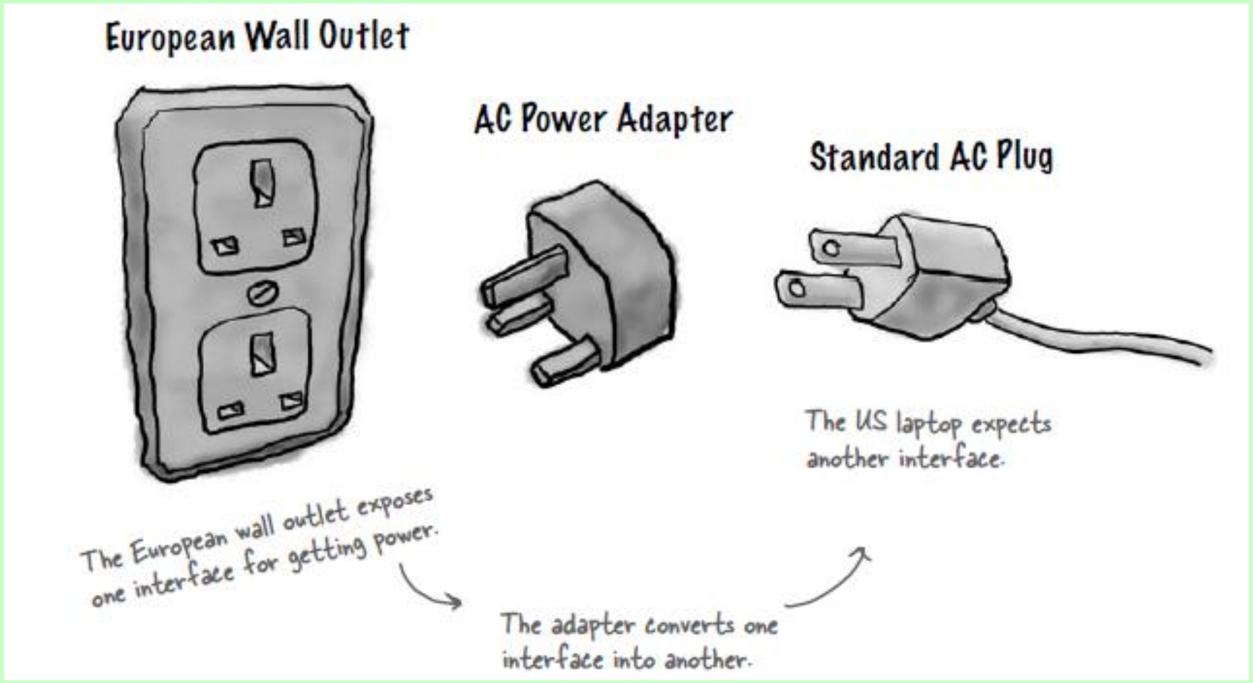
Adapter Pattern

Being Adaptive

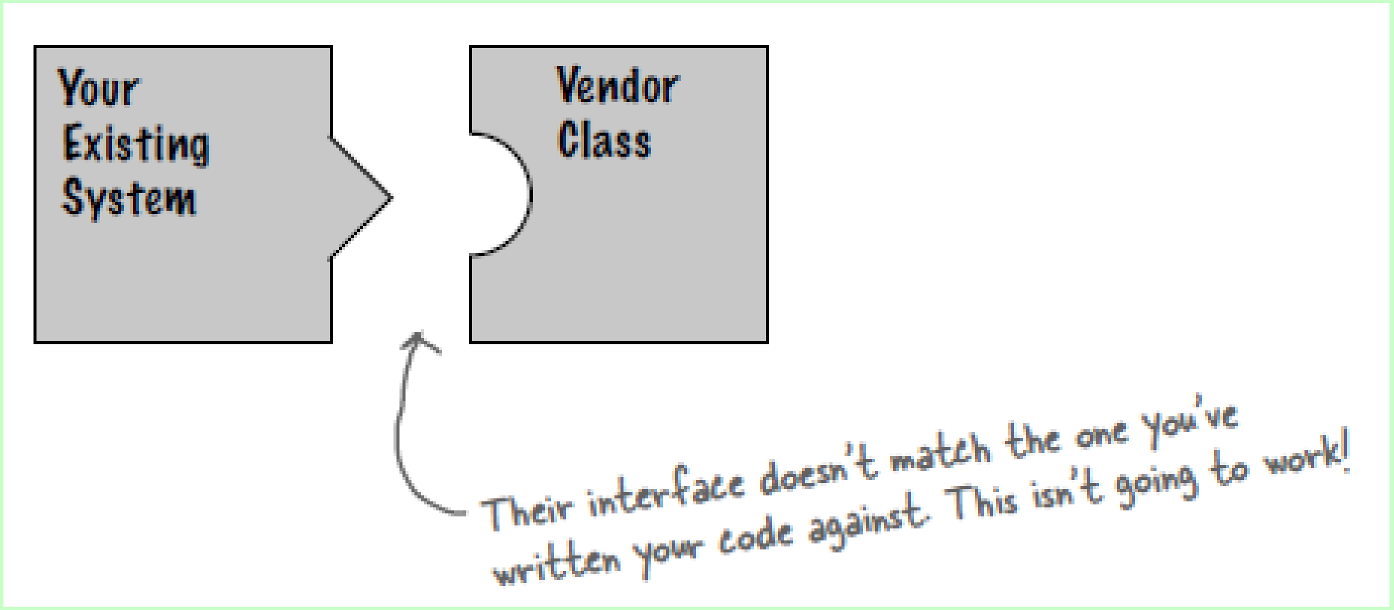
Adapters all around us

- You’ ll have no trouble understanding what an OO adapter is because the real world is full of them.


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.

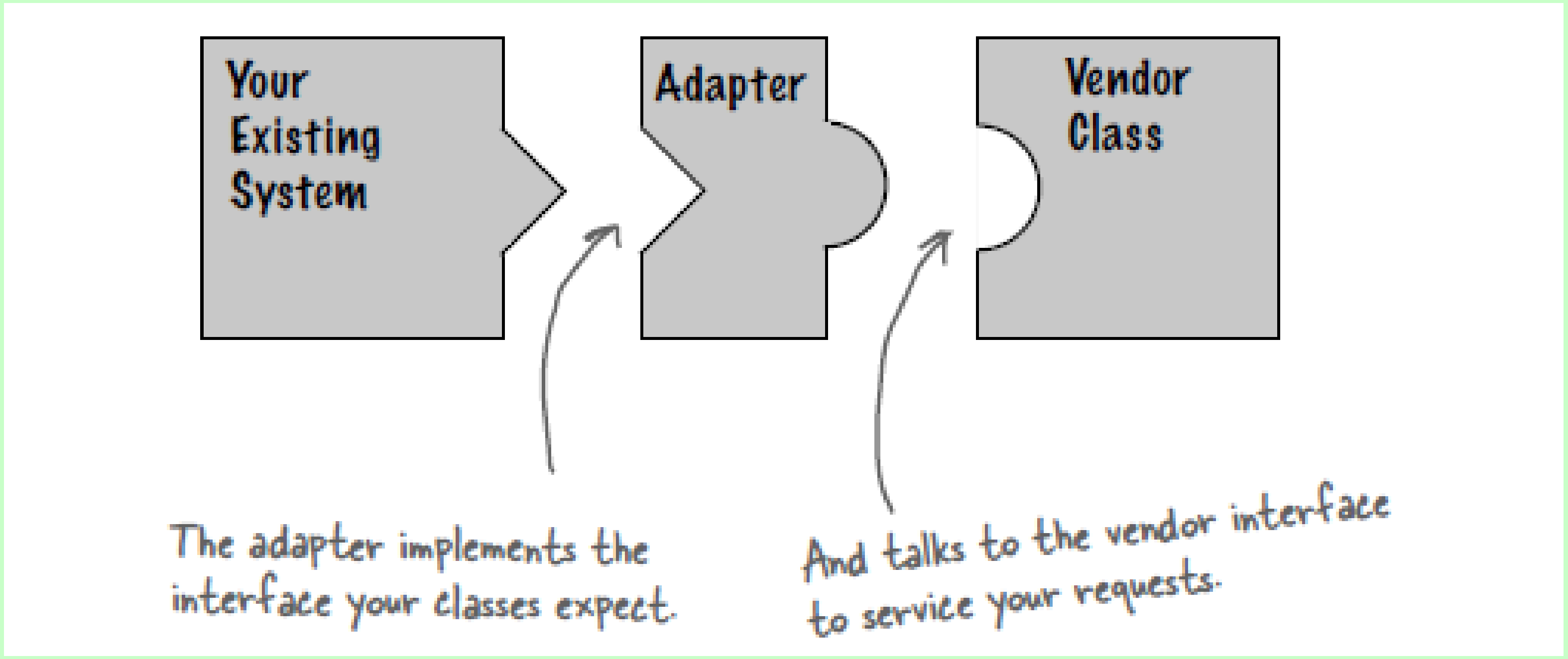
Object oriented adapters

- 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!

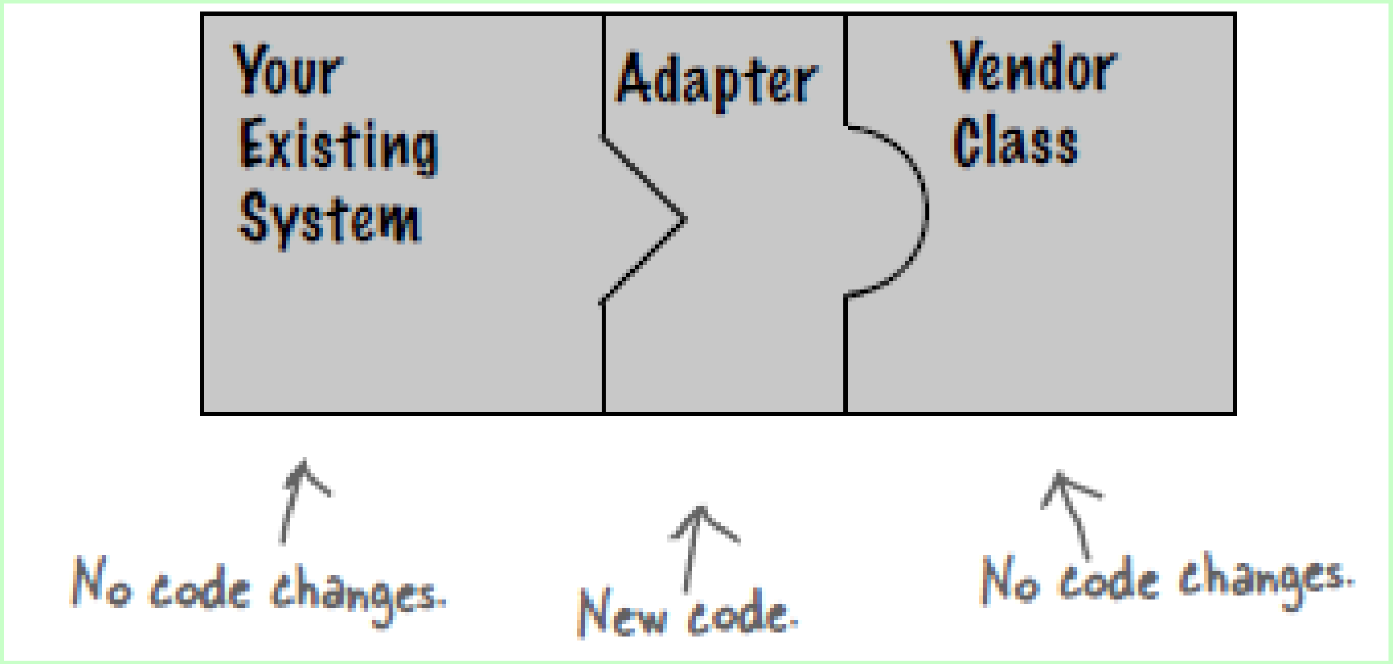
Object oriented adapters

- You don’t want to solve the problem by changing your existing code.


Your Existing System
Adapter
Vendor Class
The adapter implements the interface your classes expect.
And talks to the vendor interface to service your requests.

Object oriented adapters

- The adapter acts as the middleman by receiving requests from the client and converting them into requests that make sense on the vendor classes.



Duck Interface

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

MallardDuck implements Duck

```
public class MallardDuck
  implements Duck {
  public void quack() {

     System.out.println(“Quack”)
  ;
  }
  public void fly() {
     System.out.println(“I’m
  flying”);
  }
```

Turkey Interface

```
public interface Turkey {
  public void gobble();
  public void fly();
}
```

WildTurkey implements Turkey

```
public class WildTurkey implements
  Turkey {
  public void gobble() {
     System.out.println(“Gobble
  gobble”);
  }
  public void fly() {
     System.out.println(“I’m flying a
  short  distance”);
  }
}
```

Use Turkey as a Duck

- You’ re short on Duck objects and you’ d like to use some Turkey objects in their place

- Obviously we can’ t use the turkeys outright because they have a different interface.

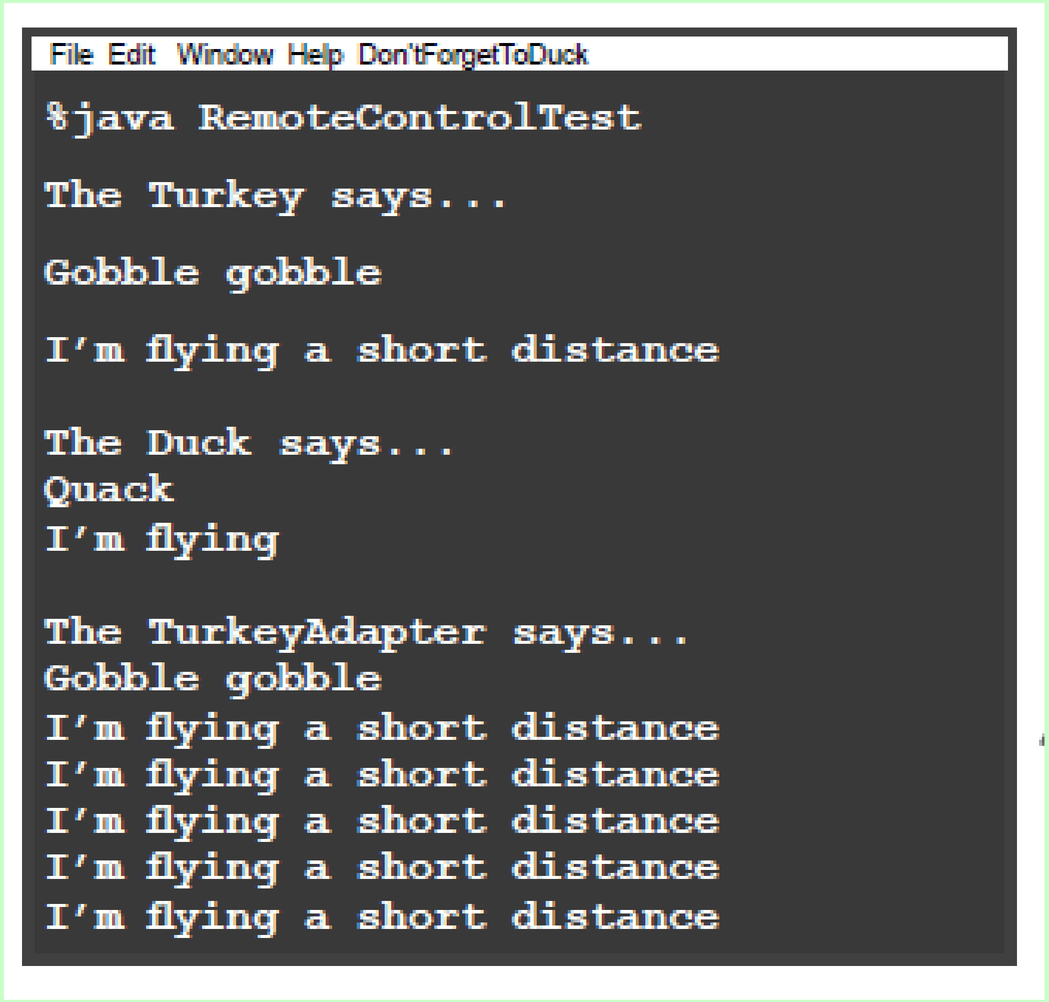
Use Turkey as a Duck

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

Test drive the adapter

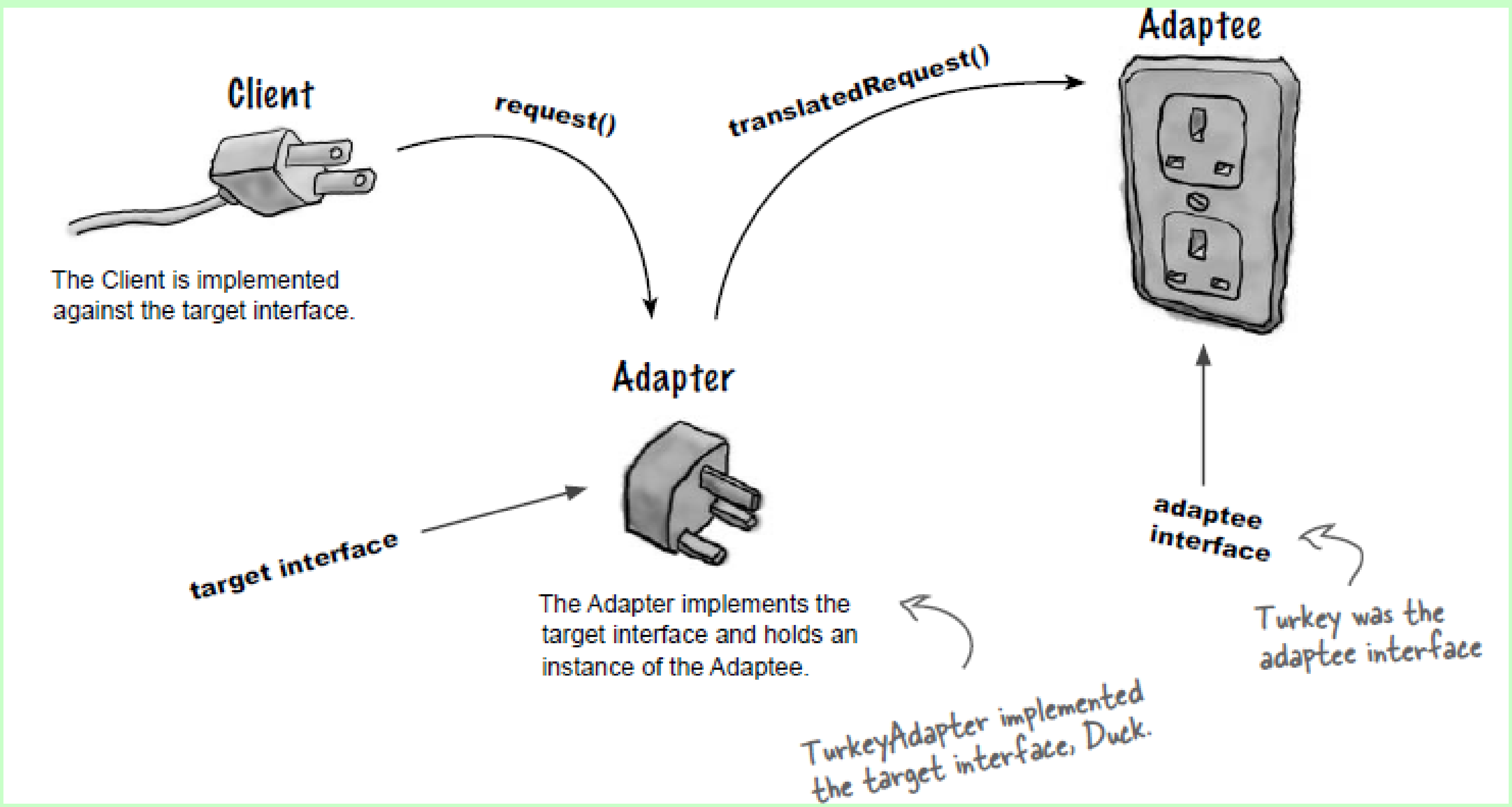
```
public class DuckTestDrive {
    public static void main(String[] args) {
          MallardDuck duck = new MallardDuck();
          WildTurkey turkey = new WildTurkey();
          Duck turkeyAdapter = new TurkeyAdapter(turkey);

          System.out.println(“The Turkey says...”);
          turkey.gobble();
          turkey.fly();

          System.out.println(“\nThe Duck says...”);
          testDuck(duck);
          System.out.println(“\nThe TurkeyAdapter says...”);
          testDuck(turkeyAdapter);
    }
    static void testDuck(Duck duck) {
          duck.quack();
          duck.fly();
    }
}
```

Test drive the adapter



```
%java RemoteControlTest

The Turkey says...

Gobble gobble

I’m flying a short distance

The Duck says...
Quack
I’m flying

The TurkeyAdapter says...
Gobble gobble
I’m flying a short distance
I’m flying a short distance
I’m flying a short distance
I’m flying a short distance
I’m flying a short distance
```

The Adapter Pattern explained


Client
request()
translatedRequest()
Adaptee
The Client is implemented against the target interface.
Adapter
target interface
adaptee 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.

How the Client uses Adapter

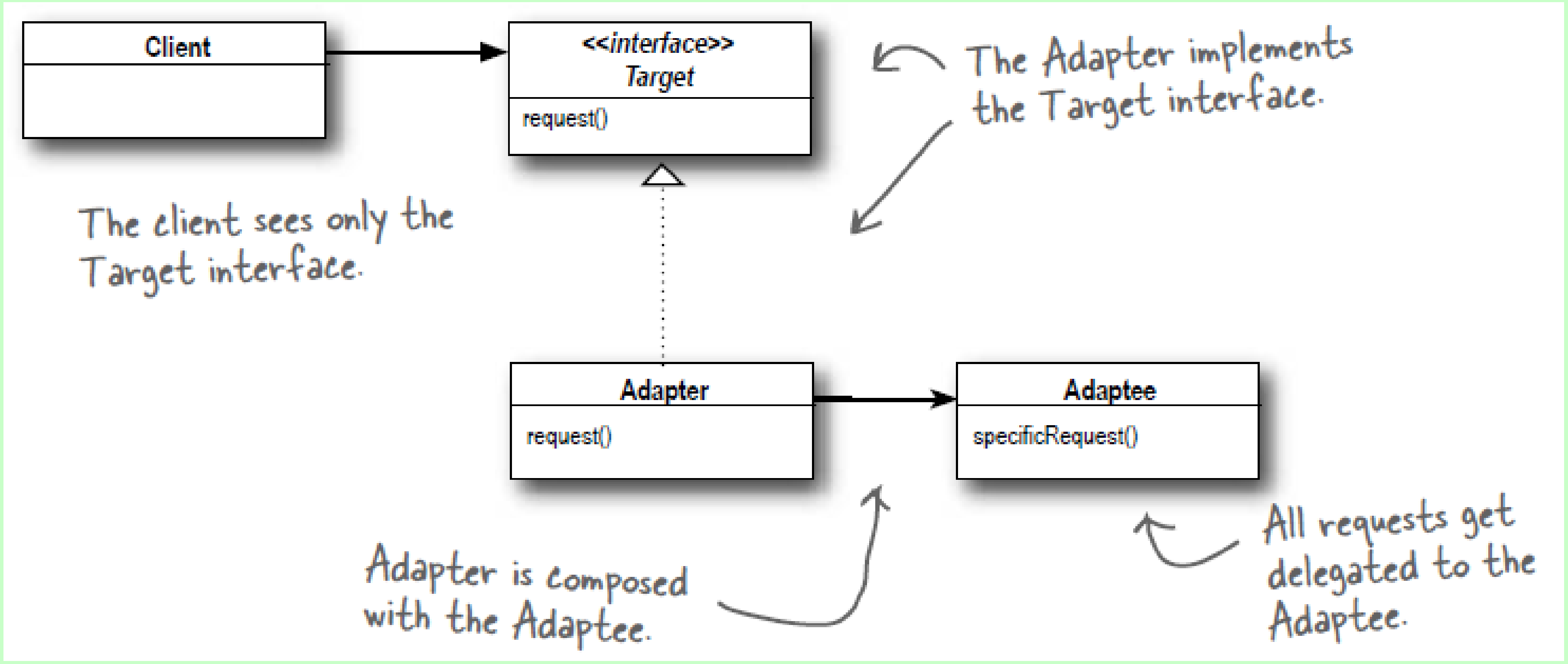
- The client makes a request to the adapter by calling a method on it using the target interface
- The adapter translates the request into one or more calls on the adaptee using the adaptee interface.
- The client receives the results of the call and never knows there is an adapter doing the translation.

Adapter Pattern defined

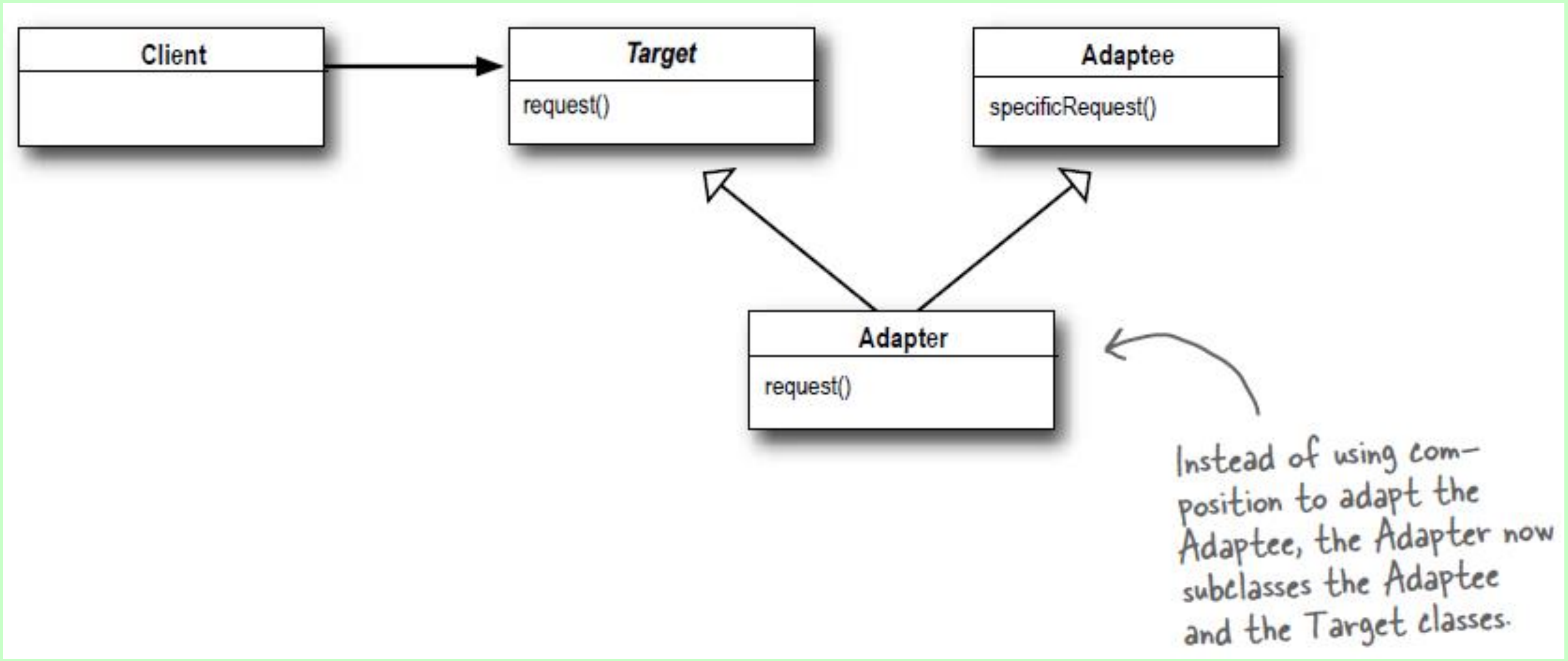
- The Adapter Pattern converts the interface of a class into another interface the clients expect.

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

Adapter Pattern defined

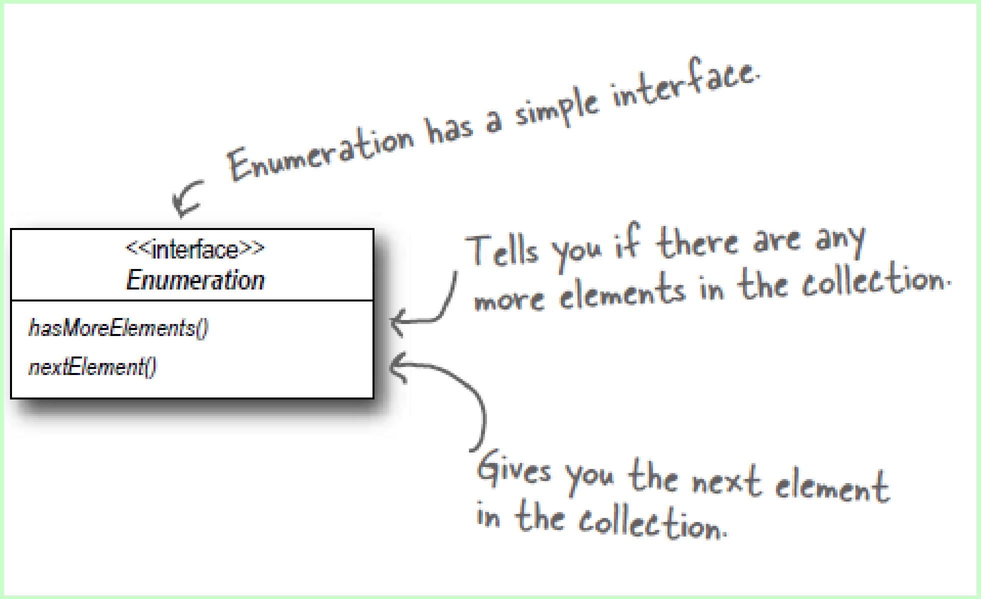

Client
<<interface>>
Target
request()
The Adapter implements the Target interface.
The client sees only the Target interface.
Adapter
request()
Adaptee
specificRequest()
Adapter is composed with the Adaptee.
All requests get delegated to the Adaptee.

Adapter Pattern using Multiple Inheritance (Class Adapter)


Client
Target
request()
Adaptee
specificRequest()
Adapter
request()
Instead of using composition to adapt the Adaptee, the Adapter now subclasses the Adaptee and the Target classes.

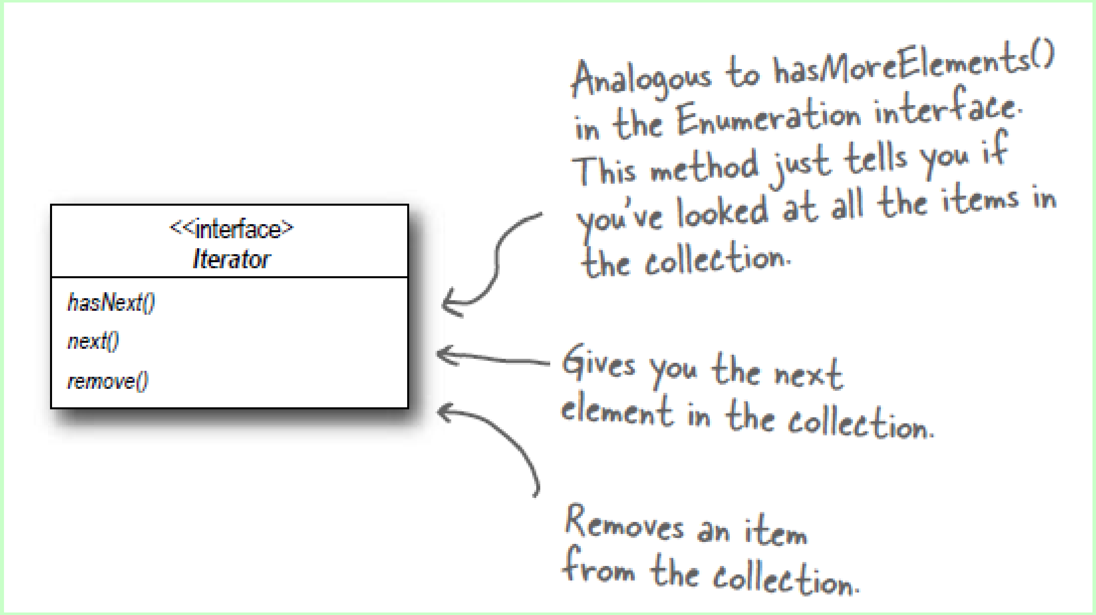
Real world adapters

- Enumerators: early collections types (Vector, Stack, Hashtable, and a few others) implement a method elements(), which returns an Enumeration.


Enumeration has a simple interface.
<<interface>>
Enumeration
hasMoreElements()
nextElement()
Tells you if there are any more elements in the collection.
Gives you the next element in the collection.

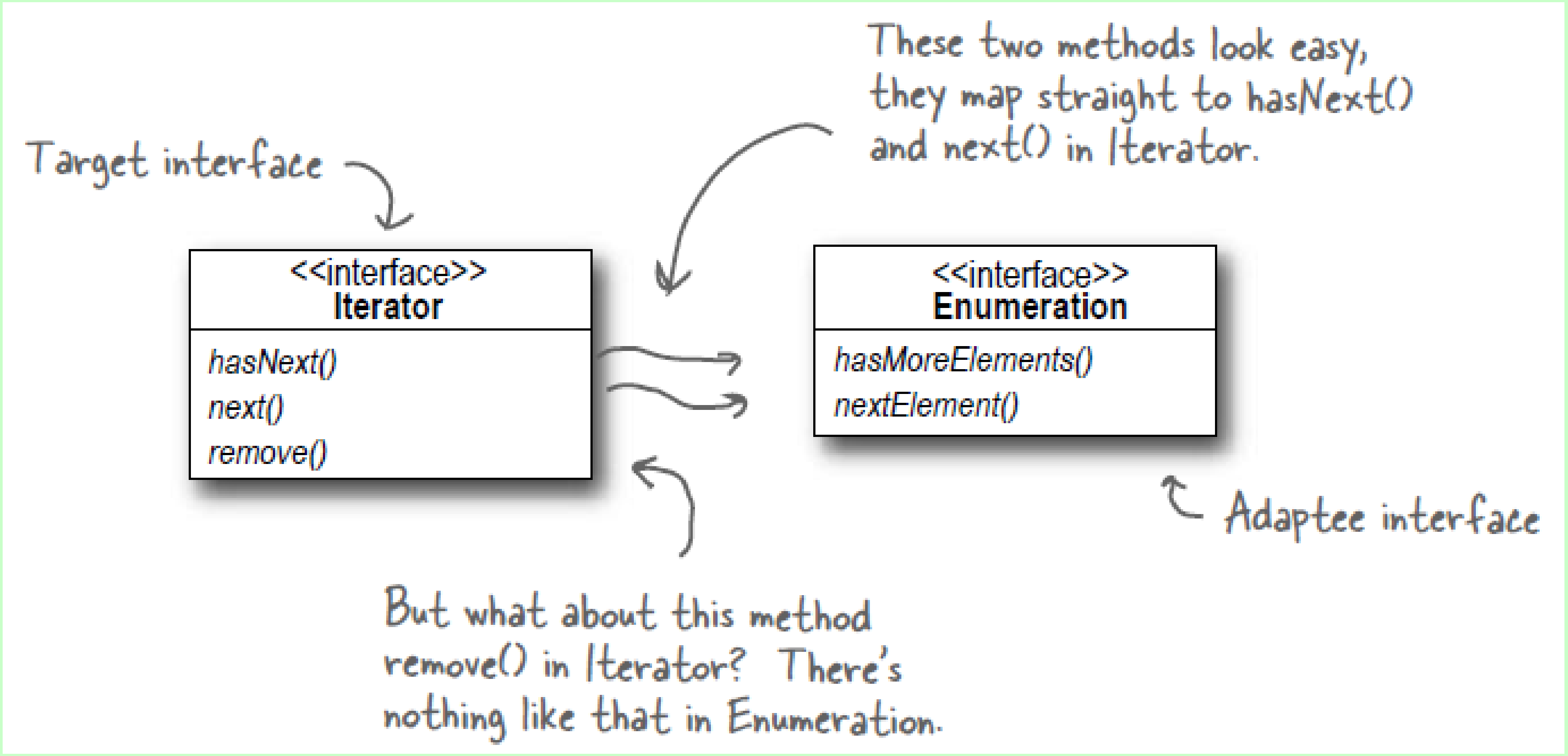
Real world adapters

- **Iterators**: more recent Collections classes they began using an Iterator interface that, like Enumeration,allows you to iterate through a set of items in a collection, but also adds the ability to remove items.

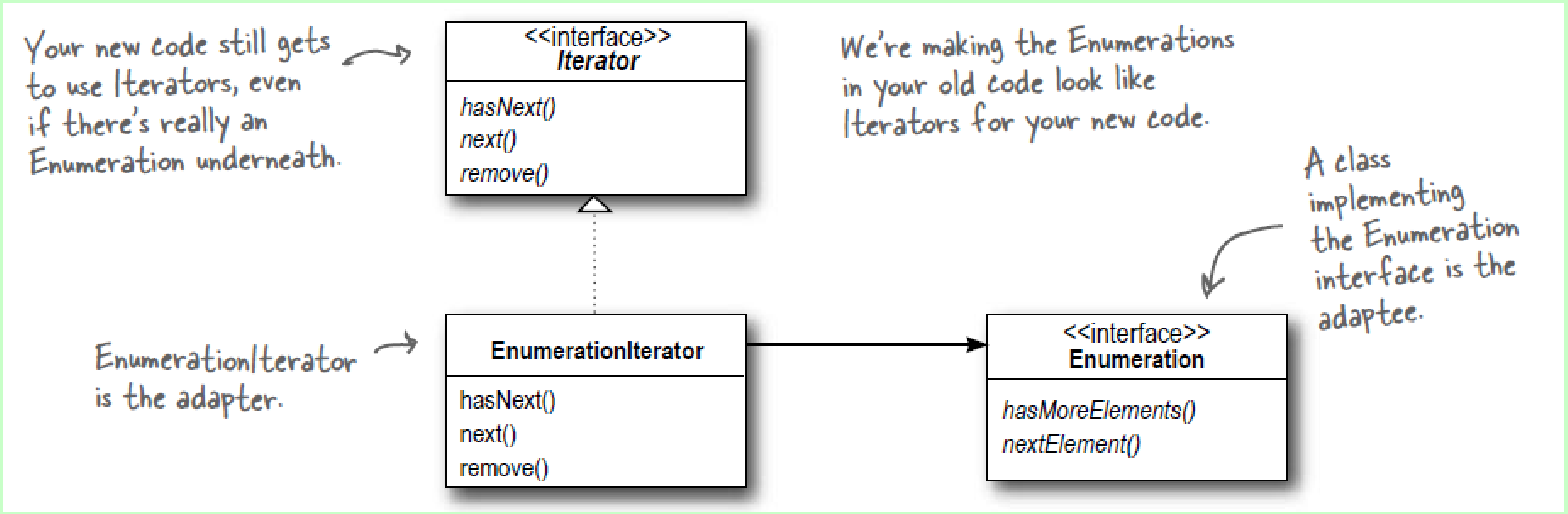


<<interface>>
Iterator
hasNext()
next()
remove()
Analogous to hasMoreElements() in the Enumeration interface. This method just tells you if you've looked at all the items in the collection.
Gives you the next element in the collection.
Removes an item from the collection.

Adapting an Enumeration to an Iterator


Target interface
These two methods look easy, they map straight to hasNext() and next() in Iterator.
<<interface>> Iterator
hasNext()
next()
remove()
<<interface>> Enumeration
hasMoreElements()
nextElement()
Adaptee interface
But what about this method remove() in Iterator? There's nothing like that in Enumeration.

Designing the Adapter


Your new code still gets to use Iterators, even if there's really an Enumeration underneath.
<<interface>>
Iterator
hasNext()
next()
remove()
We're making the Enumerations in your old code look like Iterators for your new code.
A class implementing the Enumeration interface is the adaptee.
EnumerationIterator is the adapter.
EnumerationIterator
hasNext()
next()
remove()
<<interface>>
Enumeration
hasMoreElements()
nextElement()

Dealing with the remove() method

- Enumeration just doesn’t support remove. It’s a “read only” interface.
- There’s no way to implement a fully functioning remove() method on the adapter.
- The best we can do is throw a runtime exception.
- This is a case where the adapter isn’t perfect;

Writing the EnumerationIterator adapter

```
public class EnumerationIterator implements Iterator {
   Enumeration enum;
   public EnumerationIterator(Enumeration enum) {
        this.enum = enum;
   }
   public boolean hasNext() {
        return enum.hasMoreElements();
   }
   public Object next() {
        return enum.nextElement();
   }
   public void remove() {
        throw new UnsupportedOperationException();
   }
}
```

References

- Design Patterns: Elements of Reusable Object-Oriented Software By Gamma, Erich; Richard Helm, Ralph Johnson, and John Vlissides (1995). Addison-Wesley. ISBN 0-201-63361-2.
- **Head First Design Patterns** By Eric Freeman, Elisabeth Freeman, Kathy Sierra, Bert Bates
  First Edition October 2004
  ISBN 10: 0-596-00712-4