

Computação Orientada a Objetos

Padrões de Projeto Abstract Factory e Façade

Slides baseados em:

- E. Gamma and R. Helm and R. Johnson and J. Vlissides. Design Patterns - Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.
- E. Freeman and E. Freeman. Padrões de Projetos. Use a cabeça. Alta Books Editora. 2009.
- Slides Prof. Christian Dannel Paz Trillo
- Slides Profa. Patrícia R. Oliveira

1

Profa. Karina Valdivia Delgado
EACH-USP

PADRÃO ABSTRACT FACTORY

- Padrão de Criação
- **Objetivo:**
 - prover uma interface para criação de famílias de objetos sem especificar sua classe concreta.

PADRÃO ABSTRACT FACTORY:

Motivação

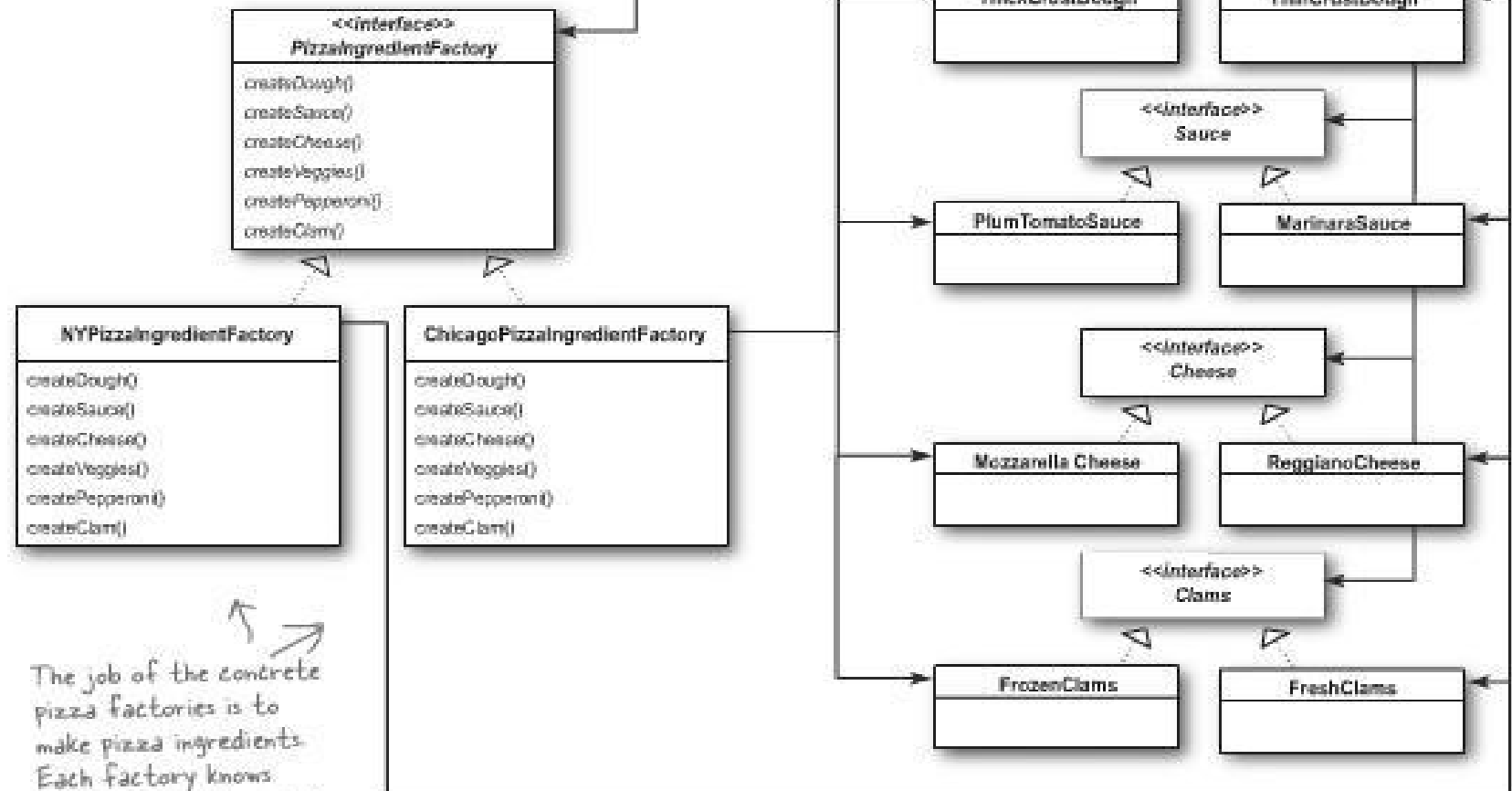
- Considere uma aplicação para criar uma fábrica de ingredientes para todas as franquias de pizzas
- Os ingredientes das pizzas são essencialmente os mesmos em cada região, mas existem variações específicas das diferentes regiões.
 - Ex: cheiro verde na região sudeste tem salsa no lugar do coentro
 - Ex: o marisco em São Luis é diferente do marisco em São Paulo
- Cada região (e as franquias regionais) usa uma família de ingredientes específico

PADRÃO ABSTRACT FACTORY:

Motivação

- É necessário lidar com essas diferentes famílias de ingredientes.
- A fábrica será responsável pela criação de famílias de ingredientes
- Não se deve permitir misturar ingredientes de franquias diferentes.

The abstract PizzaIngredientFactory is the interface that defines how to make a family of related products - everything we need to make a pizza

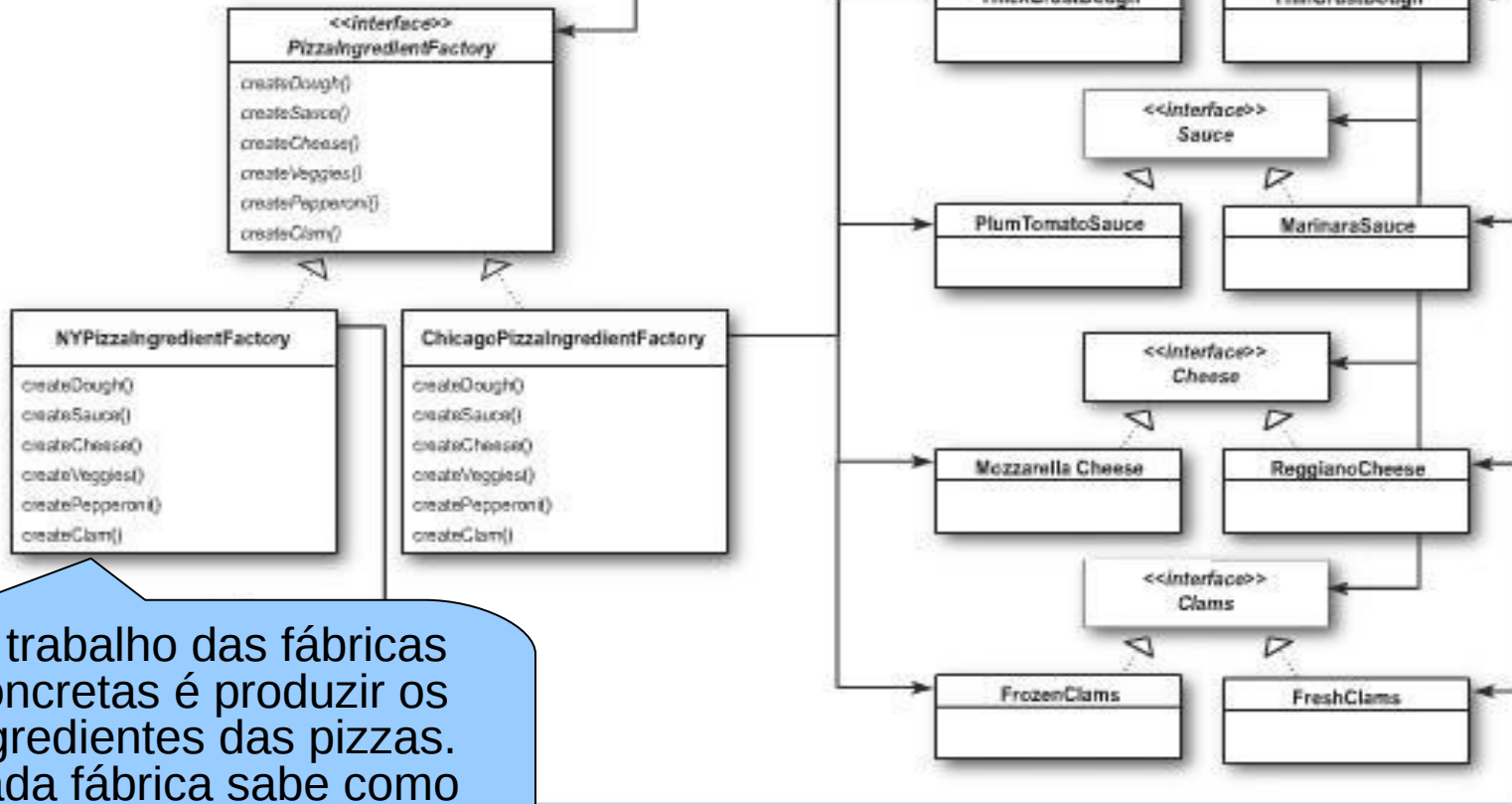


The job of the concrete pizza factories is to make pizza ingredients. Each factory knows how to create the right objects for their region.

Each factory produces a different implementation for the family of products.

A PizzIngredientFactory é a interface que define como criar uma família de produtos relacionados

Um dos clientes de AbstractFactory é a instância de NYPizzaStore



O trabalho das fábricas concretas é produzir os ingredientes das pizzas. Cada fábrica sabe como criar os objetos certos para sua região

Each factory produces a different implementation for the family of products

```
public class NYPizzaIngredientFactory implements
    PizzaIngredientFactory{

    public Dough createDough(){
        return new ThinCrustDough();
    }

    public Sauce createSauce(){
        return new MarinaSauce();
    }
    public Cheese createCheese(){
        return new ReggianoCheese();
    }
    public Veggies[] createVeggies(){
        Veggies veggies[] = {new Garlic(), new Onion()}
        return veggies;
    }
    ...
}
```

```
public class NYStylePizzaStore extends PizzaStore{
    protected Pizza createPizza(){
        PizzaIngredientFactory ingredientFactory=
            new NYPizzaIngredientFactory();
        Pizza pizza=new Pizza(ingredientFactory);
        pizza.setName("Pizza no estilo de Nova York");
        pizza.prepare();
        return pizza;
    }
}

public class Pizza{
    String name;
    Dough dough;
    Sauce sauce;
    Veggies veggies[];
    Cheese cheese;
    Pepperoni pepperoni;
    Clam clam;
    PizzaIngredientFactory ingredientFactory;
    public Pizza(PizzaIngredientFactory ingredientFactory){
        this.ingredientFactory=ingredientFactory;
    }
    public prepare(){
        System.out.println("Preparing "+name);
        dough= ingredientFactory.createDough();
        sauce= ingredientFactory.createSauce();
        ... }
    }
}
```

toda vez que precisa
de um ingrediente,
pede para a fábrica
produzi-lo.

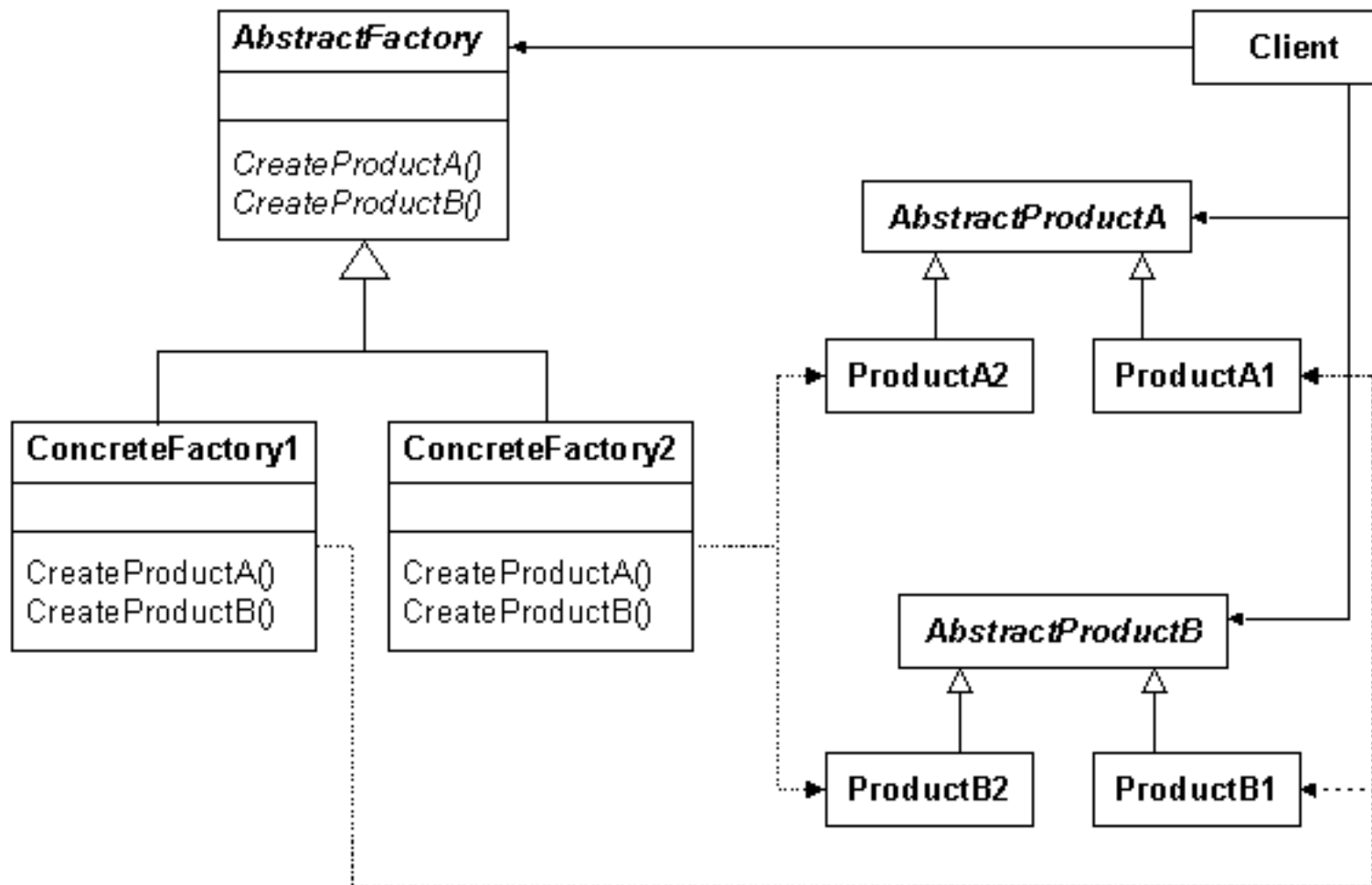
PADRÃO ABSTRACT FACTORY

○ Aplicabilidade:

- Use uma fábrica abstrata quando:
 - um sistema deve ser independente da forma como seus produtos são criados e representados;
 - um sistema deve lidar com uma família de vários produtos diferentes;
 - você quer prover uma biblioteca de classes de produtos mas não quer revelar as suas implementações, quer revelar apenas suas interfaces.

PADRÃO ABSTRACT FACTORY

○ Estrutura



PADRÃO ABSTRACT FACTORY

○ Participantes:

- AbstractFactory:
 - Define quais produtos serão “produzidos” pela fábrica.
- ConcreteFactory
 - Implementa de fato a criação dos produtos.
- AbstractProduct
 - Define os contratos (métodos) que os produtos da fábrica deverão cumprir.
- ConcreteProduct
 - Implementa os contratos de cada produto.
- Client
 - Utiliza apenas as interfaces AbstractFactory e AbstractProduct.

PADRÃO ABSTRACT FACTORY

○ Colaborações:

- Normalmente, apenas uma instância de ConcreteFactory é criada em tempo de execução.
- Esta instância cria objetos através das classes ConcreteProduct correspondentes a uma família de produtos.
- Uma AbstractFactory deixa a criação de objetos para as suas subclasses ConcreteFactory.

PADRÃO ABSTRACT FACTORY

○ Consequências

- Isola as classes concretas dos clientes;
- Facilita a troca de famílias de produtos (basta trocar uma linha do código pois a criação da fábrica concreta aparece em um único ponto do programa);
- Promove a consistência de produtos (não há o perigo de misturar objetos de famílias diferentes);
- Dificulta a criação de novos produtos ligeiramente diferentes (pois temos que modificar a fábrica abstrata e todas as fábricas concretas).

PADRÃO ABSTRACT FACTORY

- Implementação:

- Na fábrica abstrata, cria-se um método fábrica para cada tipo de produto. Cada fábrica concreta implementa o código que cria os objetos de fato.

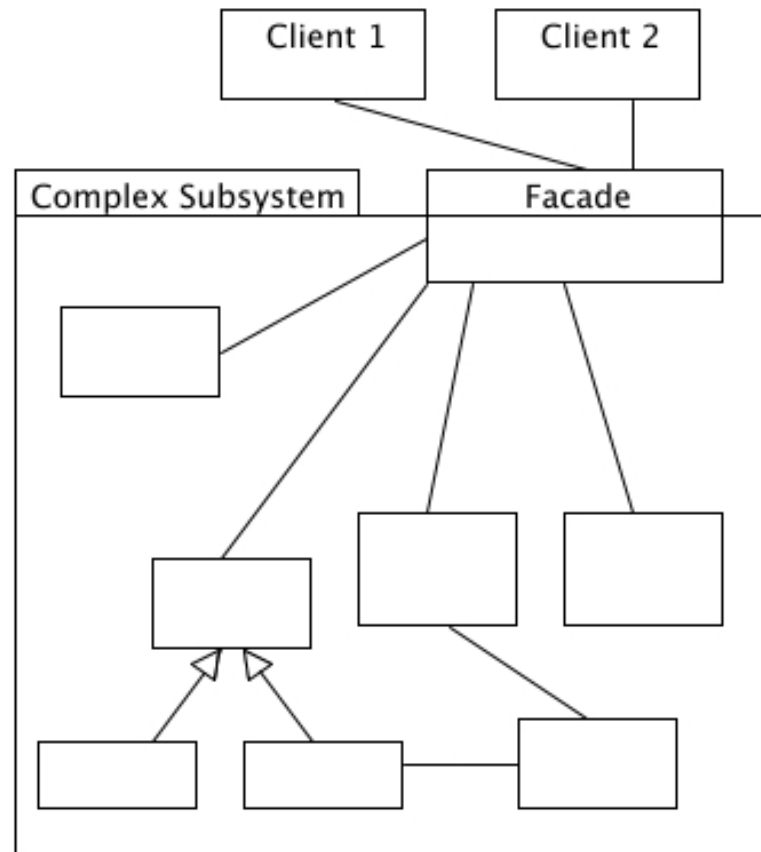
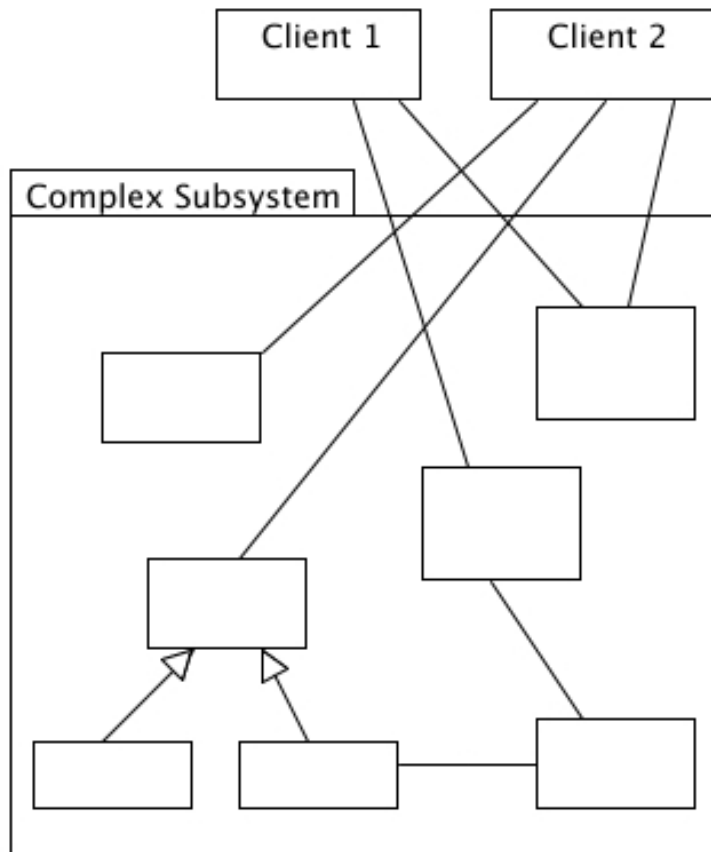


PADRÃO FAÇADE

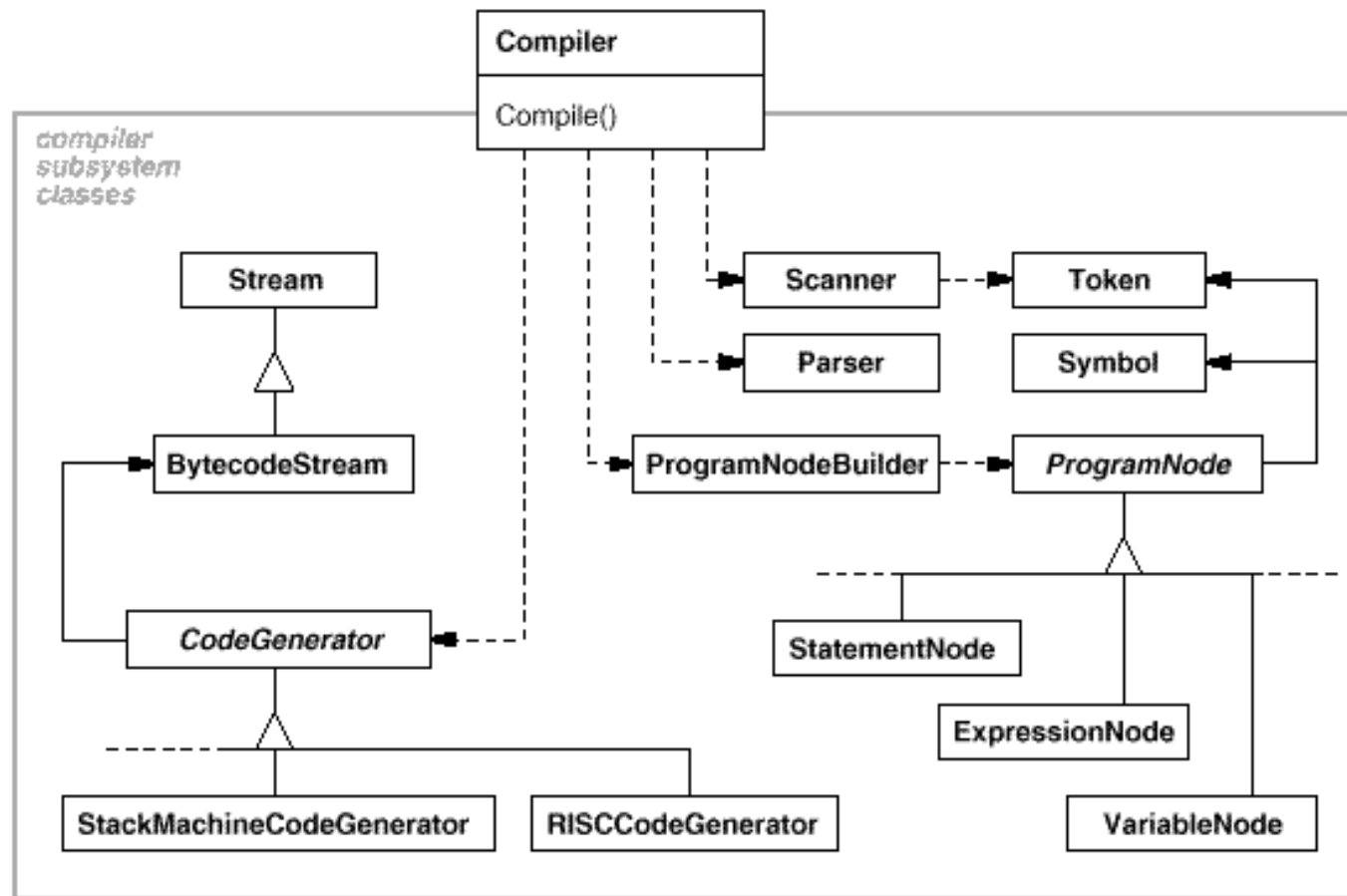
PADRÃO FAÇADE

- Padrão Estrutural.
- **Objetivo:**
 - Oferecer uma **interface única** para um conjunto de interfaces de um subsistema.
 - Façade define uma interface de **mais alto nível** que torna mais fácil o uso do subsistema.
- **Motivação:**
 - Implementação com **baixo acoplamento** de interação entre subsistemas.

PADRÃO FAÇADE

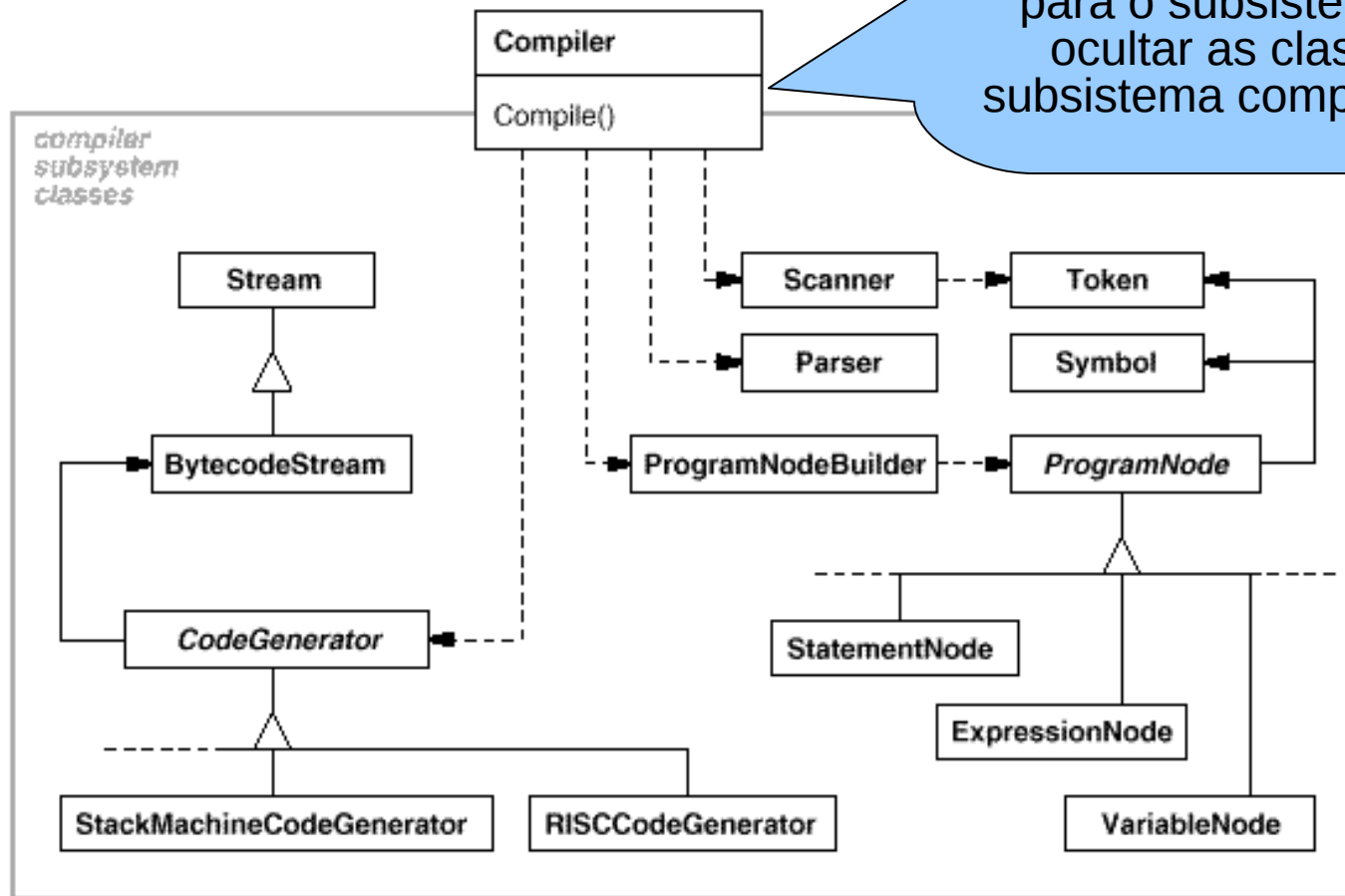


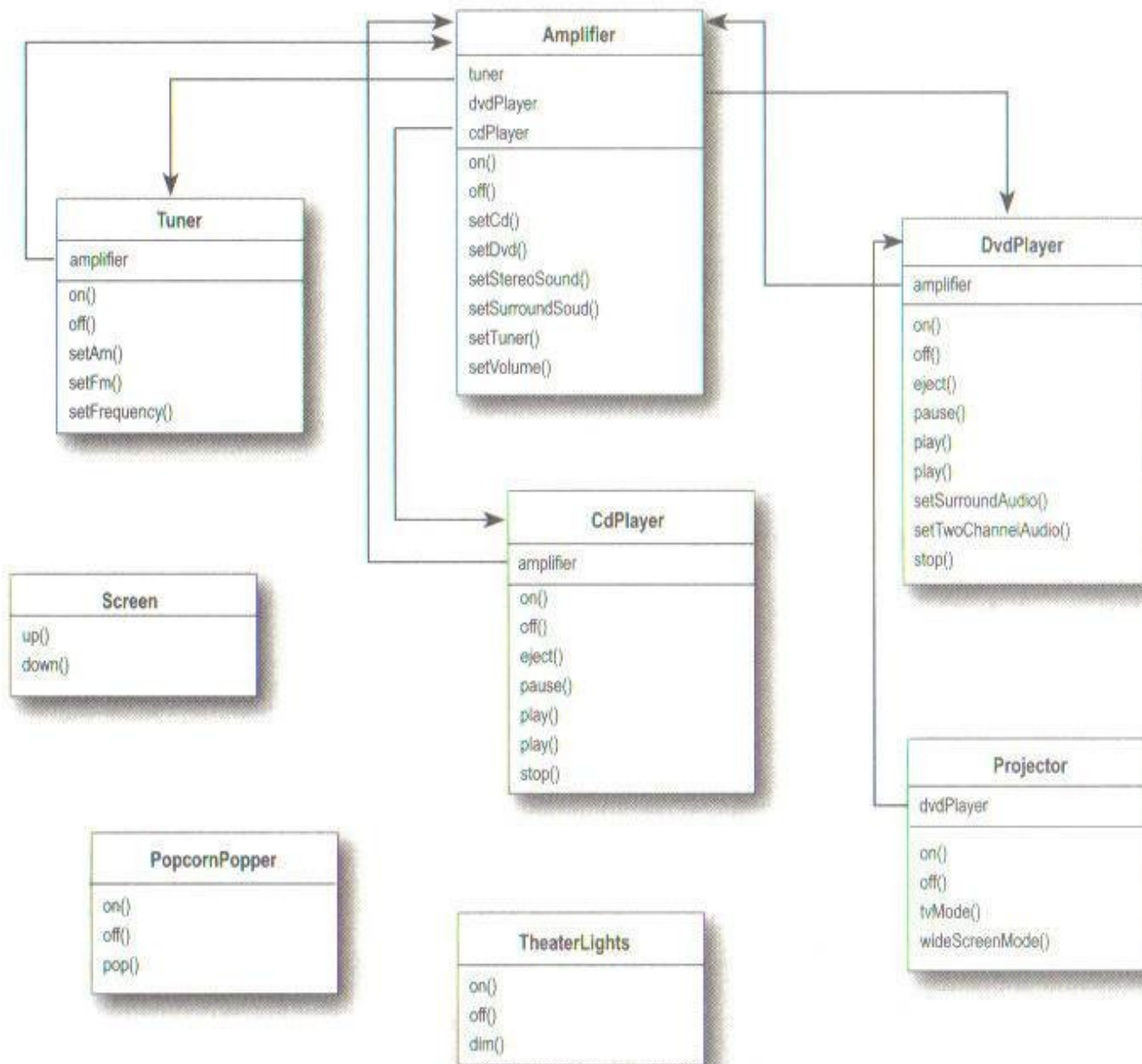
Exemplo – compilador



Exemplo – compila

A classe compiler funciona como uma fachada, oferecendo aos clientes uma interface única e simples para o subsistema, sem ocultar as classes do subsistema completamente





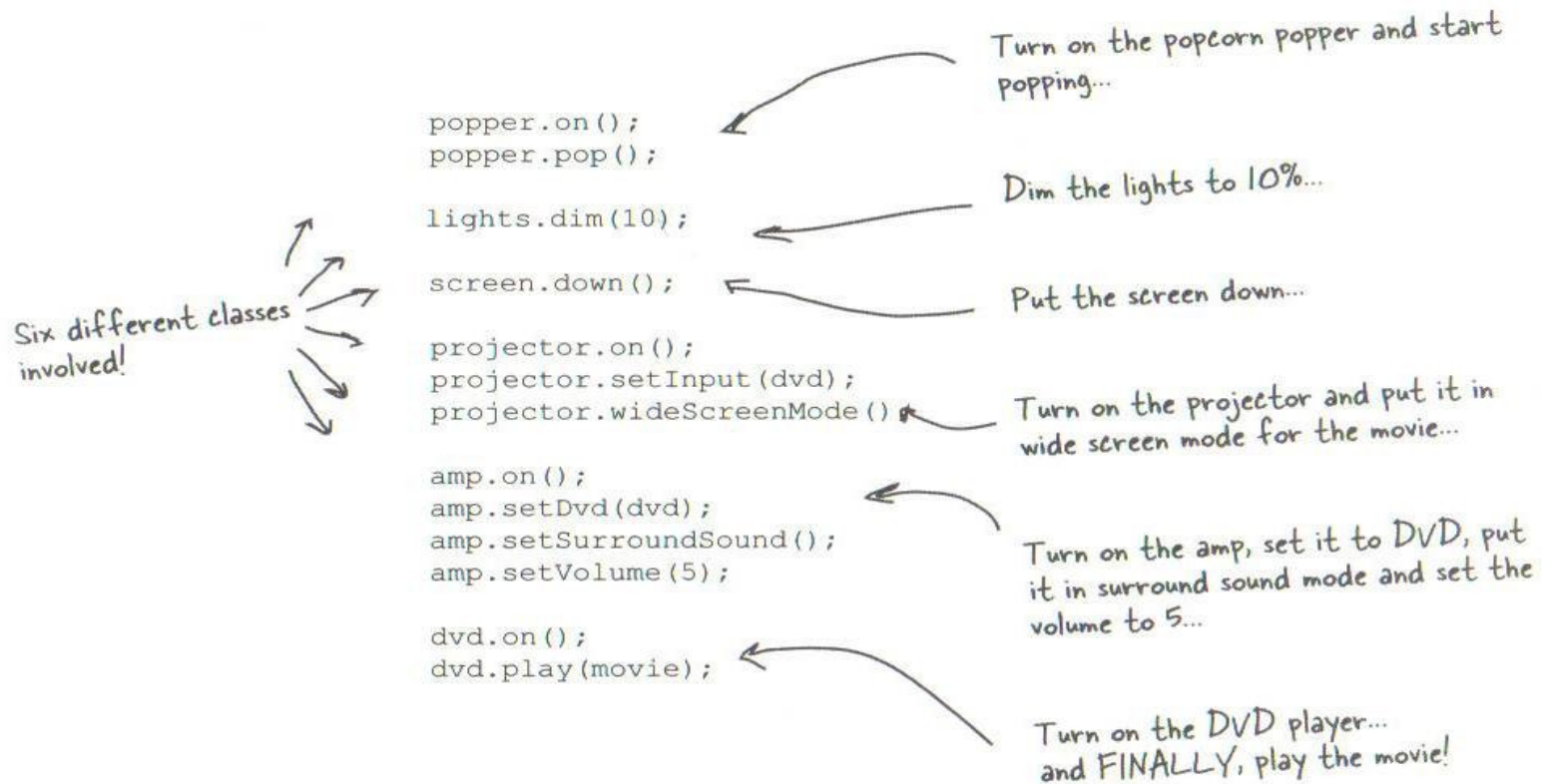
Temos várias classes, várias interações e um enorme conjunto de interfaces para aprender a usar

Assistindo a um filme da maneira mais difícil

- 1 Turn on the popcorn popper
- 2 Start the popper popping
- 3 Dim the lights
- 4 Put the screen down
- 5 Turn the projector on
- 6 Set the projector input to DVD
- 7 Put the projector on wide-screen mode
- 8 Turn the sound amplifier on
- 9 Set the amplifier to DVD input
- 10 Set the amplifier to surround sound
- 11 Set the amplifier volume to medium (5)
- 12 Turn the DVD Player on
- 13 Start the DVD Player playing



Classes e chamadas de métodos necessárias para assistir o filme

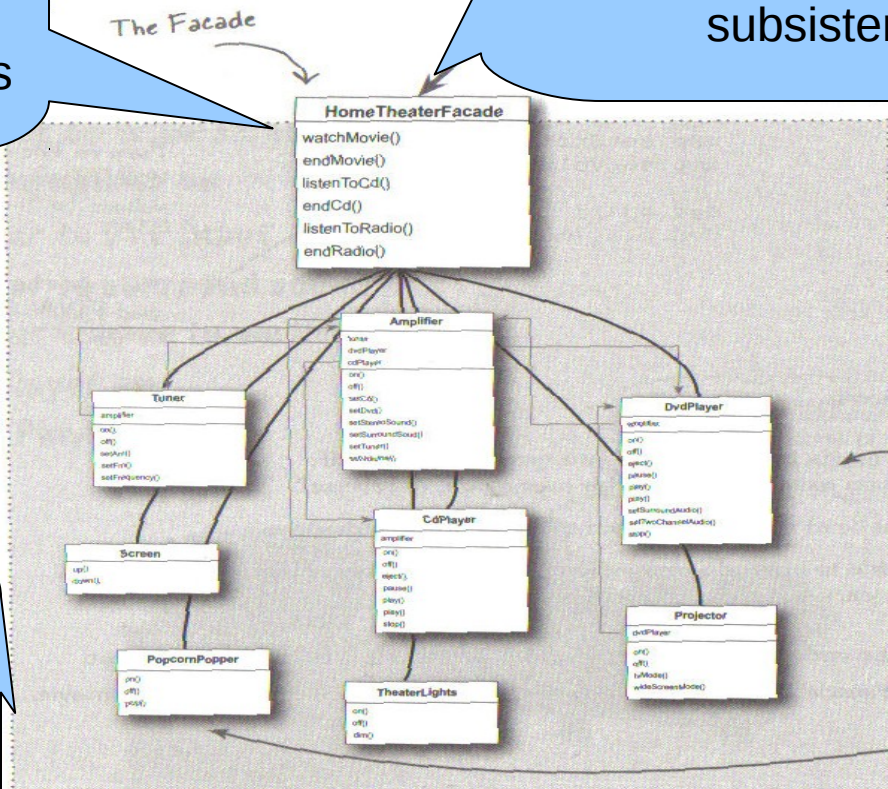


Criamos uma nova classe, com alguns métodos simples

Agora o cliente chama métodos da Fachada e não os métodos das classes do subsistema.

A tarefa do cliente foi simplificada. Cliente feliz!

Subsistema que a Fachada está simplificando



A Fachada preserva o acesso direto ao subsistema



CONSTRUINDO A FACHADA

```
public class HomeTheaterFacade  
{
```

```
    Amplifier amp;  
    Tuner tuner;  
    DvdPlayer dvd;  
    CdPlayer cd;  
    Projector projector;  
    TheaterLights lights;  
    Screen screen;  
    PopcornPopper popper;  
    // construtor ...
```

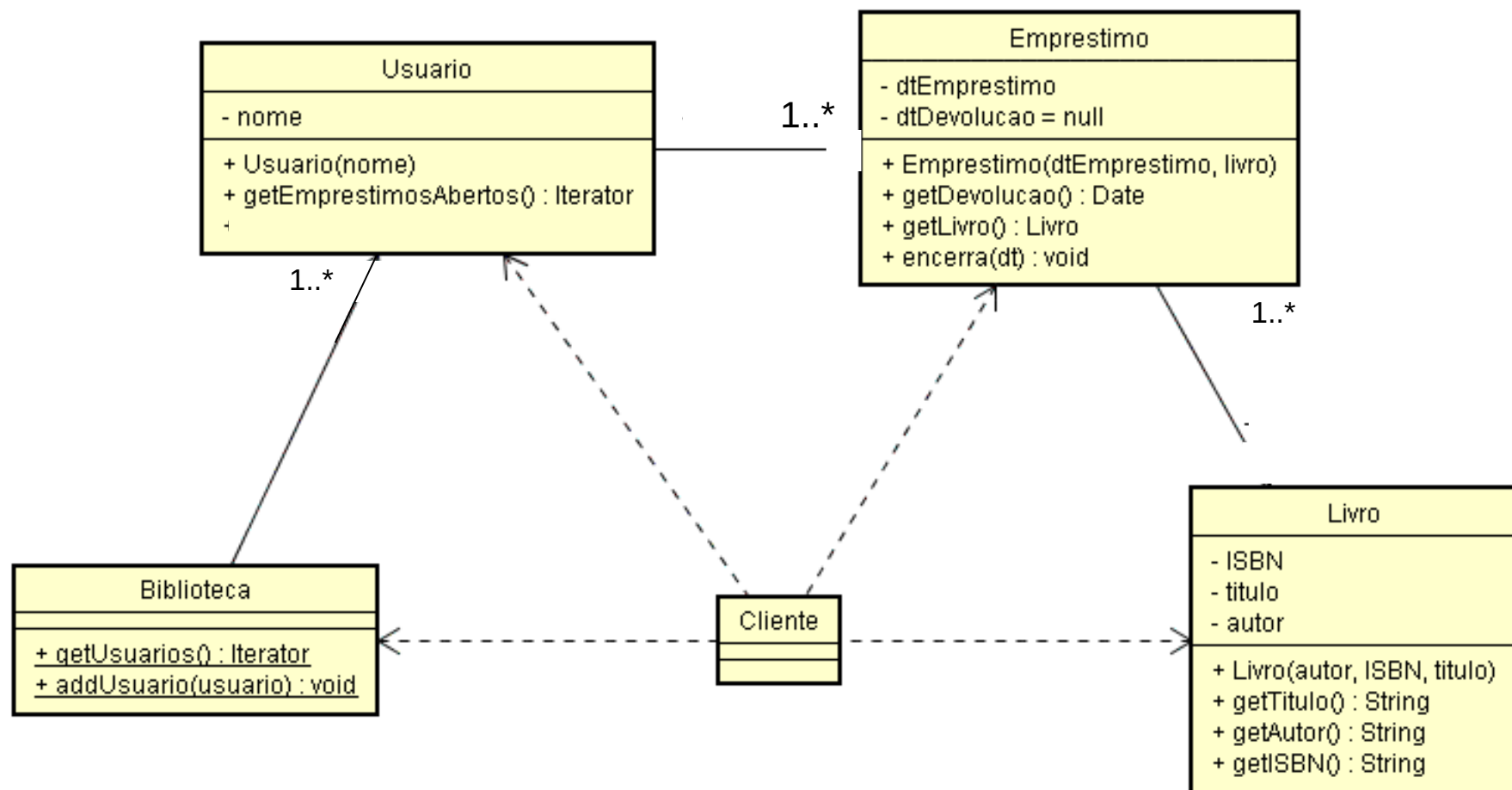
Todos estes são componentes
do subsistema

```
    public void watchMovie(String movie){  
        popper.on();  
        popper.pop();  
        lights.dim(10);  
        screen.down();  
        projector.on();  
  
        ...
```

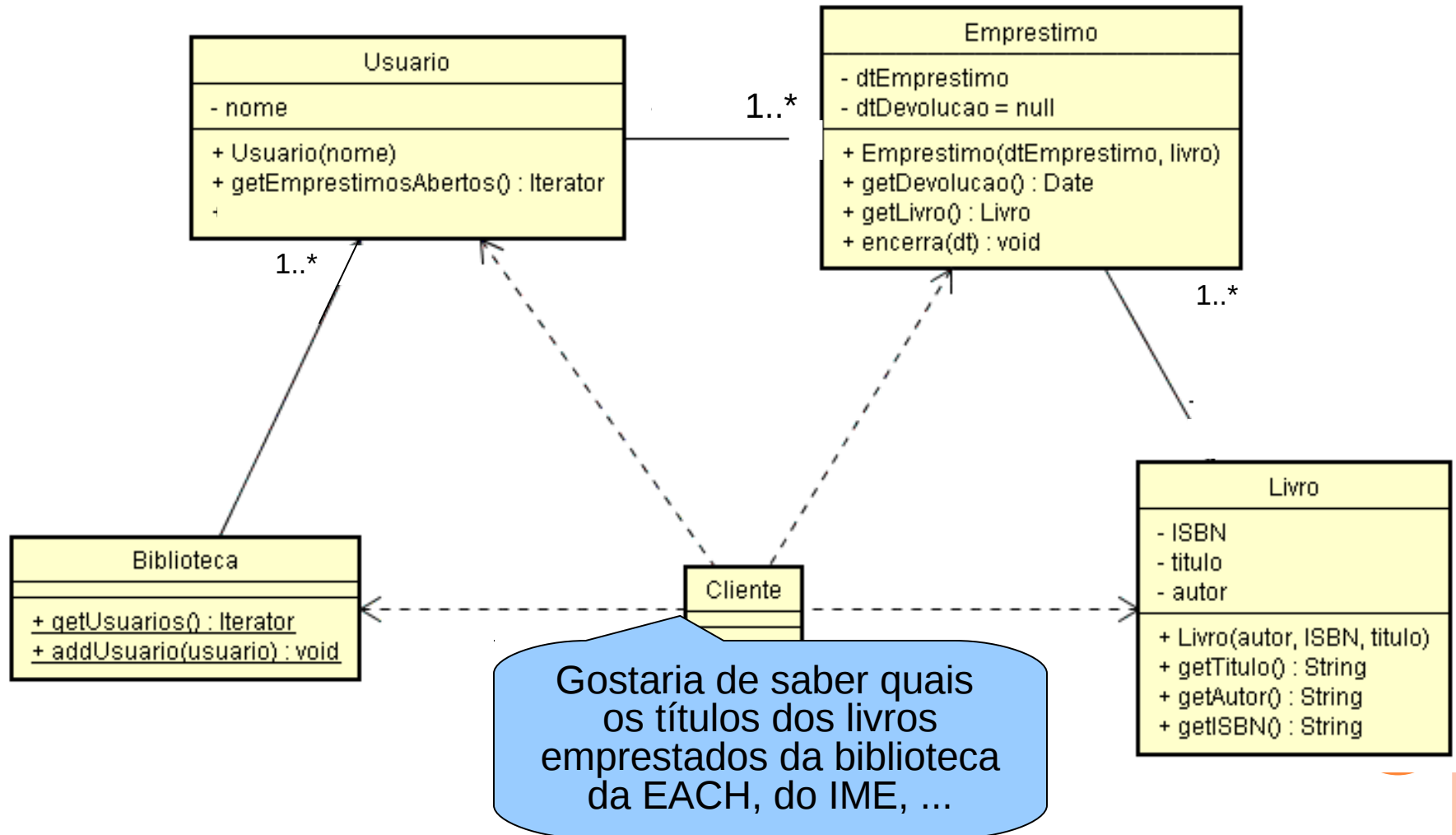
delegação

```
    }  
}
```

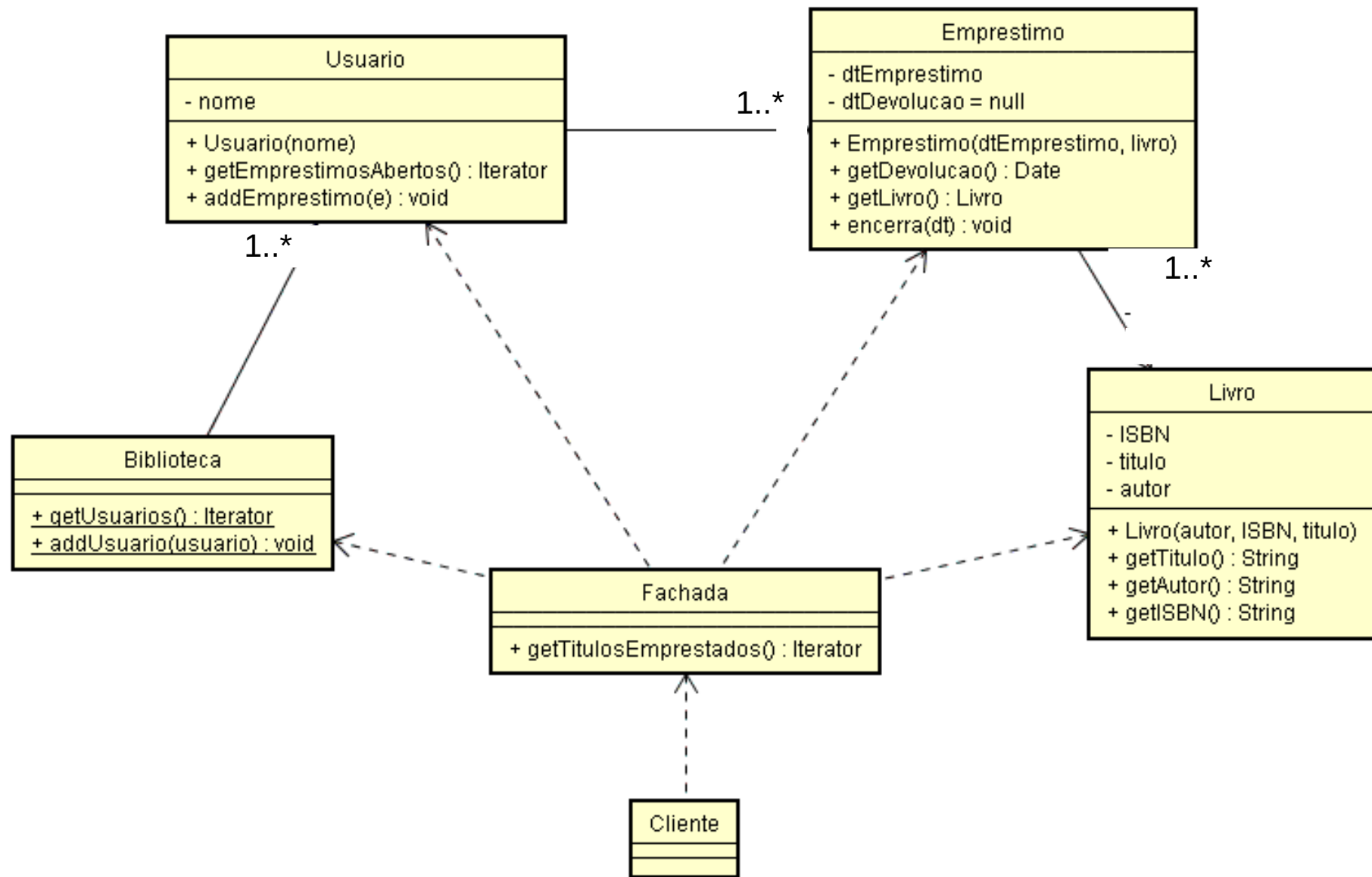

Exemplo Biblioteca: Uma implementação pobre



Exemplo Biblioteca: Uma implementação pobre



Exemplo Biblioteca: Uma implementação melhor



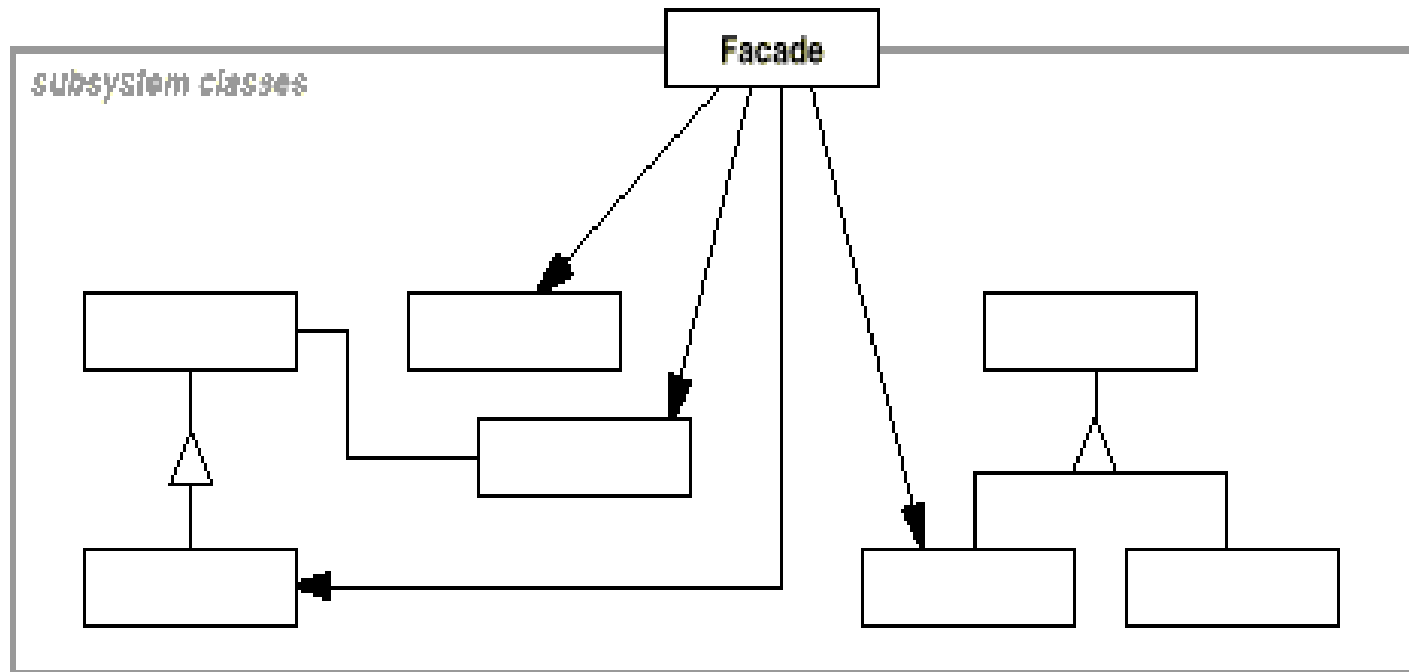
PADRÃO FAÇADE

○ Aplicabilidade:

- Oferecer uma interface simples para um subsistema complexo.
- Quando houver muitas interações com classes internas de um subsistema:
 - As responsabilidades por cada método exposto estão em diversas classes do subsistema.
 - A introdução de um "Façade" irá desacoplar o subsistema dos clientes dos outros subsistemas, promovendo assim, a independência e portabilidade desses subsistemas.

PADRÃO FAÇADE

- Estrutura:



PADRÃO FAÇADE

○ Participantes:

- Façade
 - Conhece quais as classes do subsistema são responsáveis pelo atendimento de uma solicitação.
 - Delega as solicitações do cliente a objetos apropriados do subsistema.
- Classes do Subsistema
 - Implementam a funcionalidade do subsistema.
 - Não são cientes da existência do Façade.
 - Encarregam-se do trabalho delegado pelo Façade.
- Cliente
 - O cliente não acessa as classes do subsistema.

PADRÃO FAÇADE

○ Consequências:

- Torna o subsistema complexo mais **fácil** de utilizar.
- Promove um **acoplamento fraco** entre o cliente e o subsistema.

PADRÃO FAÇADE: alguns detalhes

- As fachadas **não encapsulam** as classes do subsistema, elas apenas fornecem uma interface simplificada. Continua expondo todas as funcionalidades do subsistema para aqueles que precisarem delas.
- O padrão permite criar qualquer número de fachadas
- Permite **desconectar** o cliente de qualquer subsistema específico. Se o subsistema mudar o código do cliente não precisará ser modificado, pois ele acessa a fachada.
- Uma fachada **pode** fornecer uma interface simplificada **para uma única classe** que tenha uma interface muito complexa.

Princípio de Conhecimento Mínimo

- ▶ **Princípio de Conhecimento Mínimo = Lei de Demeter**

“Não fale com estranhos, só fale com seus amigos mais próximos”



Princípio de Conhecimento Mínimo

- ▶ Ao projetar um sistema, deve tomar cuidado com o **número de classes** com que qualquer objeto interage e também a forma como essa interação ocorre.
- ▶ O princípio **impede** de criar projetos com um **grande número de classes interconectadas**, o que faz com que qualquer alteração numa parte do sistema exerça um efeito em cascata sobre outras partes.



Princípio de Conhecimento Mínimo

- ▶ Só podemos invocar métodos que pertençam:
 - Ao próprio objeto
 - A objetos que tenham sido passados como parâmetros para o método
 - A qualquer objeto que seja criado ou instanciado pelo método
 - A quaisquer componentes do objeto



Sem o princípio

```
public float getTemp()  
{  
    Thermometer thermometer=station.getThermometer();  
    return thermometer.getTemperature();  
}
```

Chama um método de um objeto que recebemos como resultado de outra chamada de método. Aumentando o número de objetos que conhecemos diretamente.

Com o princípio

```
public float getTemp()  
{  
    return station.getTemperature();  
}
```

Acrescentamos à classe Station um método que faz a solicitação da temperatura ao objeto thermometer para nós. Isso reduz o número de classes das quais dependemos

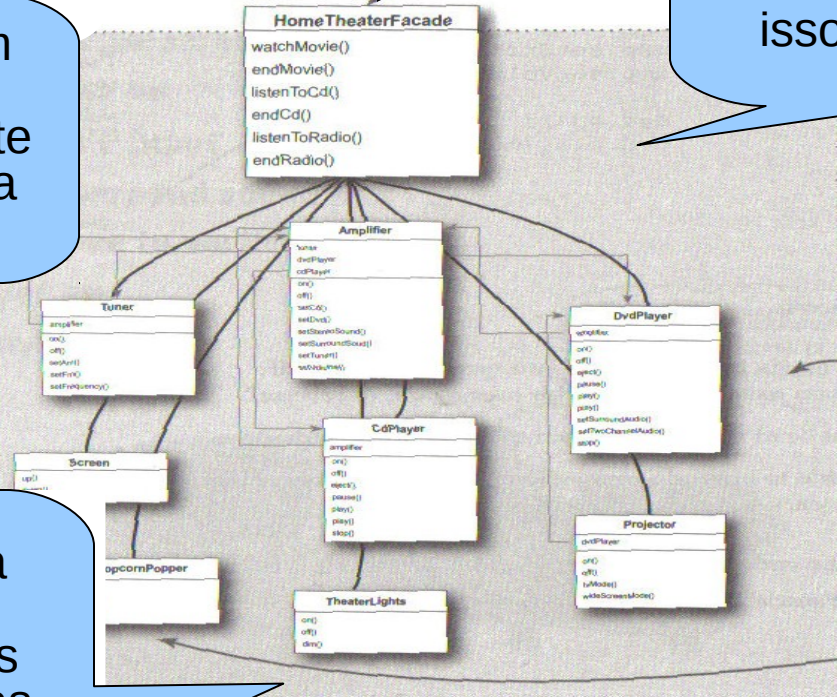
Client

Facade


Podemos trocar os componentes do subsistema sem que isso afete o cliente

O cliente só tem um amigo. Na POO, ter somente um amigo é uma coisa BOA!

Se o subsistema ficar complexo demais, podemos introduzir fachadas adicionais



Resumo Padrão Façade

- ▶ Quando você precisar **simplificar** e unificar um conjunto complexo de interfaces, use uma fachada.
 - ▶ Uma fachada **desconecta** um cliente de um subsistema complexo.
 - ▶ A implementação da fachada exige que **componhamos** a fachada com o seu subsistema e usemos **delegação** para executar o trabalho.
 - ▶ É possível implementar mais de uma fachada para o mesmo subsistema.
- 

Exemplo Sistema de Segurança

- ▶ A figura mostra algumas classes que podem ser usadas em um sistema de segurança.
- ▶ Use o padrão Façade para fornecer um único ponto de contato entre o cliente e o subsistema.

