INSTITUTO FEDERAL
Santa Catarina

Padrões de Projeto de Software

INSTITUTO FEDERAL
Santa Catarina

Revisão

- O que vimos aula passada?

INSTITUTO FEDERAL
Santa Catarina

Façade / Facade / Fachada

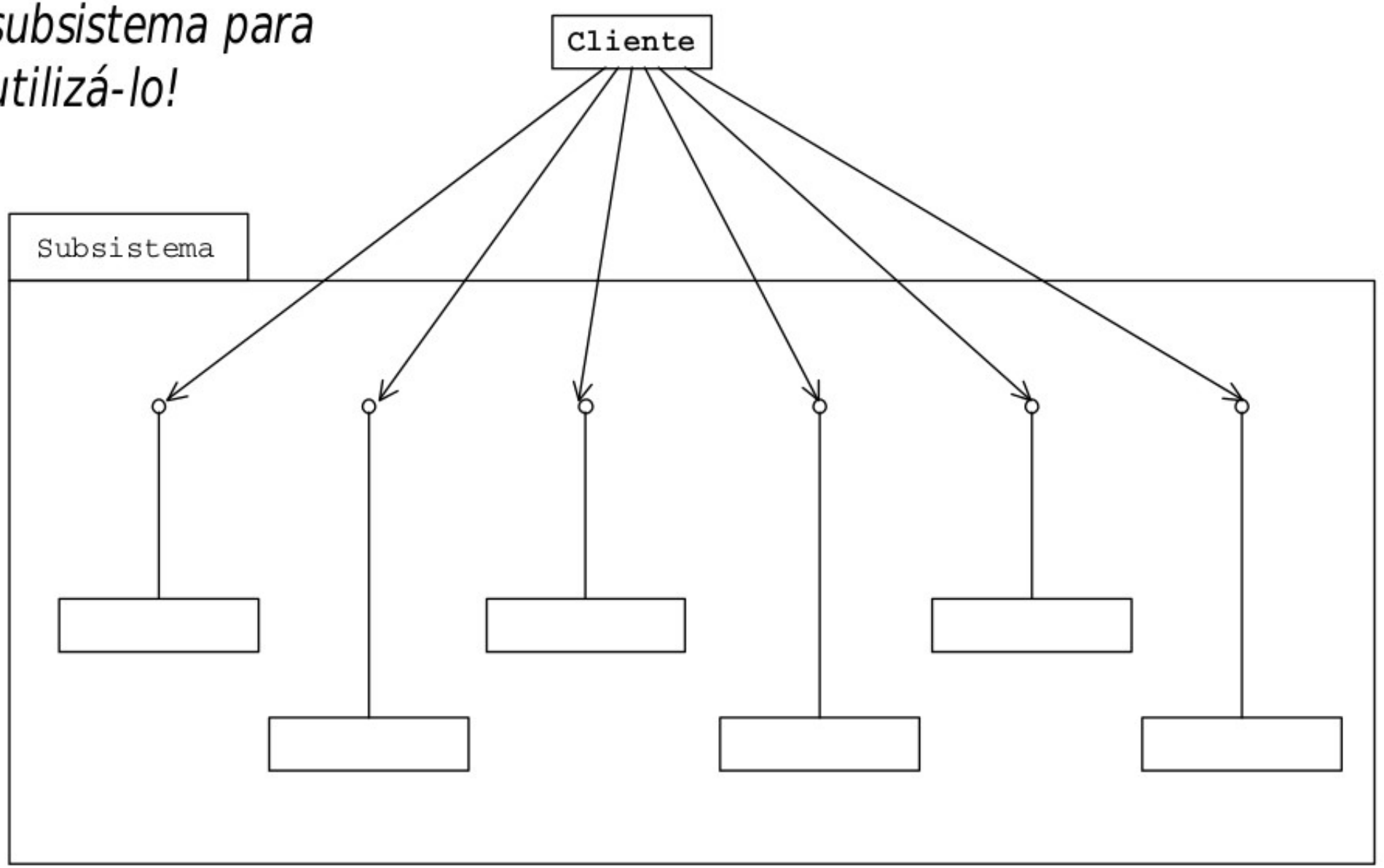
INSTITUTO FEDERAL
Santa Catarina

Facade

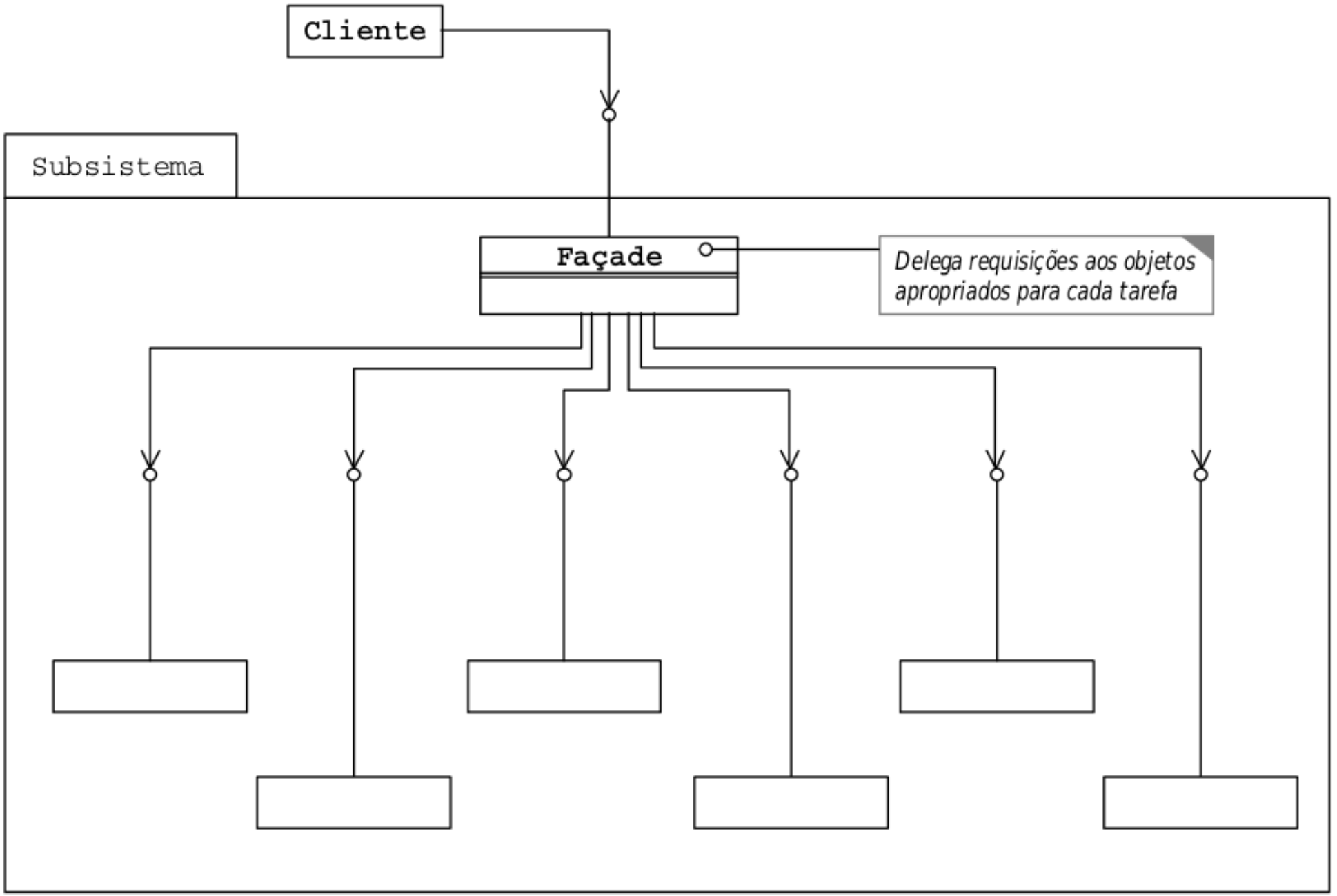
- Oferecer uma interface única para um conjunto de interfaces de um subsistema.
- Facade define uma interface de nível mais elevado que torna o subsistema mais fácil de usar.
- A Facade define uma interface unificada de nível superior para um subsistema que facilita a sua utilização.

Problema

Cliente precisa saber muitos detalhes do subsistema para utilizá-lo!



Estrutura de Façade



INSTITUTO FEDERAL
Santa Catarina

Consequências

- Isola os clientes dos componentes do sub-sistema, reduzindo o número de objetos com os quais o cliente precisa lidar e tornando o sub-sistema mais fácil de usar.
- Promove fraco acoplamento entre o sub-sistema e seus clientes:
  - Componentes de um sub-sistema geralmente são fortemente acoplados.
  - Com o Facade pode-se variar os componentes do sub-sistema sem afetar seus clientes.
- Não impede que aplicações utilizem diretamente as classes do sub-sistema se assim desejarem.

INSTITUTO FEDERAL
Santa Catarina

Aplicabilidade

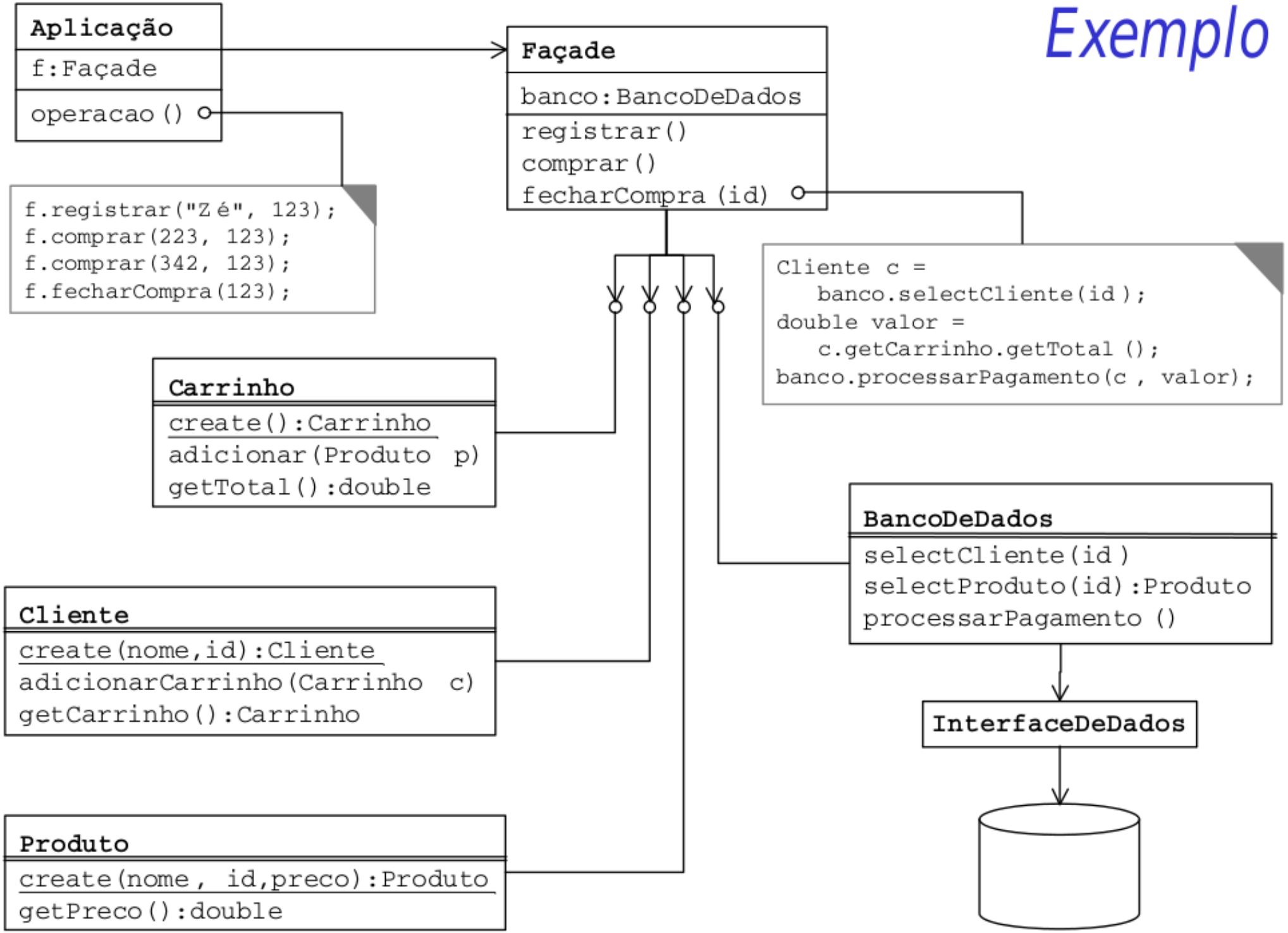
- Deseja-se disponibilizar uma forma de acesso (interface) simples a um sub-sistema complexo:
  - À medida em que evoluem e utilizam mais padrões de projeto, os sistemas passam a ser formados por um número maior de classes, geralmente pequenas.
  - Isso torna o sistema mais reutilizável e fácil de configurar, mas também o torna mais difícil de ser utilizado por clientes que não necessitam configurá-lo.
  - O Facade disponibiliza uma visão simples do sistema, suficiente para a maioria dos clientes. Somente aqueles clientes que precisam de uma maior capacidade de configuração irão acessar o sub-sistema sem utilizar o Facade.

INSTITUTO FEDERAL
Santa Catarina

Implementação

- Reduzindo acoplamento entre o cliente e o sub-sistema:
  - Uma alternativa à herança de interface é configurar o Facade com diferentes objetos do sub-sistema. Para modificar substitui-se um ou mais objetos do Sub-sistema.
- Classes do sub-sistemas públicas ou privadas:
  - Pode-se controlar quais classes do sub-sistema estão disponíveis para os clientes

Exemplo


Aplicação
f:Façade
operacao()
f.registrar("Zé", 123);
f.comprar(223, 123);
f.comprar(342, 123);
f.fecharCompra(123);
Façade
banco:BancoDeDados
registrar()
comprar()
fecharCompra(id)
Cliente c =
banco.selectCliente(id);
double valor =
c.getCarrinho.getTotal();
banco.processarPagamento(c , valor);
Carrinho
create():Carrinho
adicionar(Produto p)
getTotal():double
BancoDeDados
selectCliente(id)
selectProduto(id):Produto
processarPagamento()
Cliente
create(nome,id):Cliente
adicionarCarrinho(Carrinho c)
getCarrinho():Carrinho
InterfaceDeDados
Produto
create(nome, id,preco):Produto
getPreco():double

Façade em Java

```
class Aplicação {
    ...
    Facade f;
    // Obtem instancia f
    f.registrar("Zé", 123);

    f.comprar(223, 123);
    f.comprar(342, 123);

    f.fecharCompra(123);

    ...
}
```

```
public class Facade {
  BancoDeDados banco = Sistema.obterBanco();
  public void registrar(String nome, int id) {
     Cliente c = Cliente.create(nome, id);
     Carrinho c = Carrinho.create();
     c.adicionarCarrinho();
  }
  public void comprar(int prodID, int clienteID) {
     Cliente c = banco.selectCliente(cliente ID);
     Produto p = banco.selectProduto(prodID) {
     c.getCarrinho().adicionar(p);
   }
    public void fecharCompra(int clienteID) {
       Cliente c = banco.selectCliente(clienteID);
       double valor = c.getCarrinho.getTotal();
       banco.processarPagamento(c, valor);
    }
}
```

```
public class Carrinho {
  static Carrinho create() {...}
  void adicionar(Produto p) {...}
  double getTotal() {...}
}
```

```
public class Produto {
  static Produto create(String nome,
                        int id,double preco) {...}
  double getPreco() {...}
}
```

```
public class Cliente {
 static Cliente create(String nome,
                       int id) {...}
 void adicionarCarrinho(Carrinho c) {...}
 Carrinho getCarrinho() {...}
}
```

```
public class BancoDeDados {
 Cliente selectCliente(int id) {...}
 Produto selectProduto(int id) {...}
 void processarPagamento() {...}
}
```

INSTITUTO FEDERAL
Santa Catarina

Factory Method

INSTITUTO FEDERAL
Santa Catarina

Factory Method

- Intenção:
  - Definir uma interface para criar um objeto mas deixar que subclasses decidam que classe instanciar.
  - Factory Method permite que uma classe delegue a responsabilidade de instanciamento às subclasses.

INSTITUTO FEDERAL
Santa Catarina

Motivação

- Criar um objeto sem ter conhecimento algum de sua classe concreta.
- Esse conhecimento não precisa estar no cliente.
- O FactoryMethod define uma interface comum para criar objetos.
- O objeto específico é determinado nas diferentes implementações dessa interface.

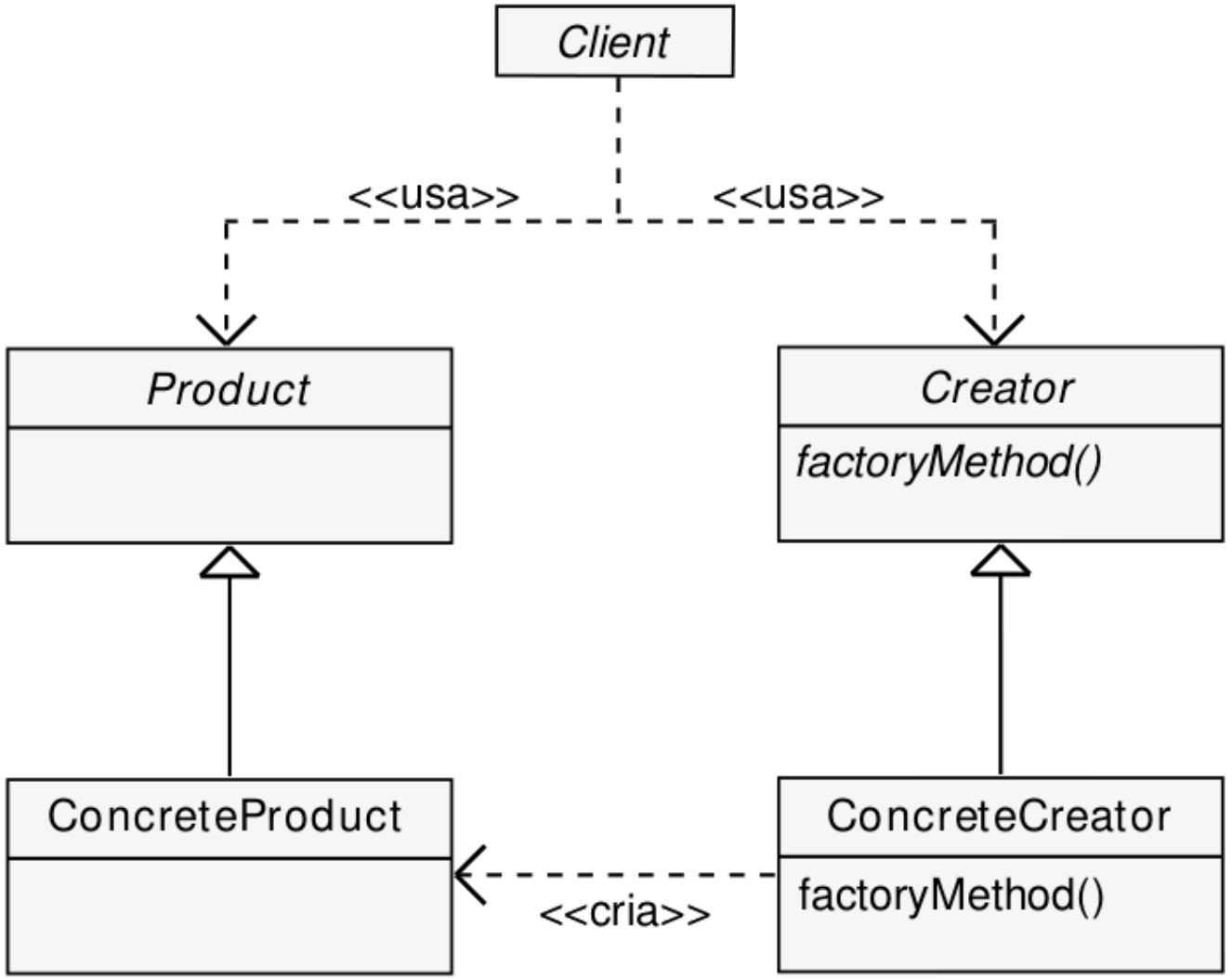
INSTITUTO FEDERAL
Santa Catarina

Aplicabilidade

- Uma classe não conhece previamente a classe de objetos que deve criar.
- Uma classe quer que suas subclasses especifiquem os objetos a serem criados.
- Classes delegam responsabilidade para uma entre várias subclasses auxiliares, e você quer localizar o conhecimento de qual subclasse que é a delegada.

INSTITUTO FEDERAL
Santa Catarina

Estrutura



INSTITUTO FEDERAL
Santa Catarina

Participantes

- Product:
  - define a interface dos objetos que o factory method cria.
- ConcreteProduct:
  - implementa a interface Product.
- Creator:
  - Declara o factory method, retornando um objeto do tipo Product. Pode ser abstrato ou não
  - Geralmente invoca o factory method
- ConcreteCreator: implementa (ou sobrepõe) o factory method para retornar uma implementação específica de Product (algum ConcreteProduct).

INSTITUTO FEDERAL
Santa Catarina

Consequências

- Elimina a necessidade de utilizar, no seu código, classes específicas de aplicação pois utiliza somente a interface Product.
- Uma desvantagem é que os clientes podem ter que fornecer uma subclasse Creator somente para criar um objeto ConcretProduct.
- Flexibilidade ao criar um objeto.

INSTITUTO FEDERAL
Santa Catarina

Implementação

- Creator como classe abstrata:
  - Requer a existência de sub-classes que implementam o factory method.
- Creator com implementação default do factory method:
  - Não requer um ConcreteCreator. Define o factory method simplesmente para tornar o código mais flexível: sub-classes podem futuramente sobrescrever o factory method.
- Factory methods parametrizados:
  - O factory method recebe um parâmetro indicando o tipo de produto concreto a ser criado.