Padrões de Projeto de Software Orientados a Objetos

Tecnologia em Análise e Desenvolvimento de Sistemas

Paulo Mauricio Gonçalves Júnior

Instituto Federal de Educação, Ciência e Tecnologia de Pernambuco

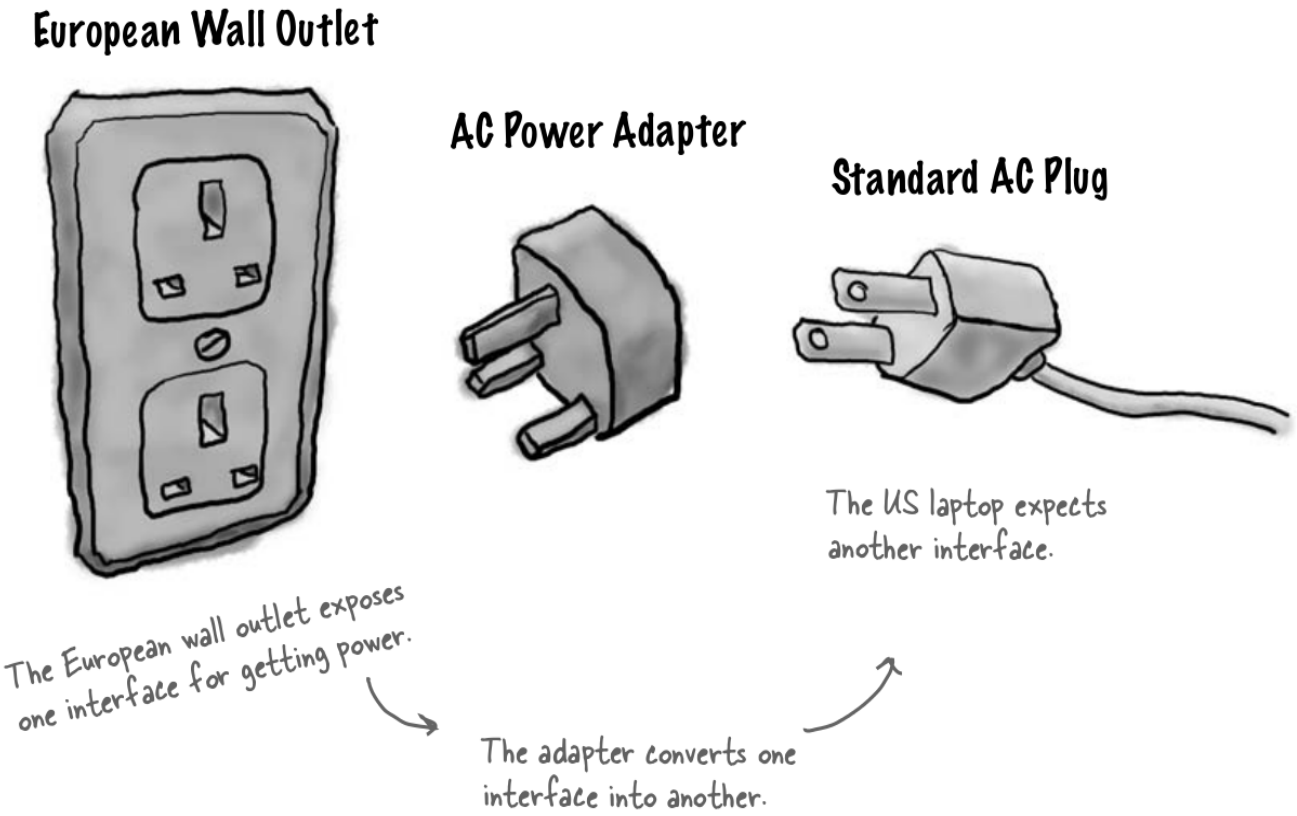
16 de abril de 2018

Parte I

Adapter e Facade

Adapter I

- Vamos encapsular objetos para fazer com que respondam a uma interface diferente.



Adapter II

```
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();
}

public class WildTurkey implements Turkey {
    public void gobble() {
```

Adapter III

```
        System.out.println("Gobble gobble");
    }
    public void fly() {
        System.out.println("I'm flying a short distance");
    }
}

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();
        }
    }
}
```

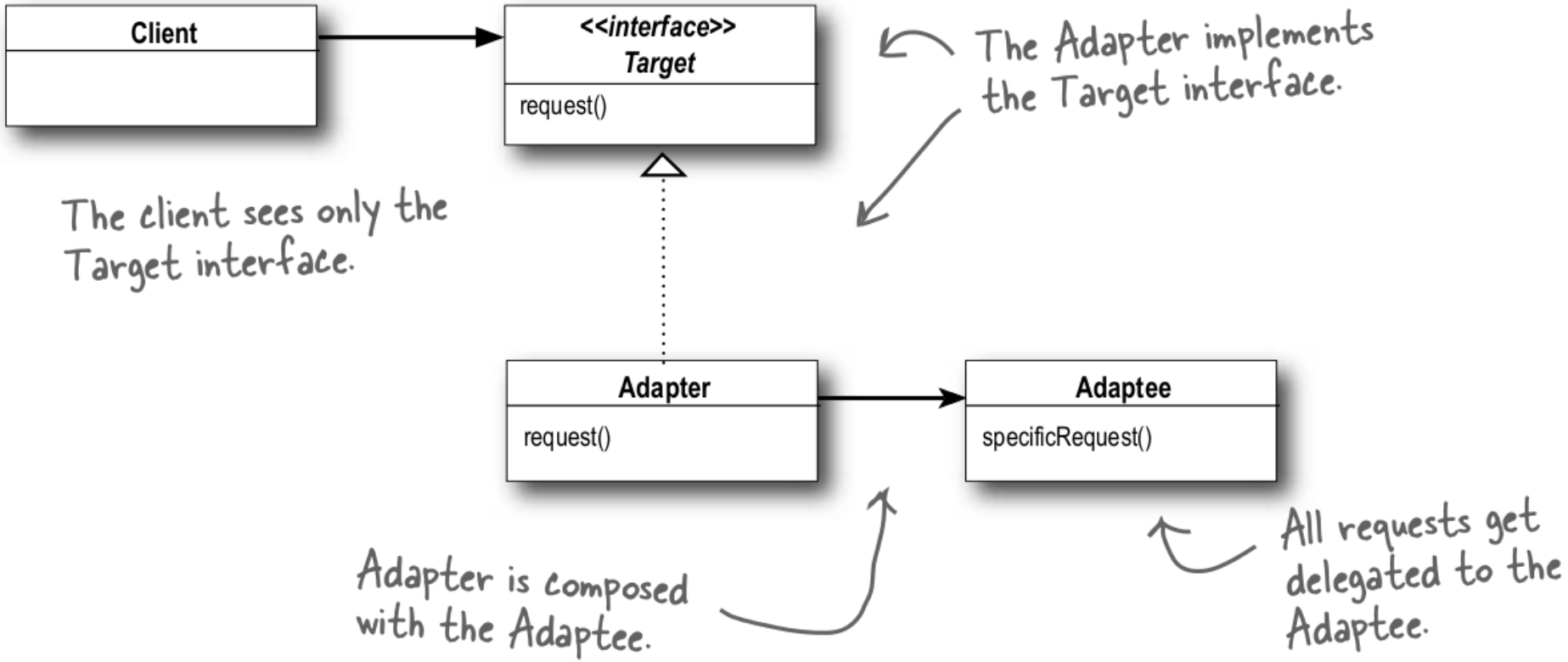
Adapter IV

1. O cliente faz uma solicitação para o adaptador chamando um método nele usando a interface alvo.
2. O adaptador traduz a solicitação em uma ou mais chamadas na classe adaptada usando a interface da classe adaptada.
3. O cliente recebe o resultado da chamada e nunca sabe que existe um adaptador fazendo a tradução.

Definição

O padrão Adapter converte a interface de uma classe em outra interface que o cliente conhece. Adapter permite a classes trabalhar em conjunto que a priori não poderiam devido a suas interfaces incompatíveis.

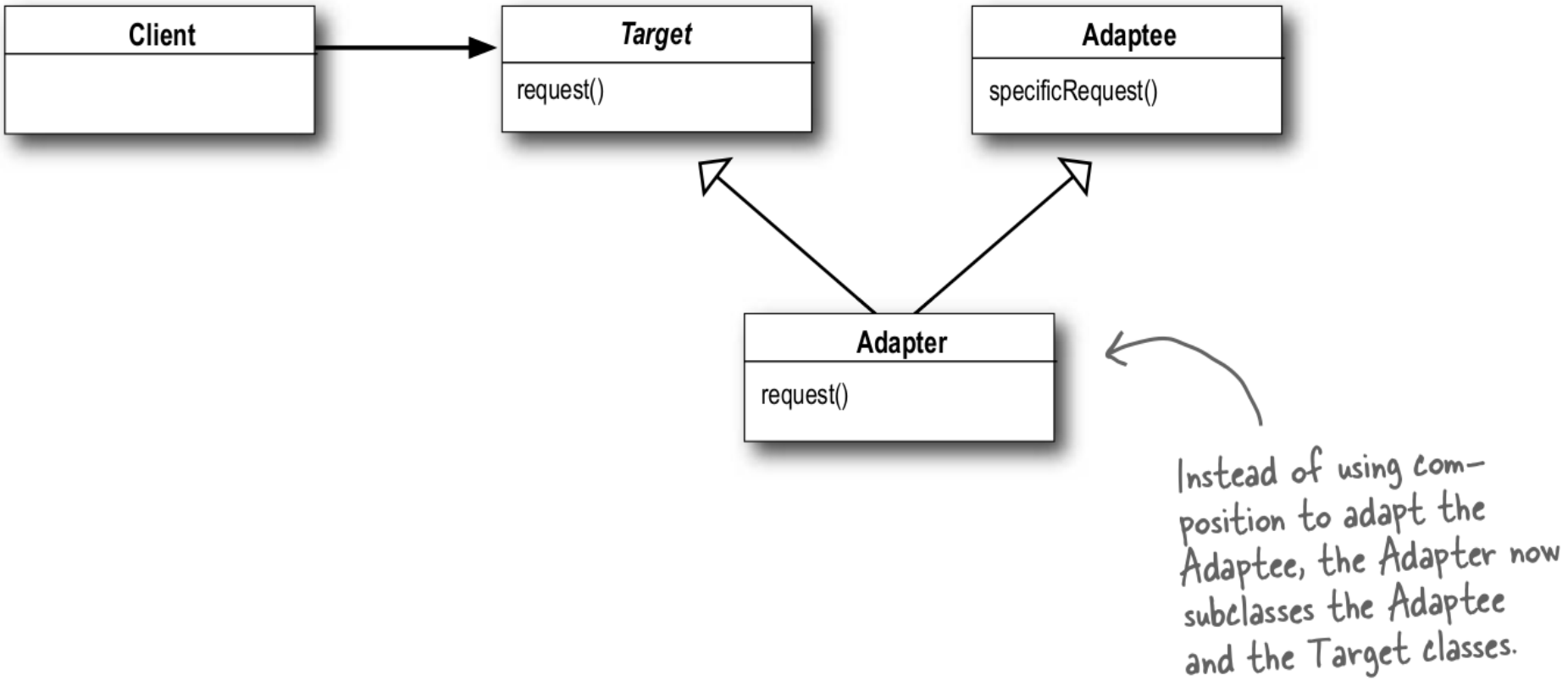
Adapter V



- Na verdade existem duas formas de implementação do padrão Adapter: com objetos ou classes.

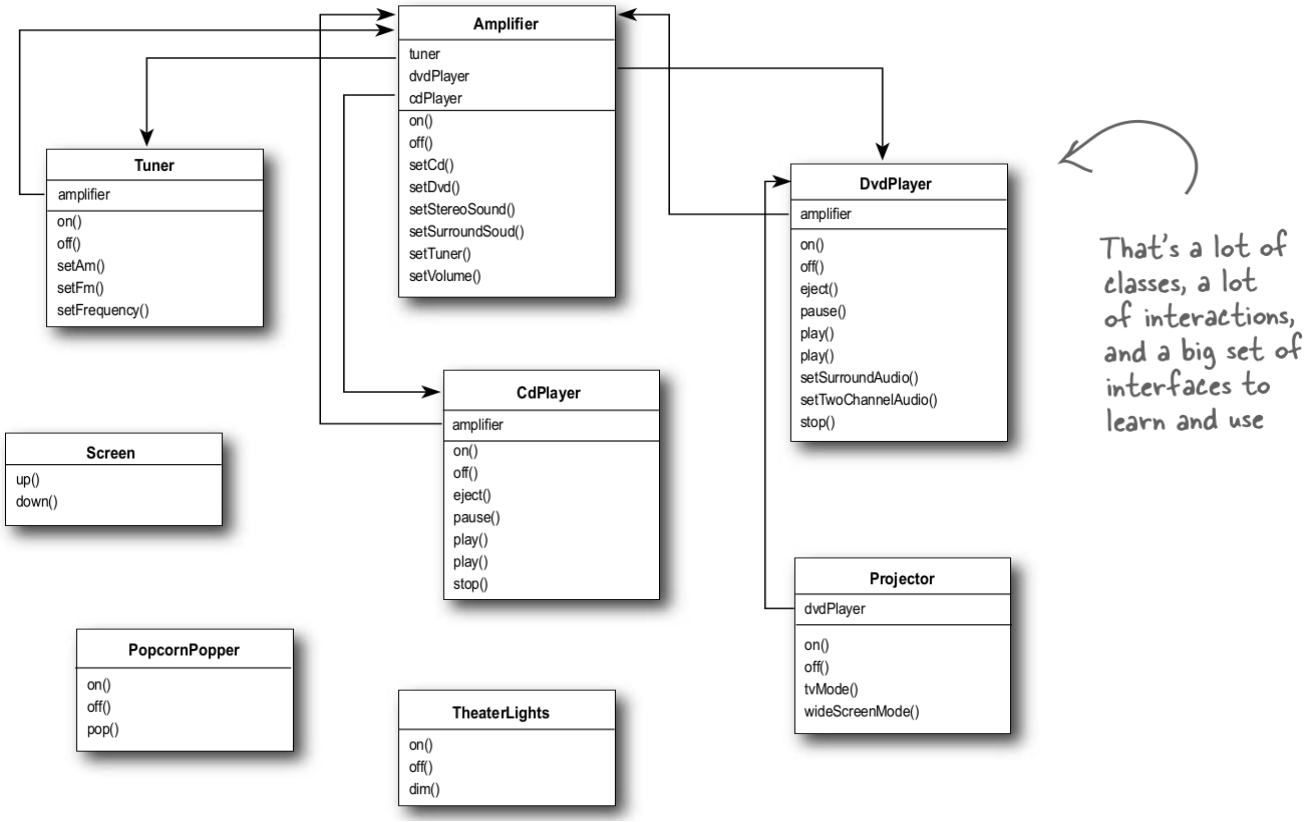
Adapter VI

- Vimos a implementação com objetos. Com classes, precisamos de herança múltipla, o que Java não permite.



Facade I

- Fachada serve para ofuscar a complexidade de uma ou mais classes, usando uma interface simplificada.
- Imagine um sistema de home theather.



Facade II

1. Ligar a máquina de fazer pipoca
2. Começar a fazer pipoca
3. Baixar as luzes
4. Baixar a tela de projeção
5. Ligar o projetor
6. Setar a entrada do projetor para modo DVD
7. Colocar o projetos em modo tela cheia
8. Ligar o amplificador do som
9. Setar a entrada do amplificador para DVD
10. Setar o amplificador para som surround
11. Setar o volume do amplificador para médio (5)
12. Ligar o DVD
13. Começar o filme

Facade III

```
popper.on();
popper.pop();
lights.dim(10);
screen.down();
projector.on();
projector.setInput(dvd);
projector.wideScreenMode();
amp.on();
amp.setDvd(dvd);
amp.setSurroundSound();
amp.setVolume(5);
dvd.on();
dvd.play(movie);

public class HomeTheaterFacade {
  Amplifier amp;
  Tuner tuner;
  DvdPlayer dvd;
  CdPlayer cd;
  Projector projector;
  TheaterLights lights;
```

Facade IV

```
Screen screen;
PopcornPopper popper;
public HomeTheaterFacade(Amplifier amp, Tuner tuner, DvdPlayer
    dvd, CdPlayer cd, Projector projector, Screen screen,
    TheaterLights lights, PopcornPopper popper) {
  this.amp = amp;
  this.tuner = tuner;
  this.dvd = dvd;
  this.cd = cd;
  this.projector = projector;
  this.screen = screen;
  this.lights = lights;
  this.popper = popper;
}
public void watchMovie(String movie) {
  System.out.println("Get ready to watch a movie...");
  popper.on();
  popper.pop();
  sequence
  lights.dim(10);
  screen.down();
```

Facade V

```
        projector.on();
        amp.on();
        amp.setDvd(dvd);
        amp.setSurroundSound();
        amp.setVolume(5);
        dvd.on();
        dvd.play(movie);
    }
    public void endMovie() {
        System.out.println("Shutting movie theater down...");
        popper.off();
        lights.on();
        screen.up();
        projector.off();
        amp.off();
        dvd.stop();
        dvd.eject();
        dvd.off();
    }
}
```

Facade VI

```
public class HomeTheaterTestDrive {
  public static void main(String[] args) {
    // instantiate components here
    HomeTheaterFacade homeTheater =
    new HomeTheaterFacade(amp, tuner, dvd, cd, projector, screen,
        lights, popper);
    homeTheater.watchMovie("Raiders of the Lost Ark");
    homeTheater.endMovie();
  }
}
```

Definição

O padrão Facade provê uma interface unificada para um conjunto de interfaces de um subsistema. Facade define uma interface de alto nível que torna mais fácil o uso do subsistema.

Facade VII

Princípio de Projeto

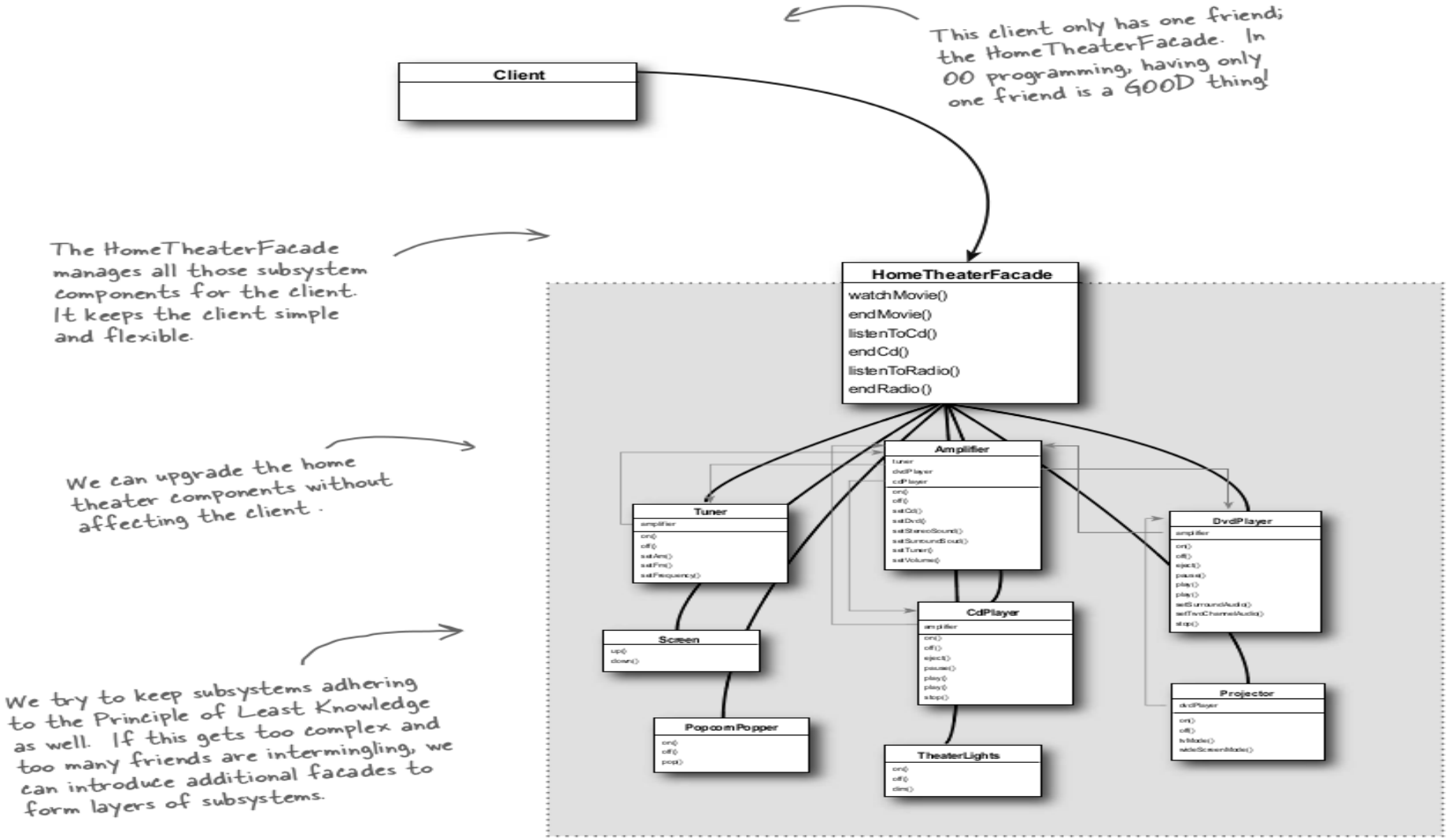
Princípio do Menor Conhecimento – fale somente com seus amigos imediatos.

- O princípio nos informa que só deveríamos invocar métodos que pertencem ao:
  - Próprio objeto.
  - Objetos passados como parâmetros para o métodos
  - Qualquer objeto que o método cria ou instancia.
  - Quaisquer componentes do objetos.

Facade VIII

```
public class Car {
  Engine engine;
  // other instance variables
  public Car() {
    // initialize engine, etc.
  }
  public void start(Key key) {
    Doors doors = new Doors();
    boolean authorized = key.turns();
    if (authorized) {
      engine.start();
      updateDashboardDisplay();
      doors.lock();
    }
  }
  public void updateDashboardDisplay() {
    // update display
  }
}
```

Facade IX


This client only has one friend; the HomeTheaterFacade. In OO programming, having only one friend is a GOOD thing!
Client
The HomeTheaterFacade manages all those subsystem components for the client. It keeps the client simple and flexible.
HomeTheaterFacade
watchMovie()
endMovie()
listenToCd()
endCd()
listenToRadio()
endRadio()
We can upgrade the home theater components without affecting the client.
Amplifier
Tuner
DvdPlayer
CdPlayer
Screen
Projector
PopcornPopper
TheaterLights
We try to keep subsystems adhering to the Principle of Least Knowledge as well. If this gets too complex and too many friends are intermingling, we can introduce additional facades to form layers of subsystems.