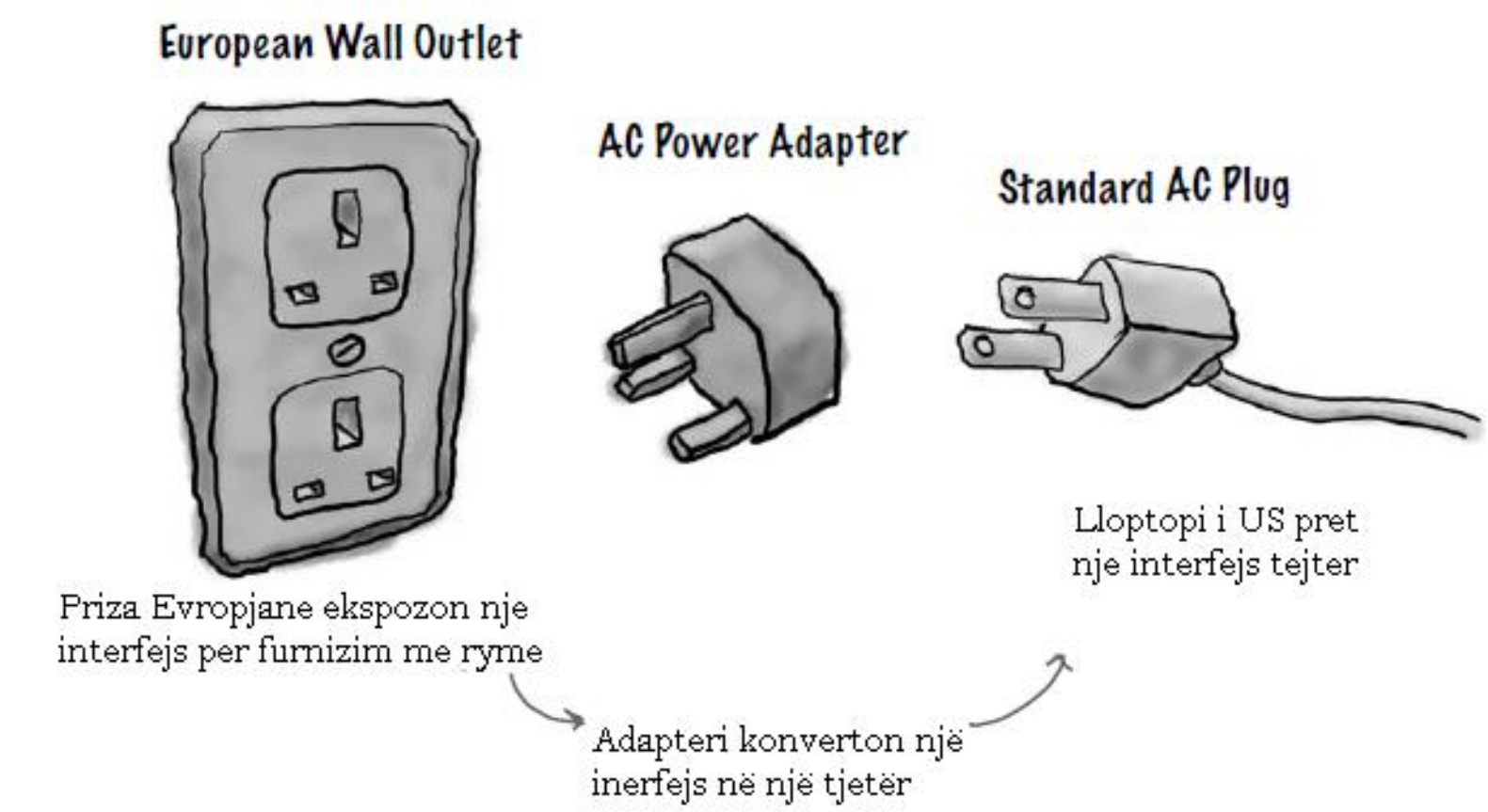
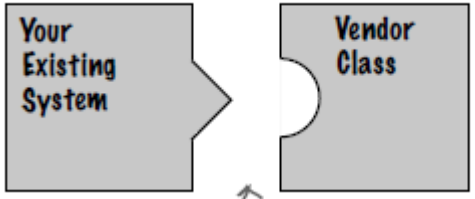
ADAPTER PATTERN

Mentor Shala

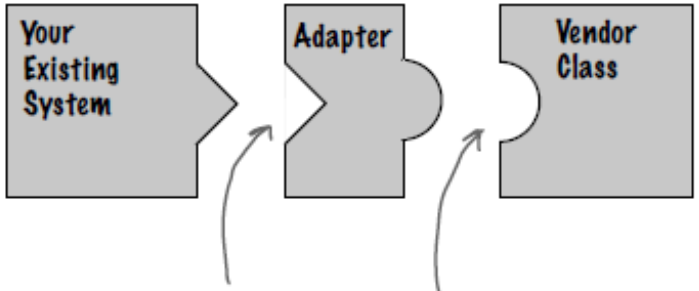
Adapteret ne perditshmeri



Adapteret Object-Oriented

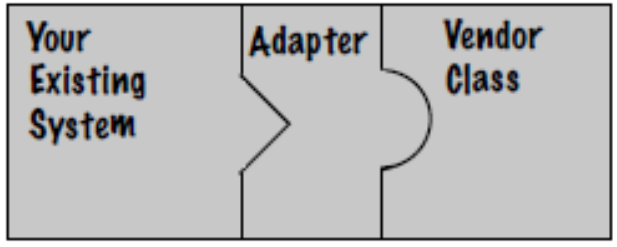


Ky interfejs nuk perputhet me kodin e shkruar per klasen tuaj, andaj nuk do te funksionoj



Adapteri implementon interfejsin qe klasa juaj e pret

Din te komunikoj me interfejsin e pales se trete per t'ju pergjigjur kerkesave



S'ka ndryshim ne kod

Kod i ri

S'ka ndryshim ne kod

Gjeli i detit dëshiron te behet rosë - shembull

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

Nen-klasa MallardDuck

```
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();
}
```

Instancimi i gjelit te detiti (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");
    }
}
```

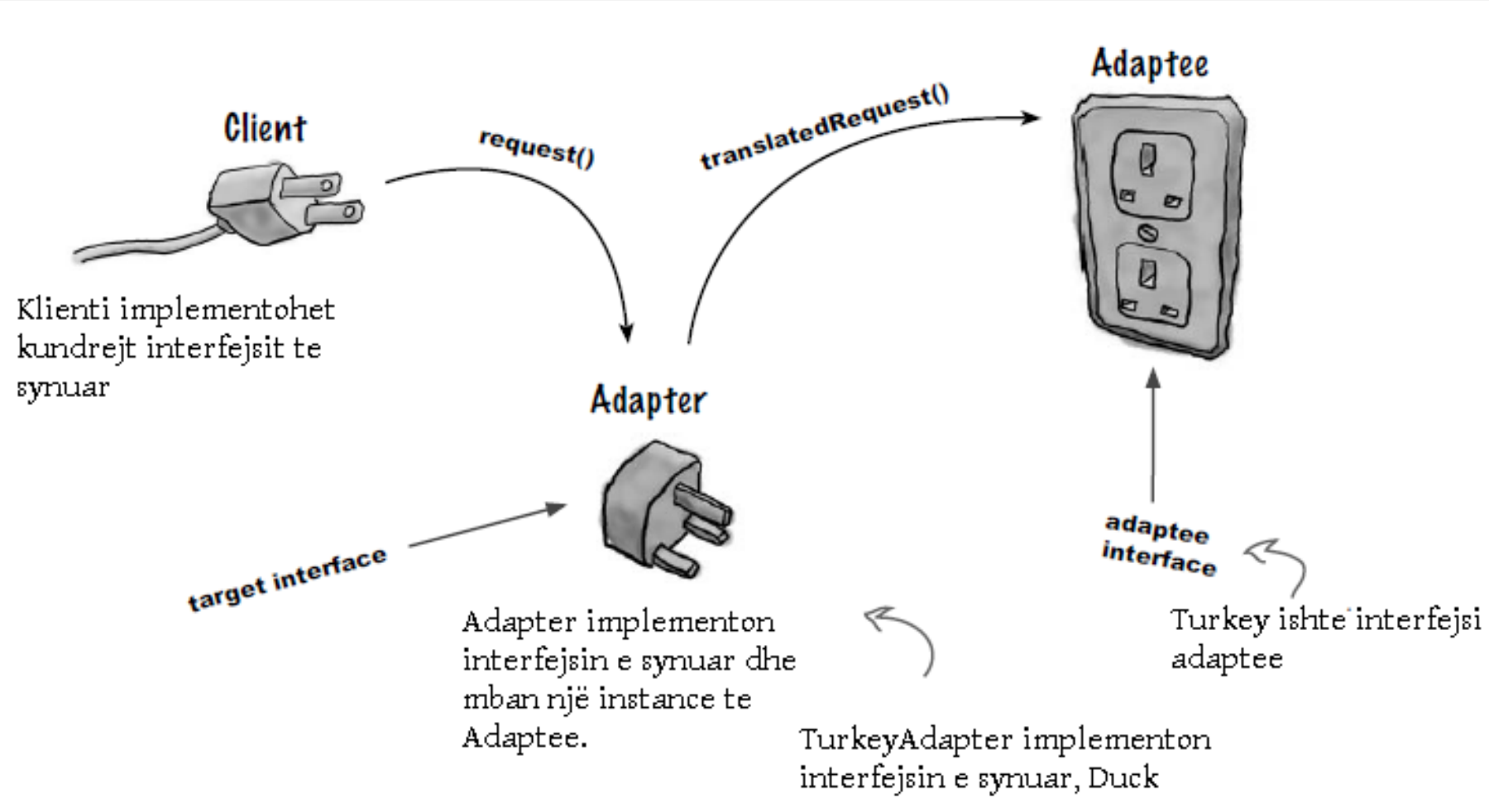
Adapteri Turkey – që e bën një gjeldeti të duket si rosë

```
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();
        }
    }
}
```

Testimi i DuckTestDrive

```
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();
    }
}
```

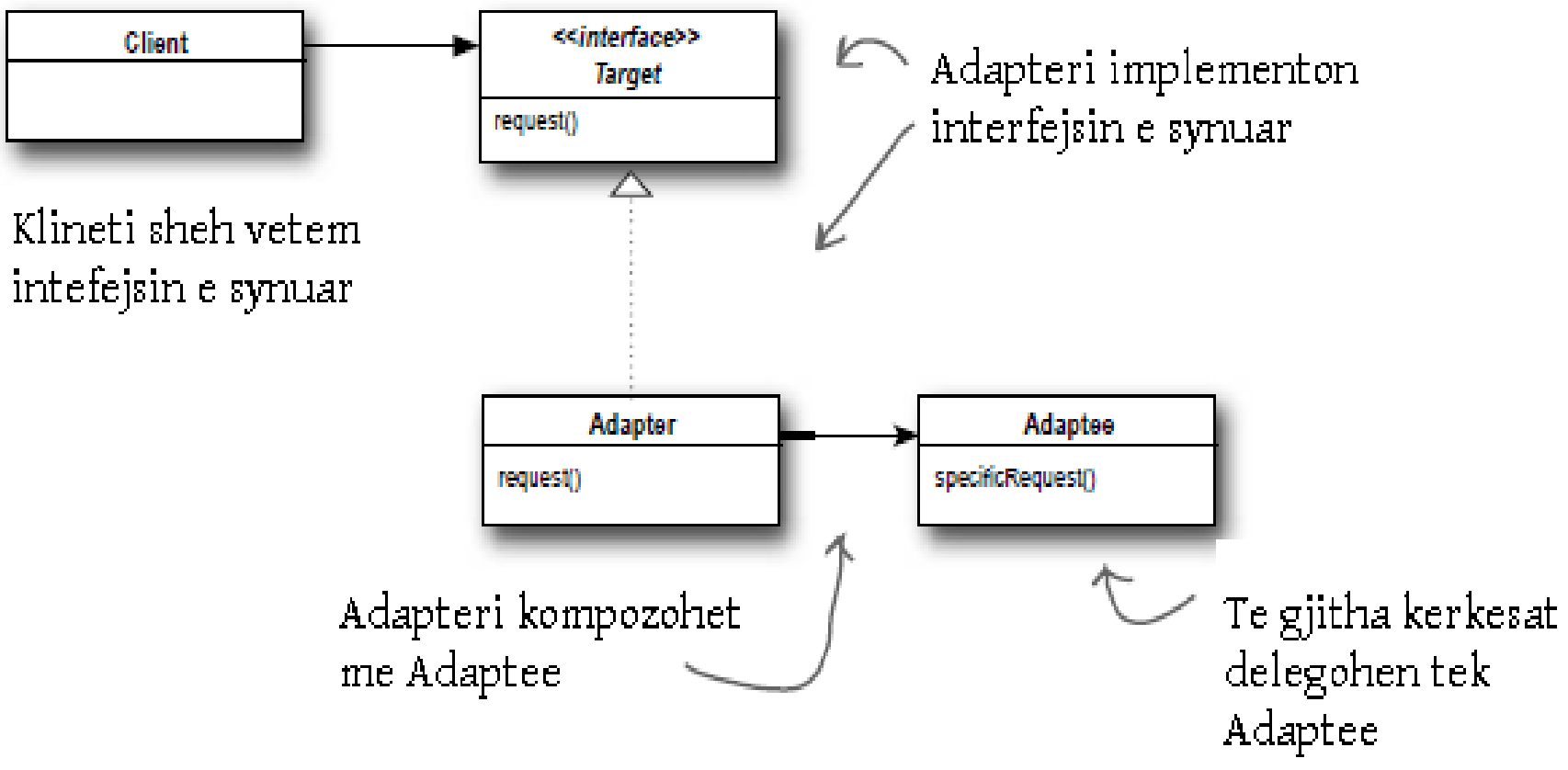
Adapter Pattern explained



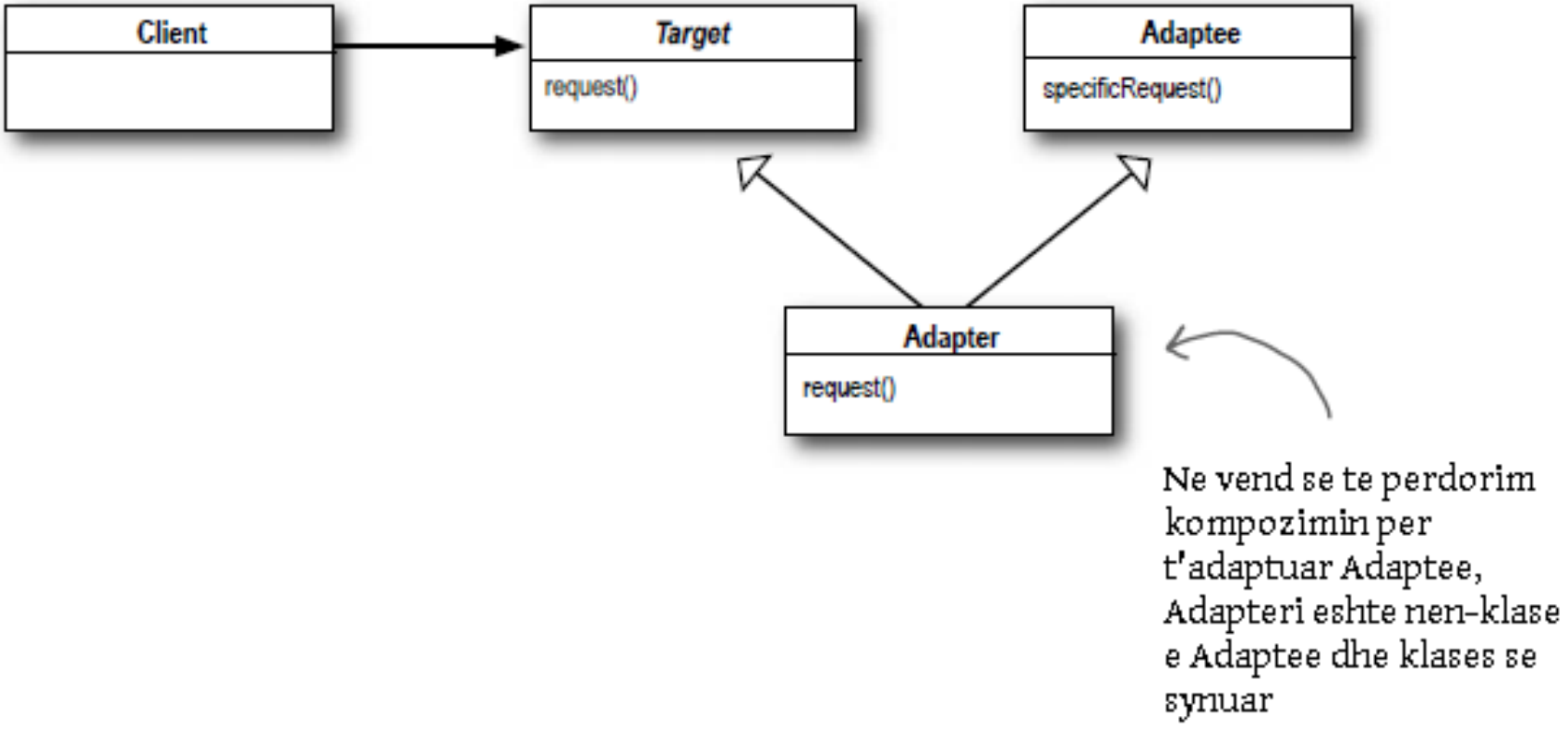
Definicioni i Adapter Pattern

Mostra Adapter konverton interfejsin e nje klase ne interfejsin e klases tjeter qe klienti e pret. Adapteri lejon klasat të punojnë së bashku që per ndryshe s’do te ishte e mundur pershkak te interfejseve qe

Object Adapter



Class adapter



Permbledhje deri më tani

- **OO Principet**
  - Identifikoni aspektet e apalikacionit tuaj te cilat ndyshojne dhe ndani ato nga ato aspekte qe mbesin te pa ndryshuara
  - Programo rreth interfejsit e jo rreth implementimit
  - Favorizo kompozimin kundrejt trashigimise
  - Orvatuni per dizajne me shoqerim te lehte per objektet qe bashkeveprojne
  - Klasat duhet të jenë te hapura për zgjerimin, por të mbyllura për modifikim.

- **OO Mostrat**

Singleton Pattern kufizon instancimin e një klase dhe siguron që vetëm një instance e klasës eshte krijuar.

Strategy Pattern përcakton një familje të algoritmeve, e enkapsulon secilin prej tyre, dhe i bën ata të këmbyeshme. Kjo moster dizajni lejon qe algoritmet te ndryshojnë në mënyrë të pavarur nga klientët që i përdorin ato.

Observer Pattern përcakton një varësi një-me-shumë në mes të objekteve në mënyrë që kur një objekt ndryshon gjendjen, të gjithë vartësit e tij njoftohen dhe azhurohen në mënyrë automatike.

Decorator Pattern shton përgjegjësi ne menyre dinamike për një objekt. Decoratoret ofrojne altrenative fleksibile kundrejt inheritimit për zgjerimin e funksionalitetit.

Vazhdim…

- **OO Principet**
  - Varet nga abstraksionet, e nuk varet nga klasa konkrete.

- **OO Mostrat**
  - Factory method definon nje interfejs per te krijuar nje objekt, por i lejon nen-klasat se cilen klase duhet instancuar. Metoda “Factory” lejon klasen qe te shtyje instancimin tek nen-klasa.
  - Abstract factory pattern ofron një interfejs për të krijuar familjet e objekteve të lidhura ose të varura pa i specifikuar klasat e tyre konkrete.
  - Mostra Adapter konverton interfejsin e nje klase ne interfejsin e klases tjeter qe klienti e pret. Adapteri lejon klasat të punojnë së bashku që per ndryshe s’do te ishte e mundur pershkak te interfejseve qe nuk pershtaten.

Pytje ??