CS342: Software Design

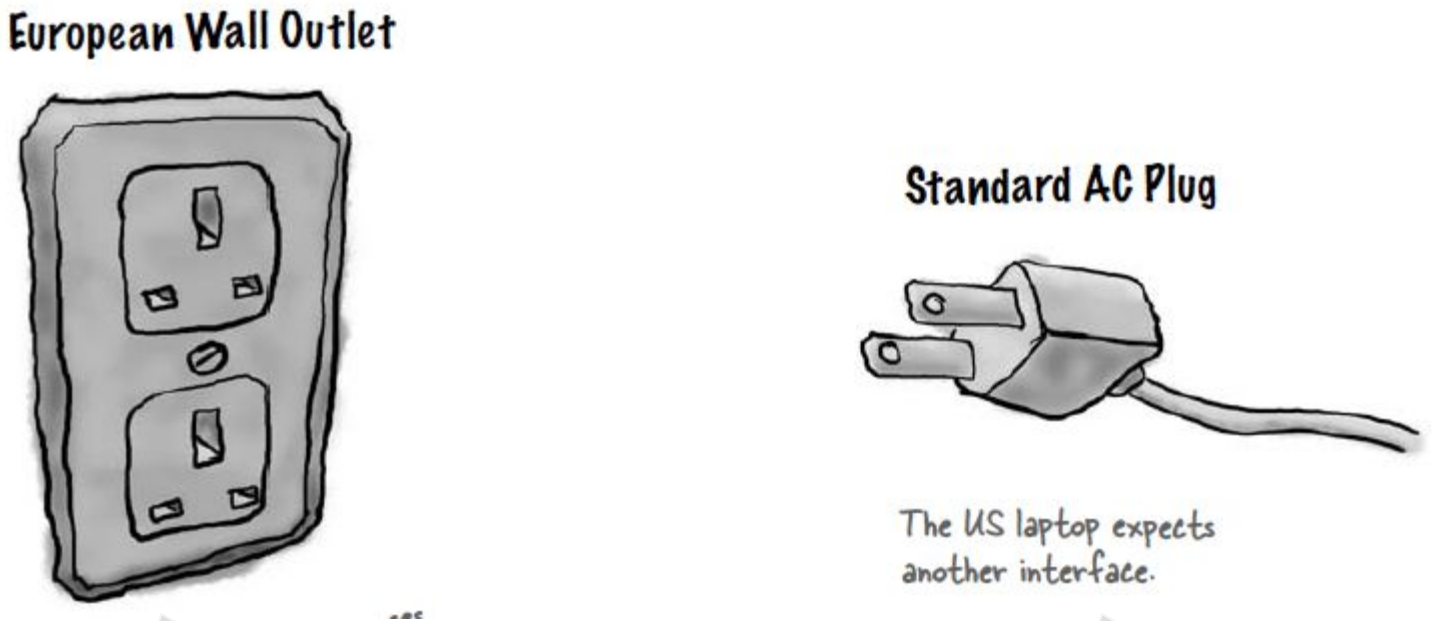
Dec. 5, 2017

Today’s topic

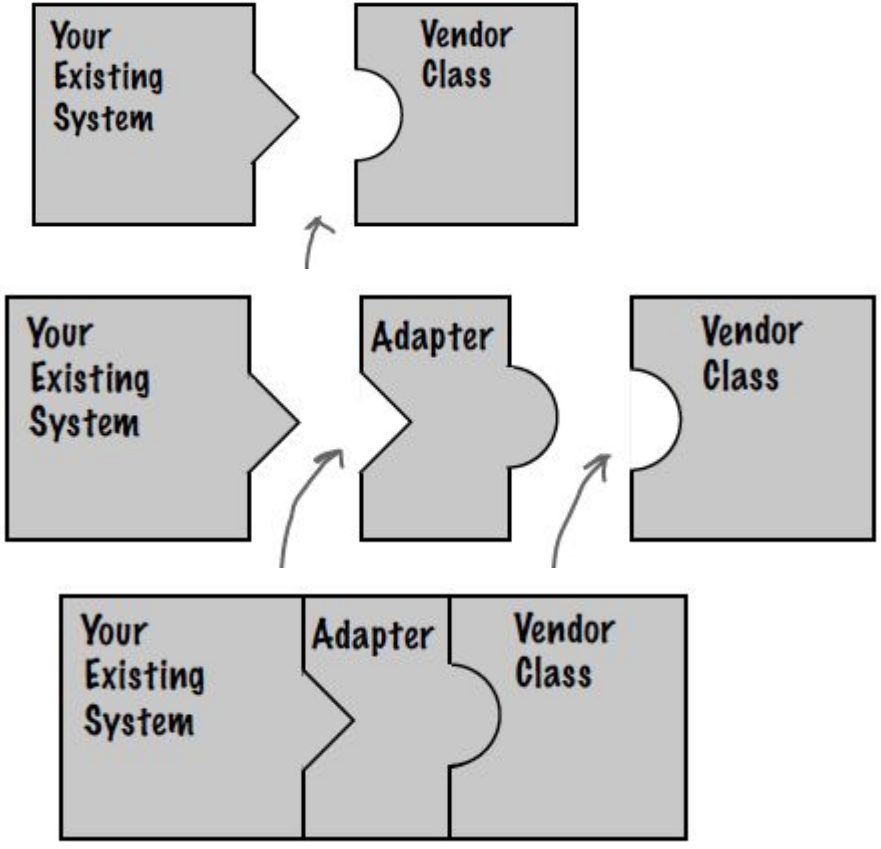
Adapter pattern

Solution stack

US plug vs European outlet



Object oriented adapters



Adapter pattern

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

- Lets classes work together that couldn’t otherwise because of incompatible interfaces.
- Decouple the client from the implemented interface.
- Encapsulates that change so that the client doesn’t have to be modified each time it needs to operate against a different interface

Duck vs. turkey

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

```
public class MallardDuck implements Duck {
    public void quack() {
        System.out.println("Quack");
    }
    public void fly() {
        System.out.println("I'm flying");
    }
}
```

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

```
static void testDuck(Duck duck) {
    duck.quack();
    duck.fly();
}
```

```
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");
    }
}
```

Adapter class

```
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();
    }
}
```

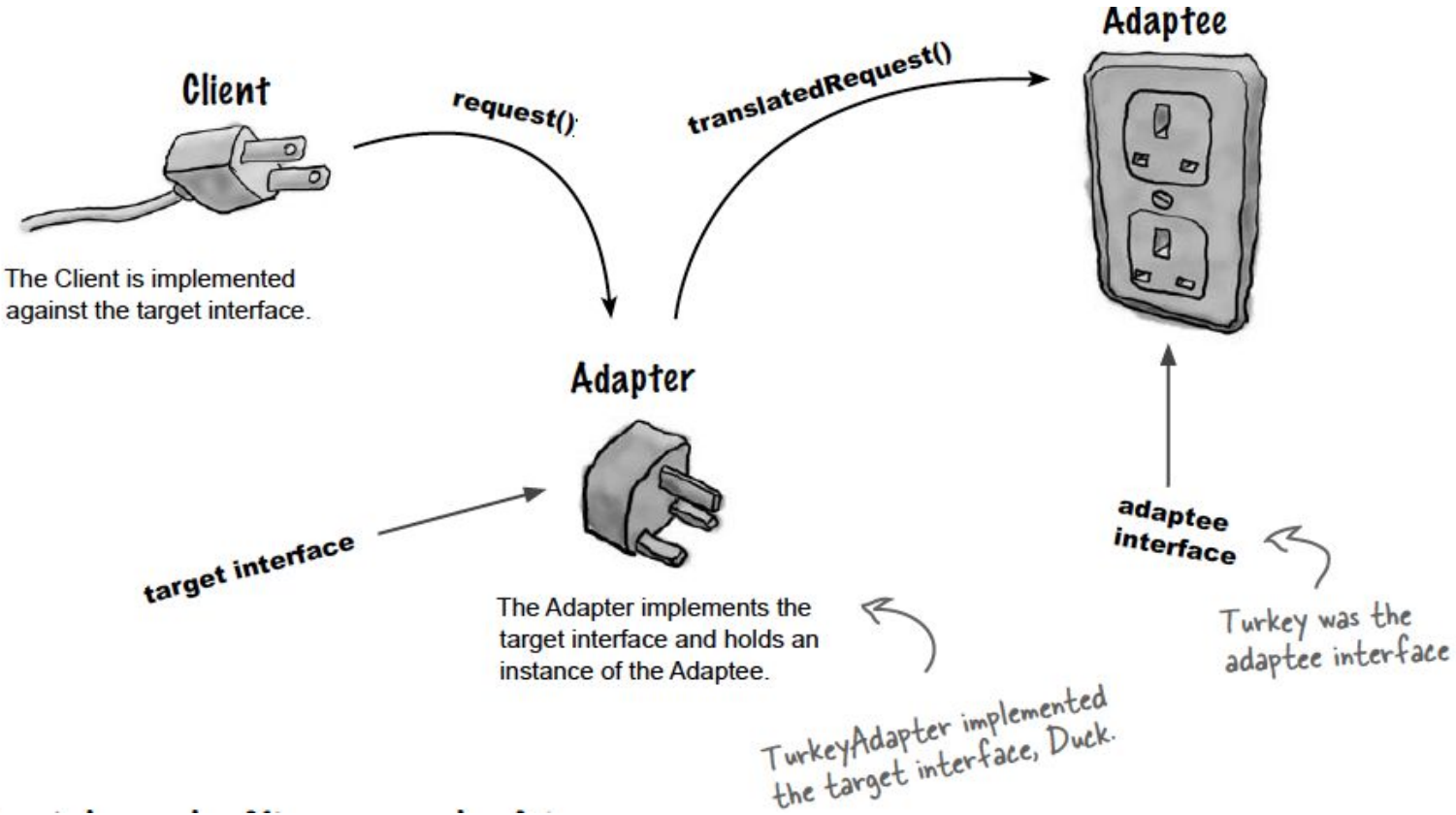
```
File Edit Window Help Don'tForgetToDuck
%java RemoteControlTest

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
```

Adapter pattern

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

- Lets classes work together that couldn’t otherwise because of incompatible interfaces.
- Decouple the client from the implemented interface.
- Encapsulates that change so that the client doesn’t have to be modified each time it needs to operate against a different interface

Adapter pattern explained



How does the client use the adapter?

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

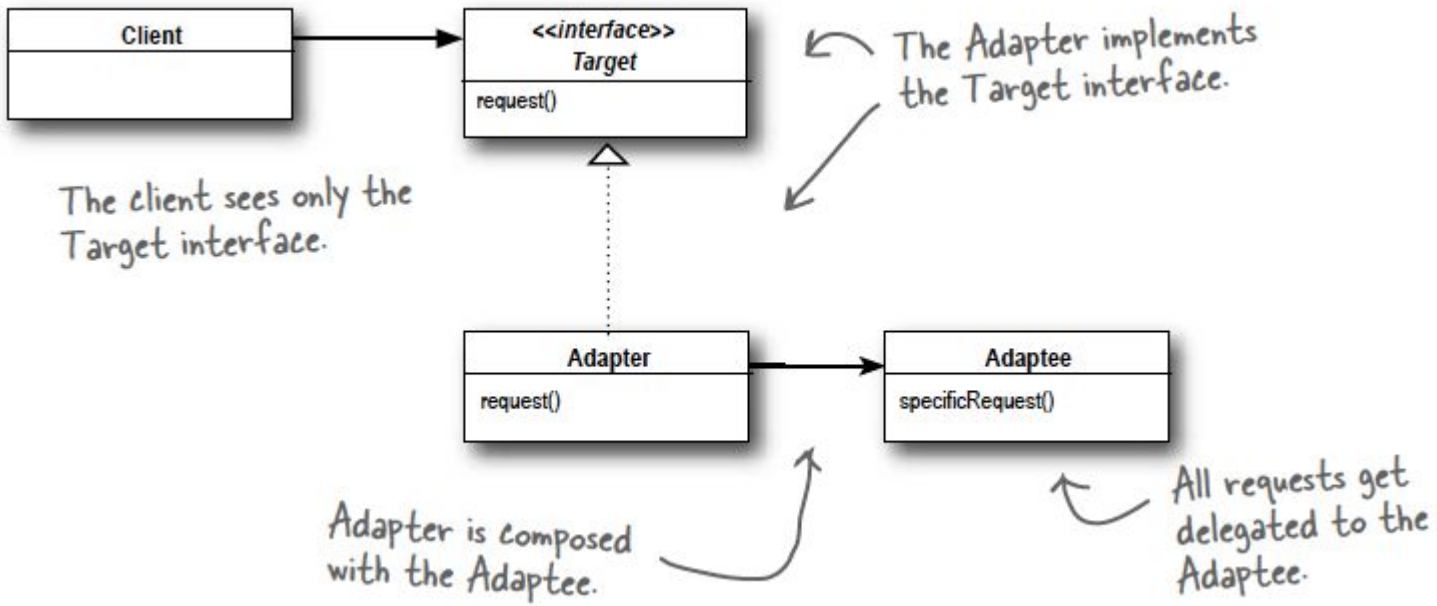
```
WildTurkey turkey = new WildTurkey();
Duck turkeyAdapter =
    new TurkeyAdapter(turkey);
turkeyAdapter.fly();
```

```
public class TurkeyAdapter implements Duck {
    Turkey turkey;
    public TurkeyAdapter(Turkey turkey) {
        this.turkey = turkey;
    }
    public void quack() {
        turkey.gobble();
    }
    public void fly() {
```

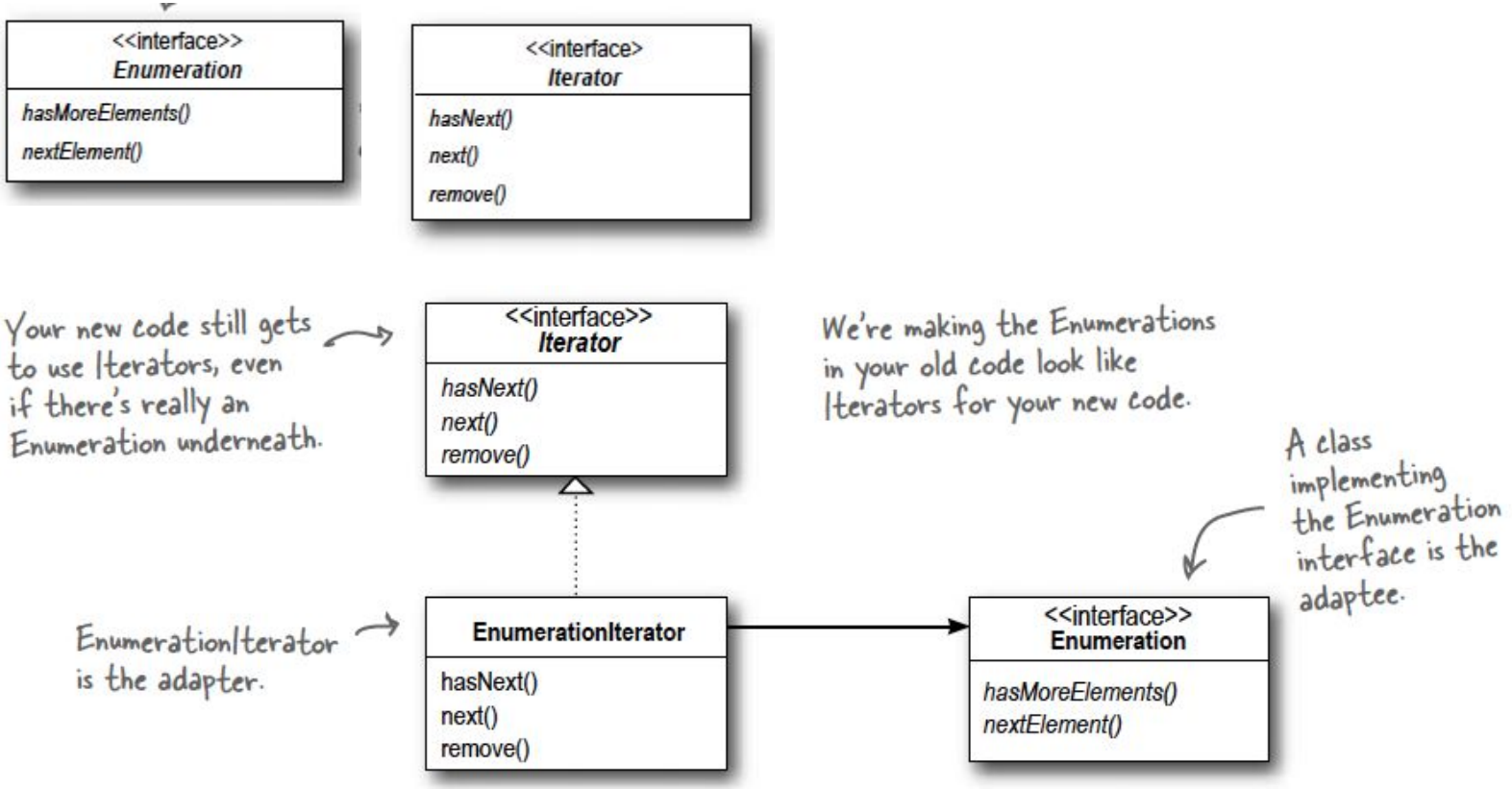
Adapter pattern advantages

- Object composition
- Program to an interface, not implementation
- Add new implementations without changing interface

Adapter diagram



Enumerator vs. Iterator



EnumerationIterator

```
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();
    }
}
```

Technology stack (aka solution stack, software stack)

A set of software components to create a complete platform such that no additional software is needed to support applications

- LAMP - linux, apache, mysql, php
- NMP - Nginx, mysql, Php
- WINS (WISA) - Windows, IIS, .NET, SQL
- Java EE
- SMACK: Apache Spark, Mesos, Akka, Cassandra, Kafka
- MEAN/MERN/MEVN

Free and open-source JavaScript software stack for building dynamic web sites and web applications

- Mongodb: document oriented db (non-relational)
- Express.js: web application framework, middleware (backend)
- Angular.js: web application framework (front end, MVC)
- Node.js: Javascript runtime for server side

Similar stacks

MERN, MEVN: react.js (facebook, instagram), vue.js (Google)

- Javascript and JSON in all layers (show examples, code, db, service call)
- Advantages?