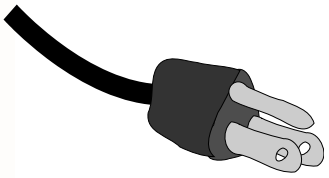
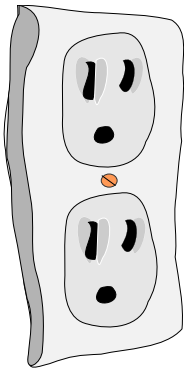
Adapter Pattern

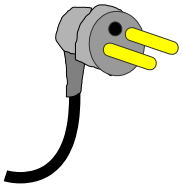


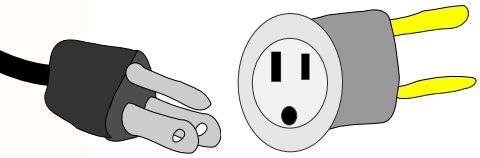
plug from US laptop expects a certain interface for power

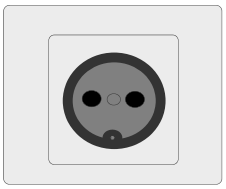


US wall outlet exposes an interface for getting power

adapter converts the German interface into a US interface

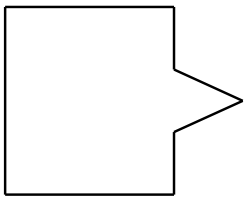




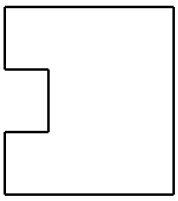


plug from US laptop expects a certain interface for power

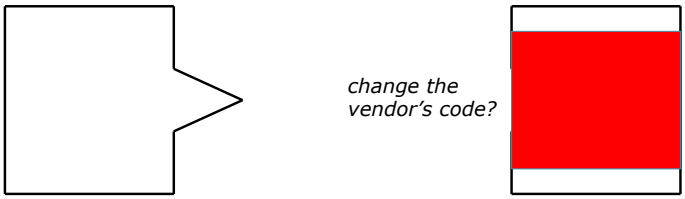
German wall outlet exposes an interface for getting power



your system expects a certain interface



vendor class provides a certain interface

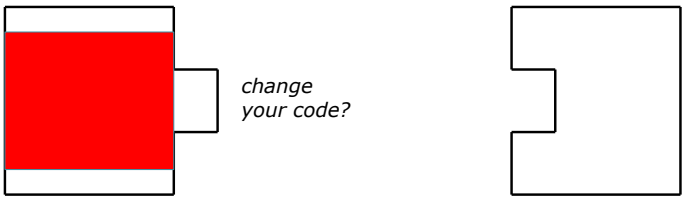


your system expects a certain interface

should not change the vendor's code

your system expects a certain interface

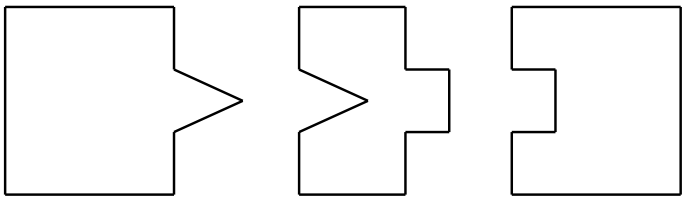
vendor class provides a certain interface



you do not want to change your code either

vendor class provides a certain interface

adapter implements the interface your system expects



your system (no change)

adapter converts requests from your system to use the vendor class

vendor class (no change)

Adapter Pattern

Design intent:

"convert the interface of a class into another interface that clients expect"

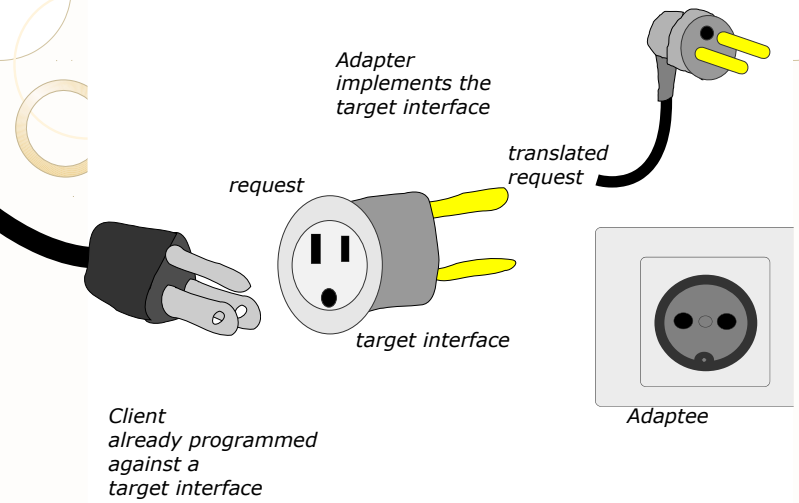
"lets classes work together that couldn't otherwise because of incompatible interfaces"

also known as a wrapper

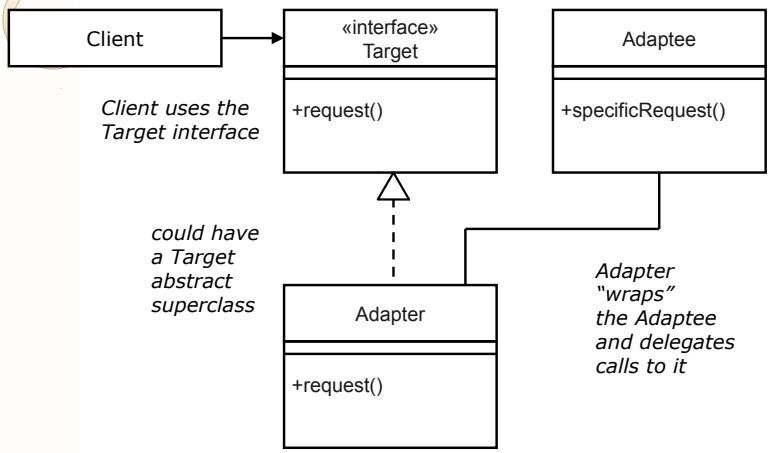
Motivation

Use:

adapting existing third-party components to suit your conventions or interfaces



Object Adapter Structure



```
// target interface
public interface Duck {
    public void fly();
    public void quack();
}
```

```
// adaptee
public class Turkey {
    public void fly() { … }
    public void gobble() { … }
}

// turkeys fly 1/5 the
// distance of a duck
```

```
// adapter
public class TurkeyAdapter … {
    …

    public TurkeyAdapter( … ) {
        …
    }
    public void fly() {
        …
    }
    public void quack() {
        …
    }
}
```

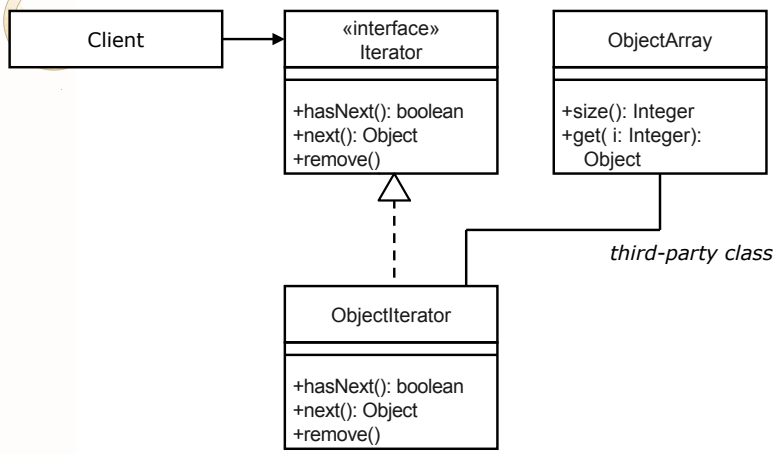


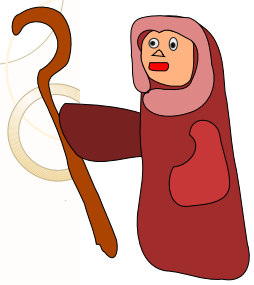
```
// target interface
public interface Duck {
    public void fly();
    public void quack();
}
```

```
// adaptee
public class Turkey {
    public void fly() { … }
    public void gobble() { … }
}

// turkeys fly 1/5 the
// distance of a duck
```

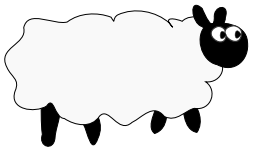
```
// adapter
public class TurkeyAdapter implements Duck {
    Turkey turkey;

    public TurkeyAdapter( Turkey turkey ) {
        this.turkey = turkey;
    }
    public void fly() {
        for (int i = 0; i < 5; i++) turkey.fly();
    }
    public void quack() {
        turkey.gobble();
    }
}
```


Object Adapter Example

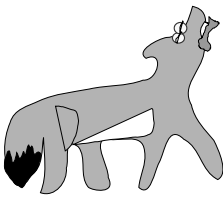




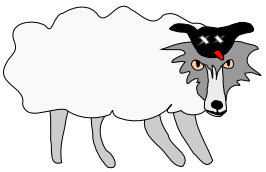
shepherd



"sheeplike"



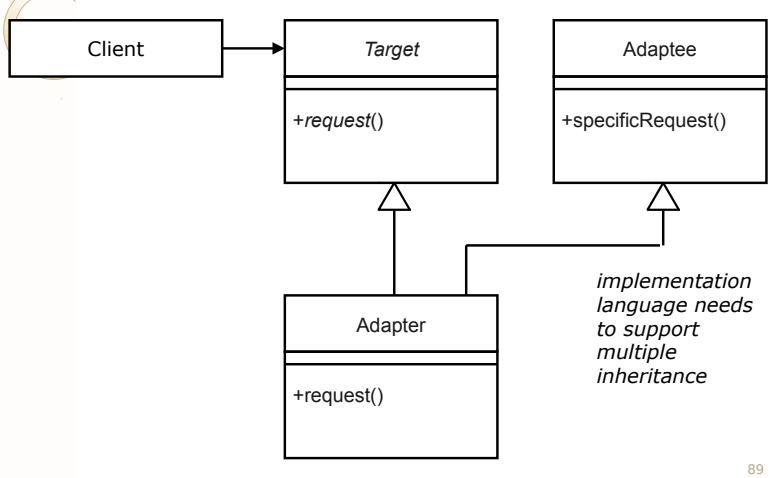
wolf



Wolf in sheep's clothing

how does the shepherd tend a wolf?

Class Adapter Structure



Consequences

Object adapter:

- more flexible since a single Adapter could adapt many Adaptees

Class adapter:

- related to Adaptee via implementation inheritance
- can override Adaptee
- less delegation

Question

True or false?

Adapting a large interface takes a lot of work.

Adapters only adapt a single class.