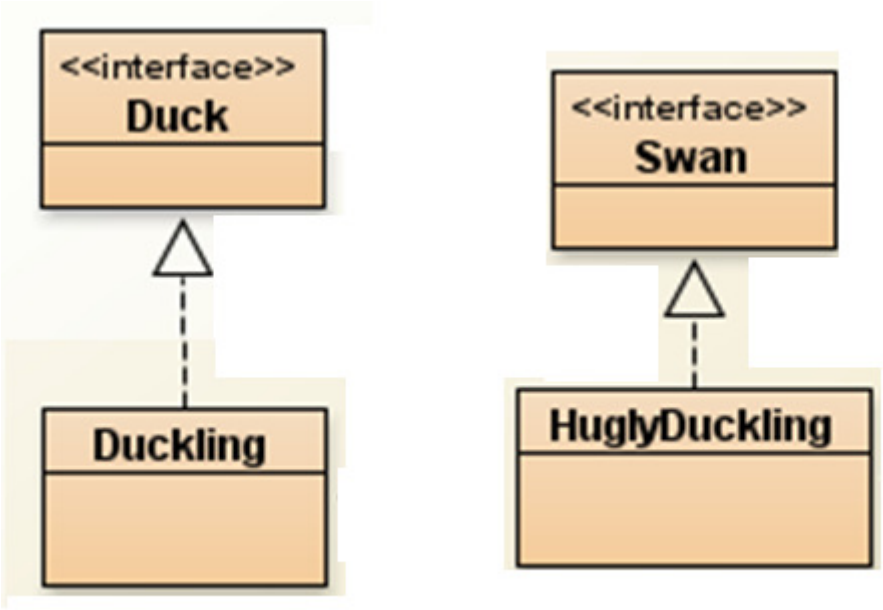


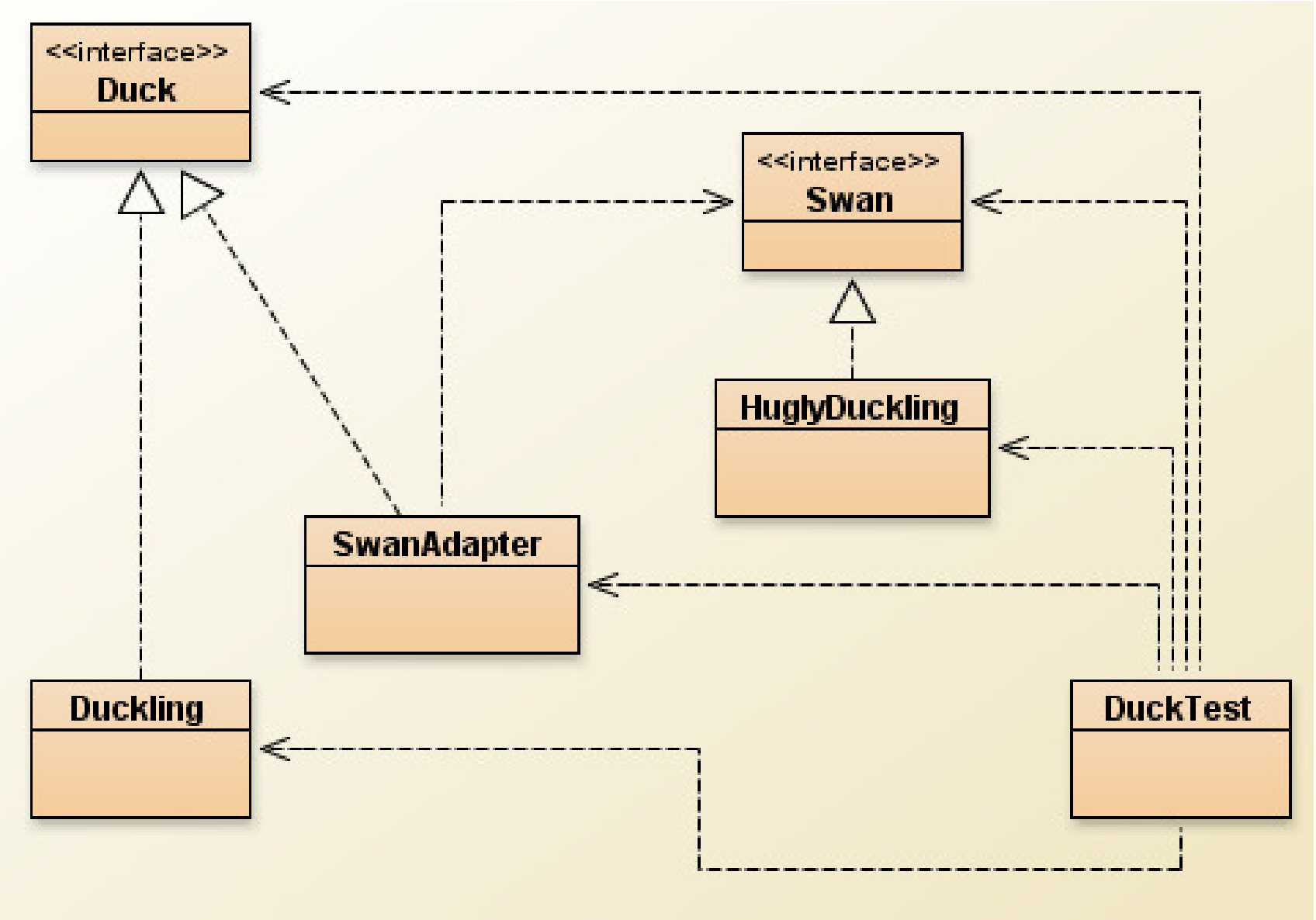
```
public interface Duck {
    public void display();
    public void swim(); }
```

```
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!'"");
    }
}
```


<<interface>>
Duck
Duckling
<<interface>>
Swan
HuglyDuckling


<<interface>>
Duck
<<interface>>
Swan
HuglyDuckling
SwanAdapter
Duckling
DuckTest

# SwanAdapter
## makes a swan look like a duck

```
public class SwanAdapter implements Duck {
   Swan swan;
   public SwanAdapter(Swan swan) {
         this.swan = swan;
   }

   public void display() {
         swan.show ();
   }

public void swim() {
         swan.swim();
   }
}
```

# Duck test drive

```
public class DuckTest {
    public static void main(String[] args) {
            Duck duck = new Duckling();
            Swan hg= new HuglyDuckling();
            Duck  swanAdapter = new SwanAdapter(hg);
            System.out.println("The HuglyDuckling  says...");
            hg.show();
            hg.swim();
            System.out.println("\nThe Duck says...");
            testDuck(duck);
            System.out.println("\nThe SwanAdapter says...");
            testDuck(swanAdapter);
    }
    static void testDuck(Duck duck) {
            duck.display();
            duck.swim();
    }
}
```

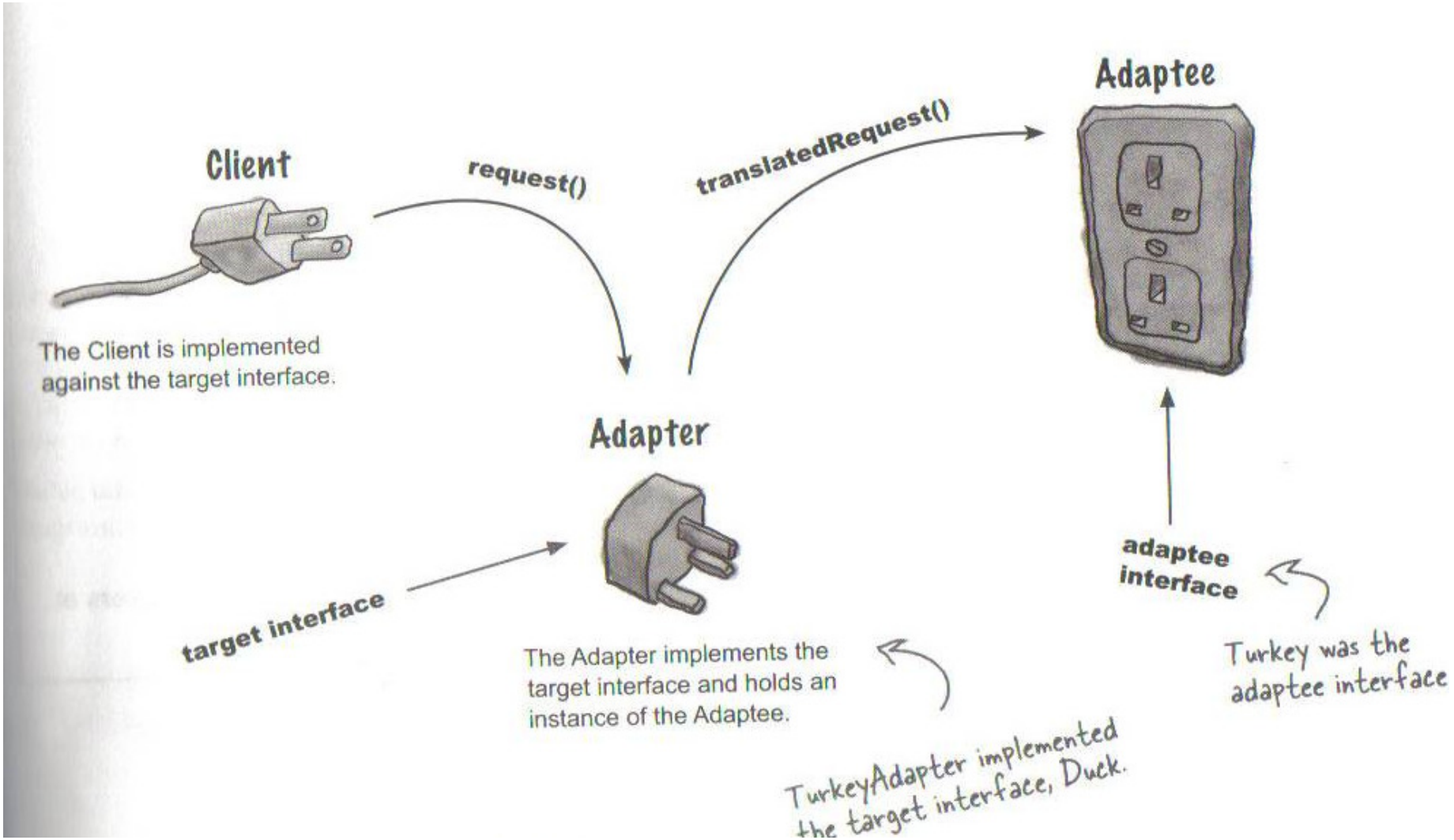
# Test run – swan that behaves like a duck

The HuglyDuckling  says...
I’m large and hugly
I'm swimming!”

The Duck says...
I'm a pretty little duckling
I'm learning…

The SwanAdapter says...
I’m large and hugly
I'm swimming!”

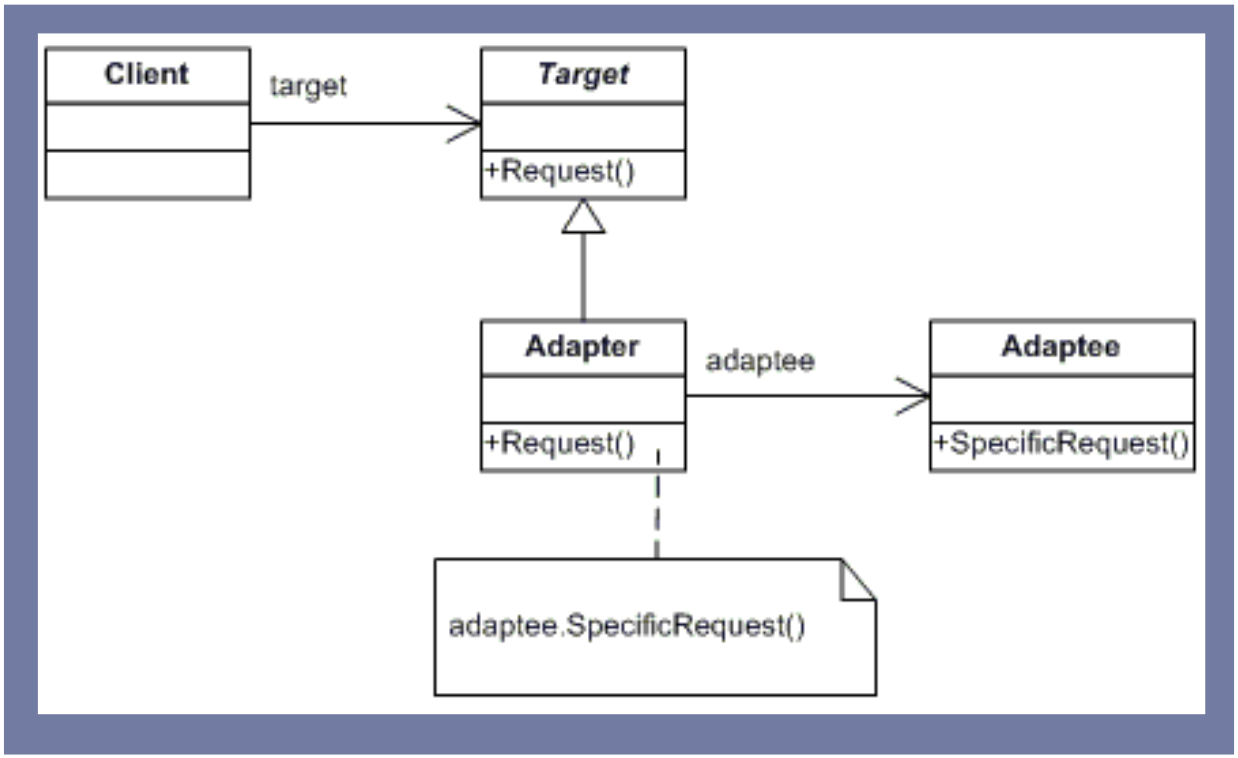
# Adapter Pattern explained



# 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

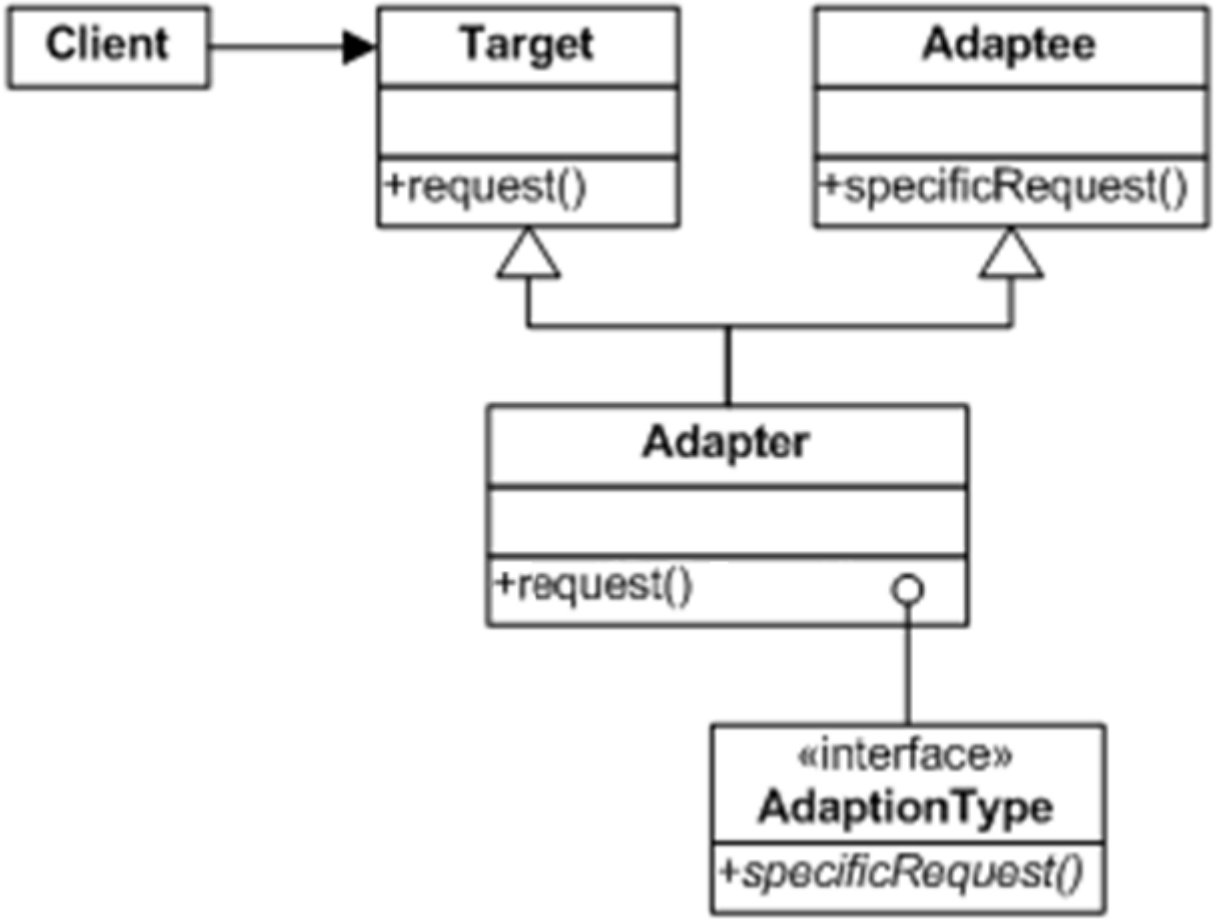



- Delegation is used to bind an Adapter to an Adaptee
- Interface inheritance is use to specify the interface of the Adapter class.
- Target and Adaptee (usually called legacy system) pre-exist the Adapter.
- Target may be realized as an interface in Java.

# 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 with multiple inheritance

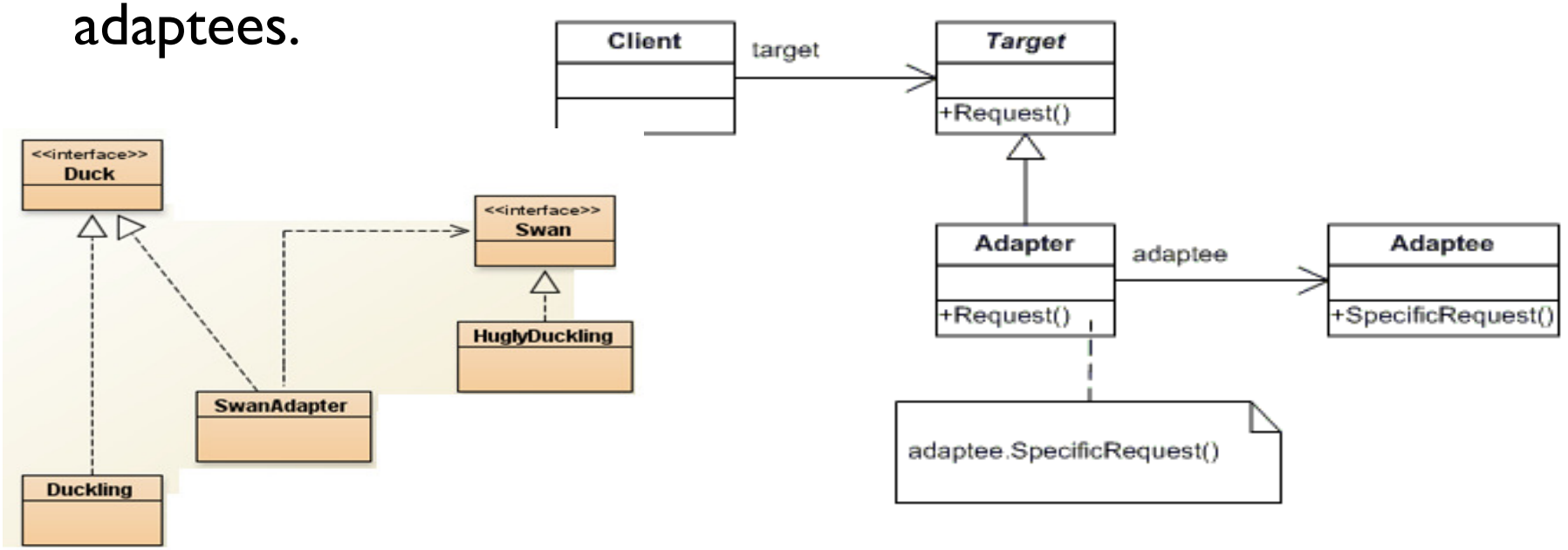


# Two-way adapter

- What if we want to have an adapter that acts as a Target or an Adaptee?
- Such an adapter is called a two-way adapter.
  - One way to implement two-way adapters is to use multiple inheritance, but we can't do this in Java
  - But we can have our adapter class implement two different Java interfaces
  - Twin pattern offeres a solution with delegation
  - Btw...do we really need it ?

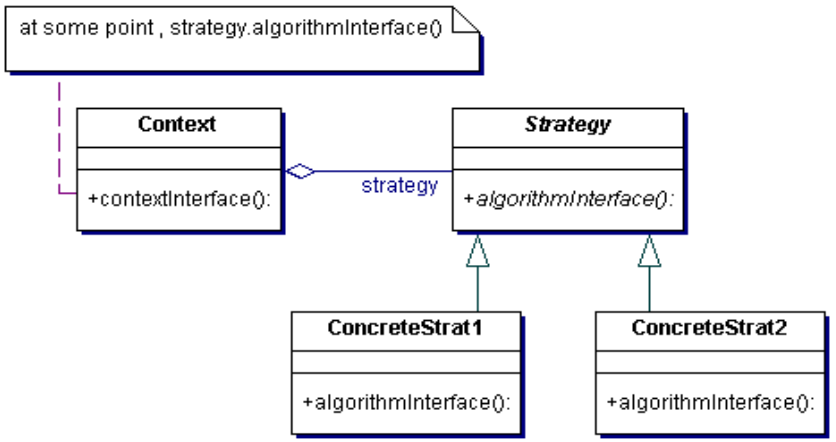
# Multiple Adapters for different Adaptees

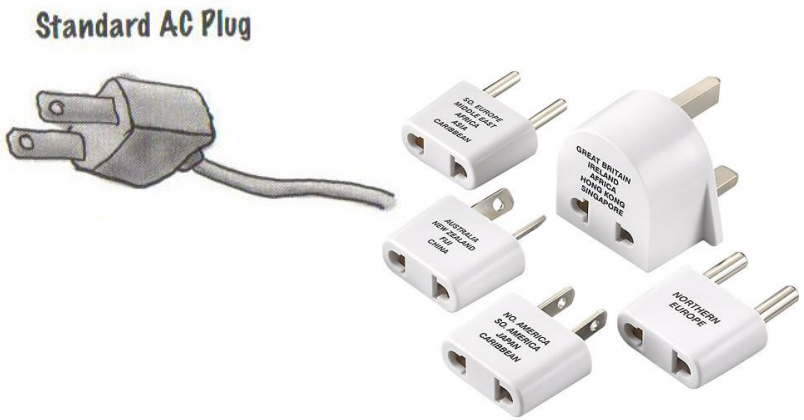
- Multiple implementations of adapters for different adaptees.




- Assume we need a turkey adapter.
  - Façade if the two adapters (and adaptees) cohexist
  - Startegy if Client refers to only one Adapter per time

# Adapter Pattern and Strategy Pattern




- There are many cases when the adapter can play the role of the Strategy Pattern. If we have several modules implementing the same functionality and we wrote adapters for them, the adapters are implementing the same interface. We can simply replace the adapters objects at run time because they implements the same interface.

# Façade

- (For assume it is possible to plug this)
- Forwards requests to many adaptees at a time
- Next lecture

# Homework

- Extend the ugly duckling example to adapt turkeys too.
  - Using strategy
  - Using Façade (see in a while)