INSTITUTO FEDERAL
Santa Catarina

Padrões de Projeto de Software

INSTITUTO FEDERAL
Santa Catarina

Revisão

- O que vimos aula passada?

INSTITUTO FEDERAL
Santa Catarina

Strategy

- Intenção:
  - Definir uma família de algoritmos, encapsular cada uma delas e torná-las intercambiáveis.
  - Permitir que o algoritmo varie independentemente dos clientes que o utilizam.

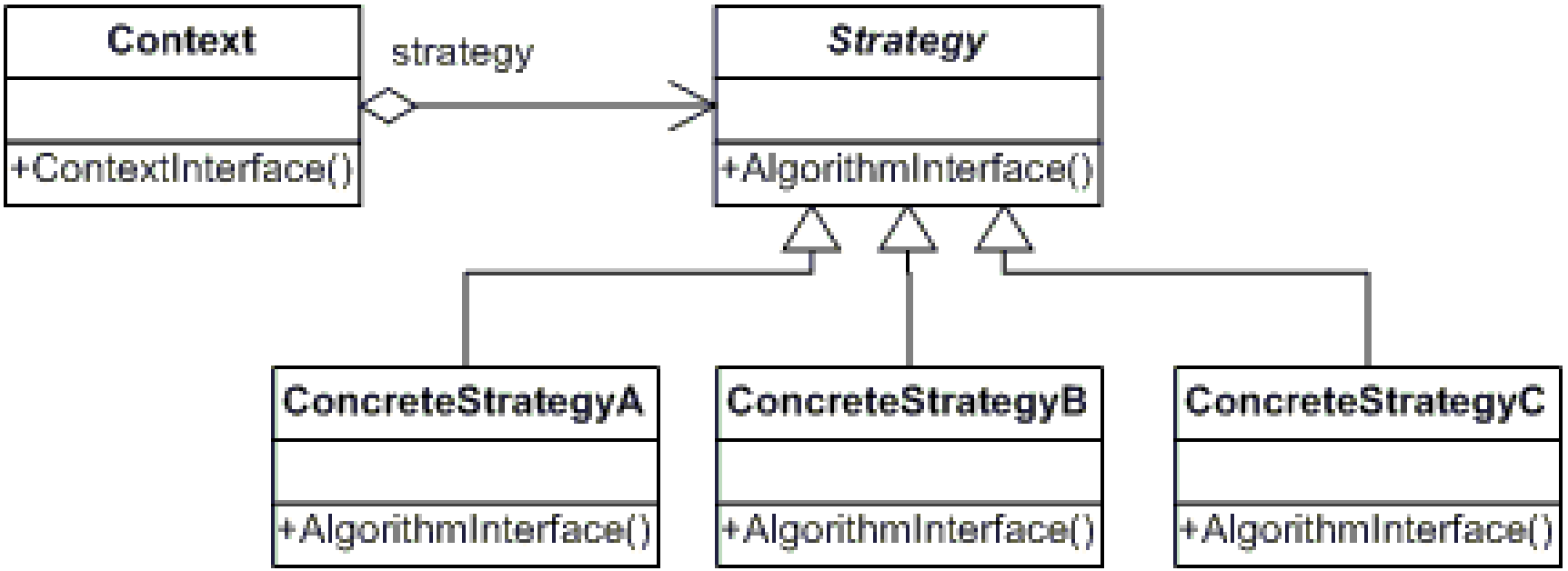
INSTITUTO FEDERAL
Santa Catarina

Aplicabilidade

- Quando muitas classes relacionadas diferem somente no seu comportamento.
- Quando é necessário utilizar diferentes variações de um algoritmo.
- Quando um algoritmo utiliza dados que não devem estar expostos aos clientes.
- Quando uma classe define múltiplos comportamentos através de sentenças condicionais em seus métodos.

INSTITUTO FEDERAL
Santa Catarina

Estrutura



INSTITUTO FEDERAL
Santa Catarina

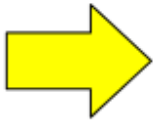
Participantes

- Strategy:
  - Declara uma interface comum a todos os algoritmos suportados.
  - O Context utiliza esta interface para chamar um algoritmo definido por um ConcreteStrategy.
- ConcreteStrategy:
  - Implementa o algoritmo utilizando a interface Strategy.
- Context:
  - É configurado com um ConcreteStrategy.
  - Mantém uma referência para o objeto Strategy.
  - Pode definir uma interface que permite que os ConcreteStrategies acessem seus dados.

Problema

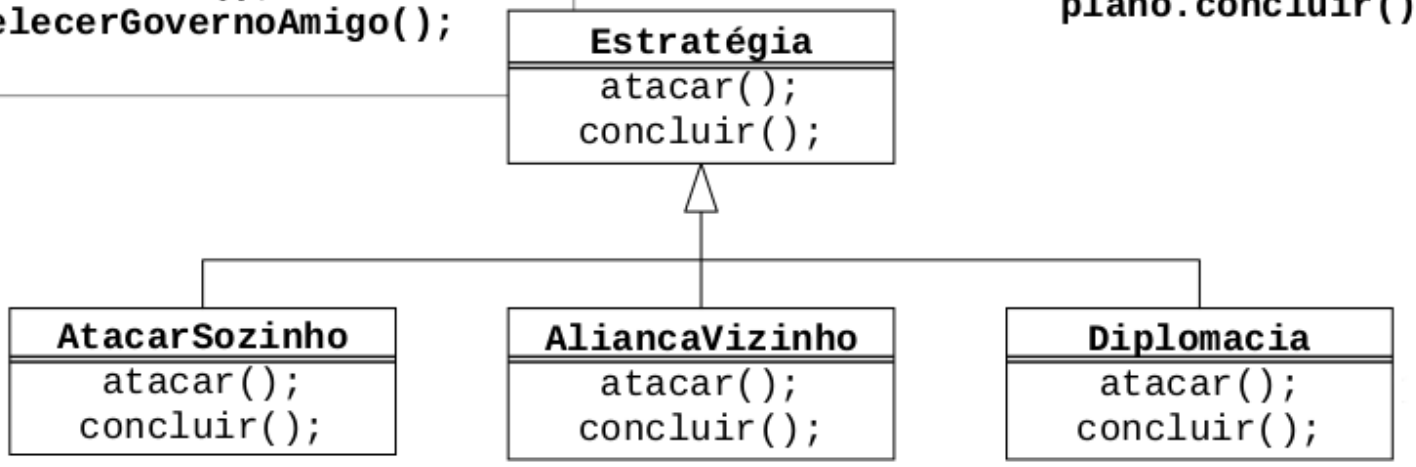
Várias estratégias, escolhidas de acordo com opções ou condições

```
if (inimigo.exercito() > 10000) {
     fazerAlianca();
     vizinhoAtacaPeloNorte();
     nosAtacamosPeloSul();
     dividirBeneficios(...);
     dividirReconstrução(...);
} else if (inimigo.isNuclear()) {
     recuarTropas();
     proporCooperacaoEconomica();
     desarmarInimigo();
} else if (inimigo.hasNoChance()) {
     plantarEvidenciasFalsas();
     soltarBombas();
     derrubarGoverno();
     estabelecerGovernoAmigo();
}
```



```
if (inimigo.exercito() > 10000) {
   plano = new AliancaVizinho();
} else if (inimigo.isNuclear()) {
   plano = new Diplomacia();
} else if (inimigo.hasNoChance())
   plano = new AtacarSozinho();
}
```

```
plano.atacar();
plano.concluir();
```



Em Java

```
public class Guerra {
   Estrategia acao;
   public void definirEstrategia() {
     if (inimigo.exercito() > 10000) {
        acao = new AliancaVizinho();
     } else if (inimigo.isNuclear()) {
        acao = new Diplomacia();
     } else if (inimigo.hasNoChance()) {
        acao = new AtacarSozinho();
     }
   }
   public void declararGuerra() {
      acao.atacar();
   }
   public void encerrarGuerra() {
      acao.concluir();
   }
}
```

```
public interface Estrategia {
   public void atacar();
   public void concluir();
}
```

```
public class AtacarSozinho
        implements Estrategia {
   public void atacar() {
      plantarEvidenciasFalsas();
      soltarBombas();
      derrubarGoverno();
   }
   public void concluir() {
      estabelecerGovernoAmigo();
   }
}
```

```
public class AliancaVizinho
        implements Estrategia {
   public void atacar() {
      vizinhoAtacaPeloNorte();
      nosAtacamosPeloSul();
      ...
   }
   public void concluir() {
      dividirBeneficios(...);
      dividirReconstrução(...);
   }
}
```

```
public class Diplomacia
        implements Estrategia {
   public void atacar() {
      recuarTropas();
      proporCooperacaoEconomica();
      ...
   }
   public void concluir() {
      desarmarInimigo();
   }
}
```

INSTITUTO FEDERAL
Santa Catarina

Façade / Facade / Fachada

INSTITUTO FEDERAL
Santa Catarina

Exemplo

INSTITUTO FEDERAL
Santa Catarina

Exemplo

- Empresa de televendas.
- O consumidor liga para o número e fala com uma atendente.
- A atendente atua como uma fachada, fornecendo uma interface para o departamento de atendimento aos pedidos, o departamento de pagamento e o departamento de entrega.

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.

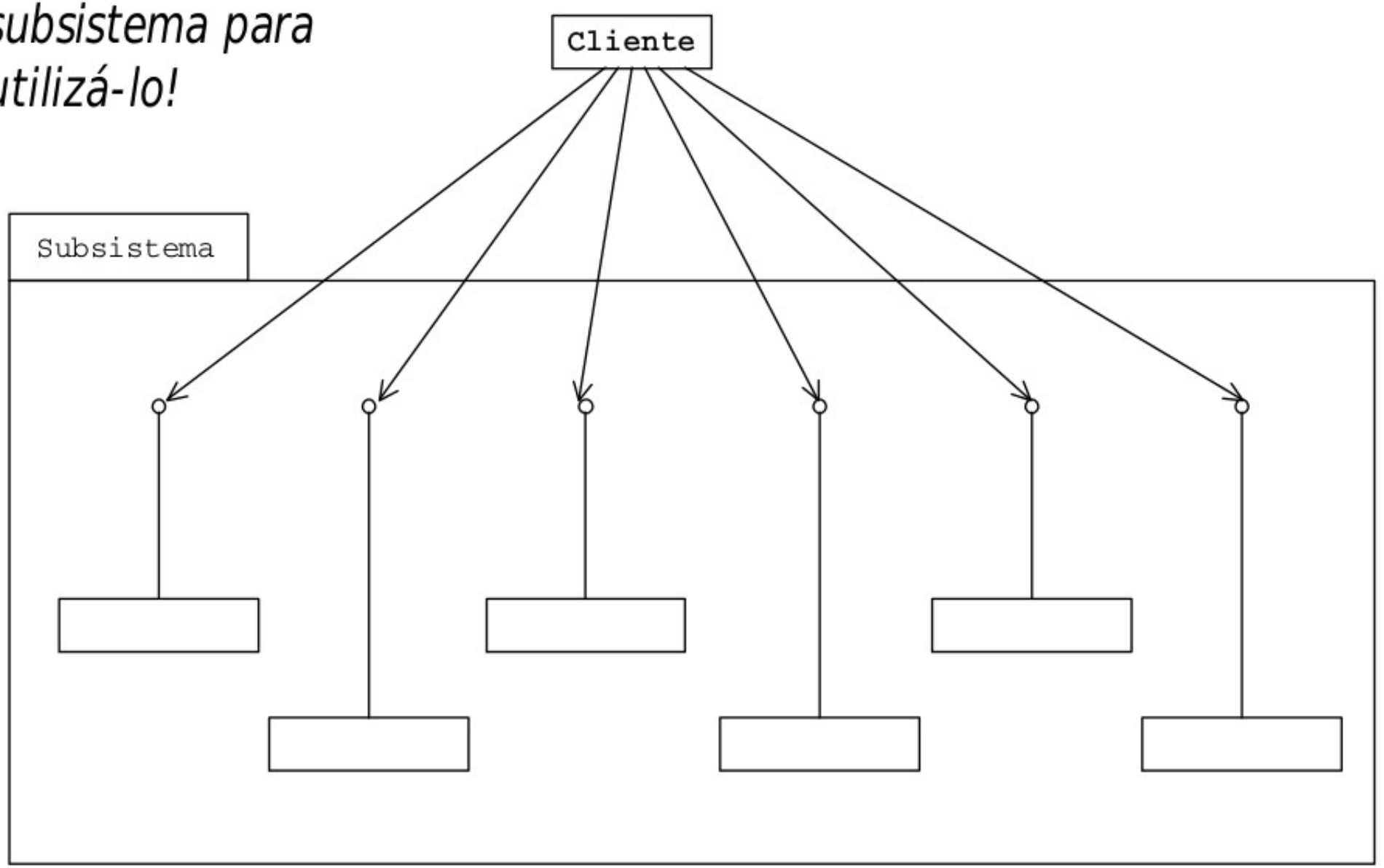
INSTITUTO FEDERAL
Santa Catarina

Motivação

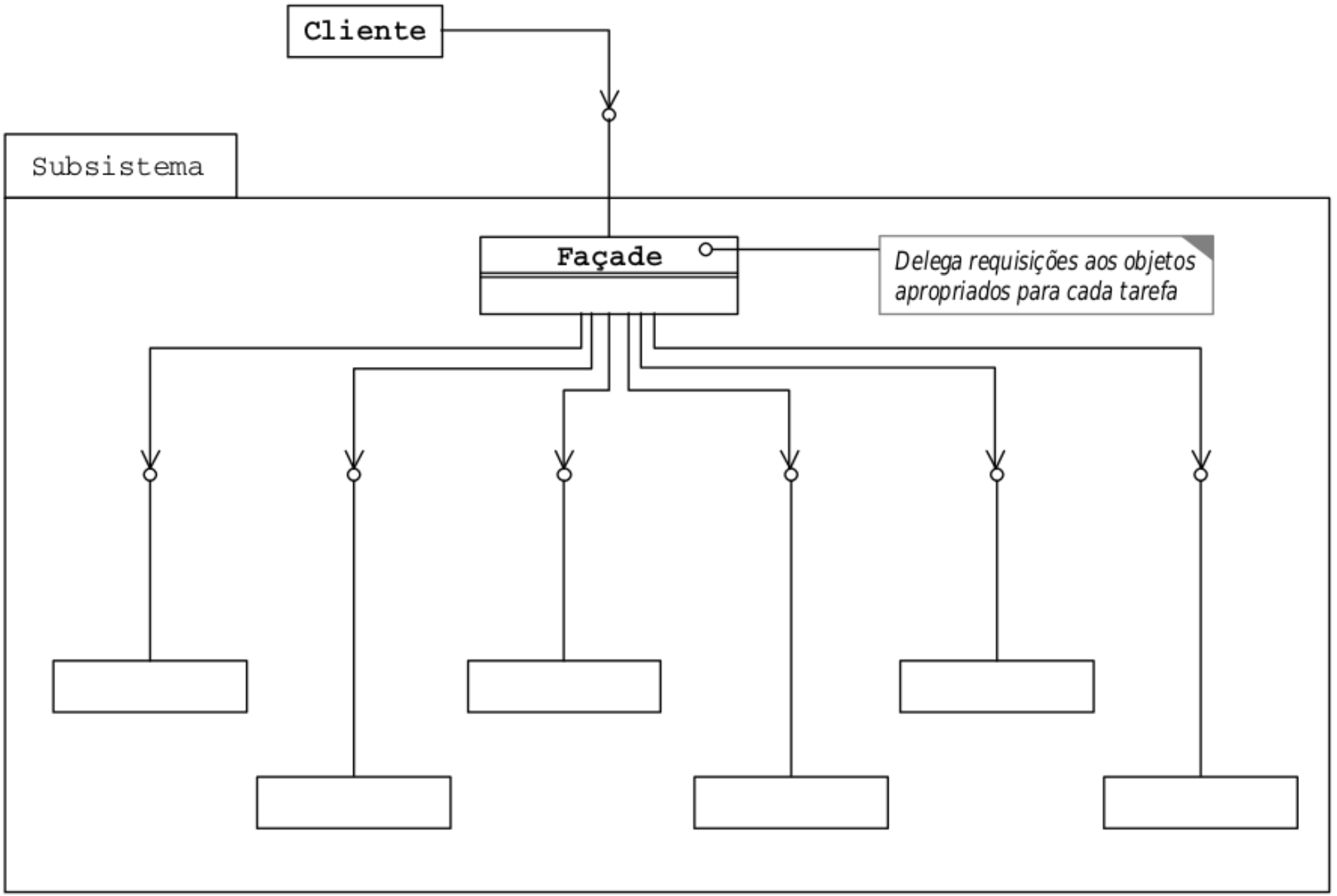
- Estruturar um sistema em sub-sistemas ajuda a reduzir complexidade.
- Geralmente deseja-se minimizar as comunicações e dependências entre sub-sistemas.
- O Facade pode ser utilizado para este objetivo.

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

Aplicabilidade

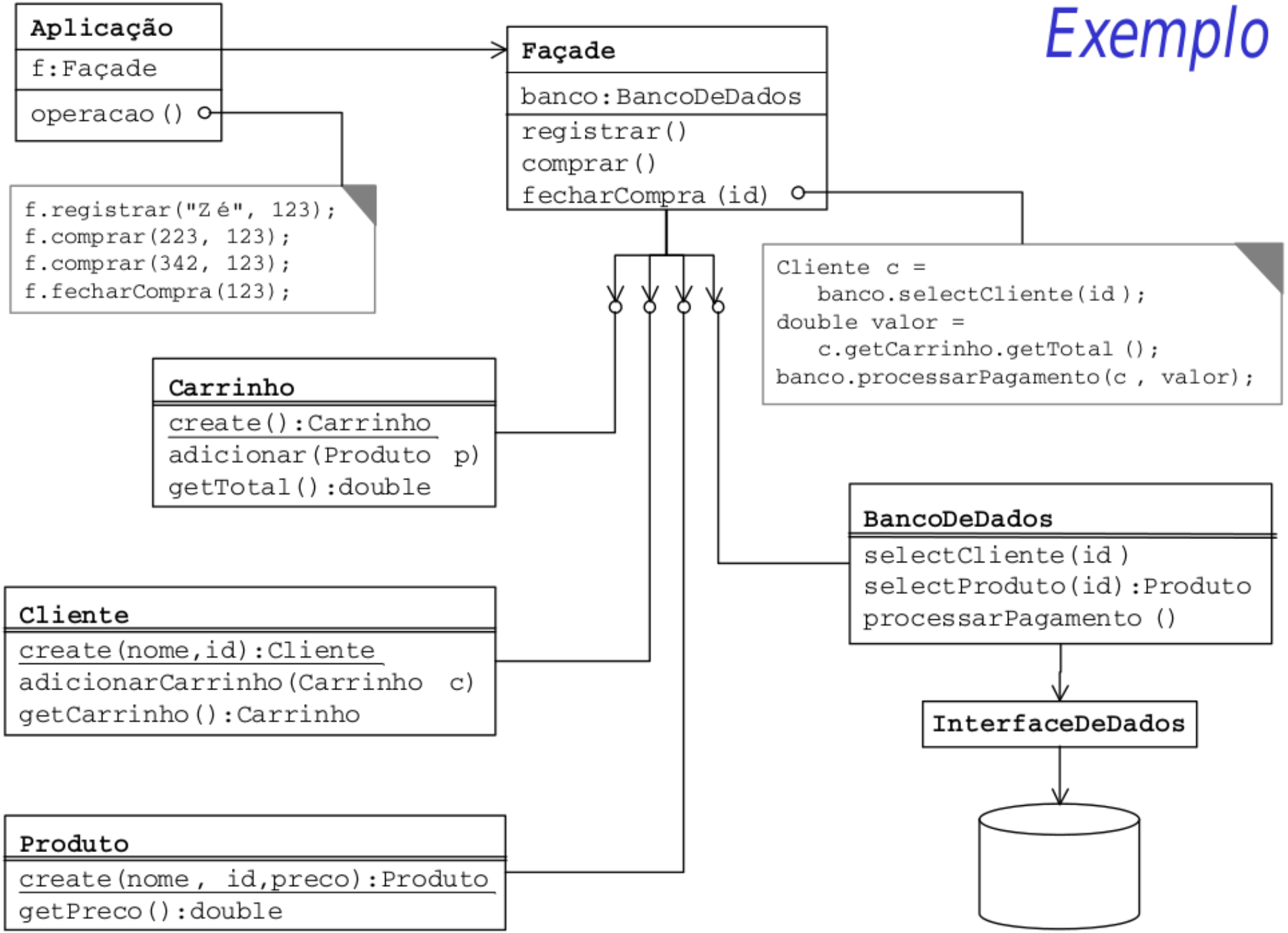
- Existe muitas dependências entre os clientes e as classes de implementação de uma abstração:
  - O Facade desacopla o sub-sistema dos clientes e também de outros sub-sistemas.
  - Promove a independência e portabilidade do sub-sistema.

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

Exercícios

- Utilize o padrão Facade no problemas disponíveis no SIGAA.