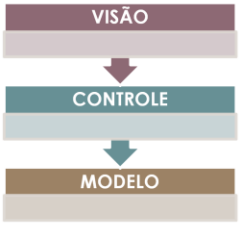
Arquitetura e Desenho de Software

AULA 10
BASE – PARTE II



Profa. Milene Serrano

UnB Gama

Agenda

- Considerações Iniciais
- Exemplo Passo a Passo com GRASP (FOCO: Demais Padrões GRASPs)
- Considerações Finais

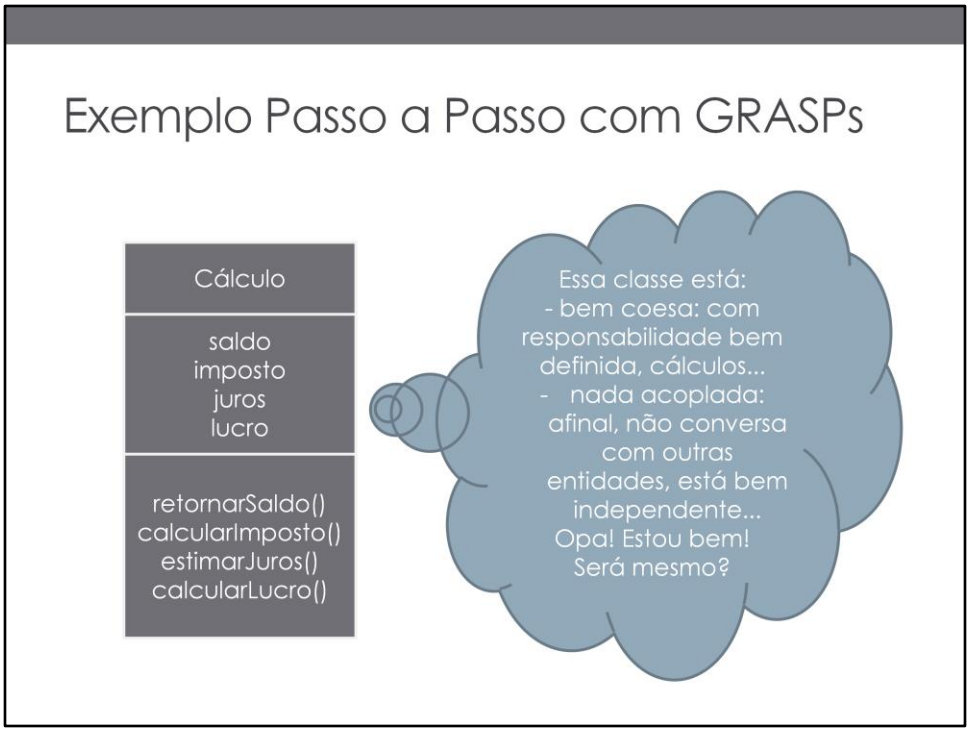
Considerações Iniciais

Considerações Iniciais

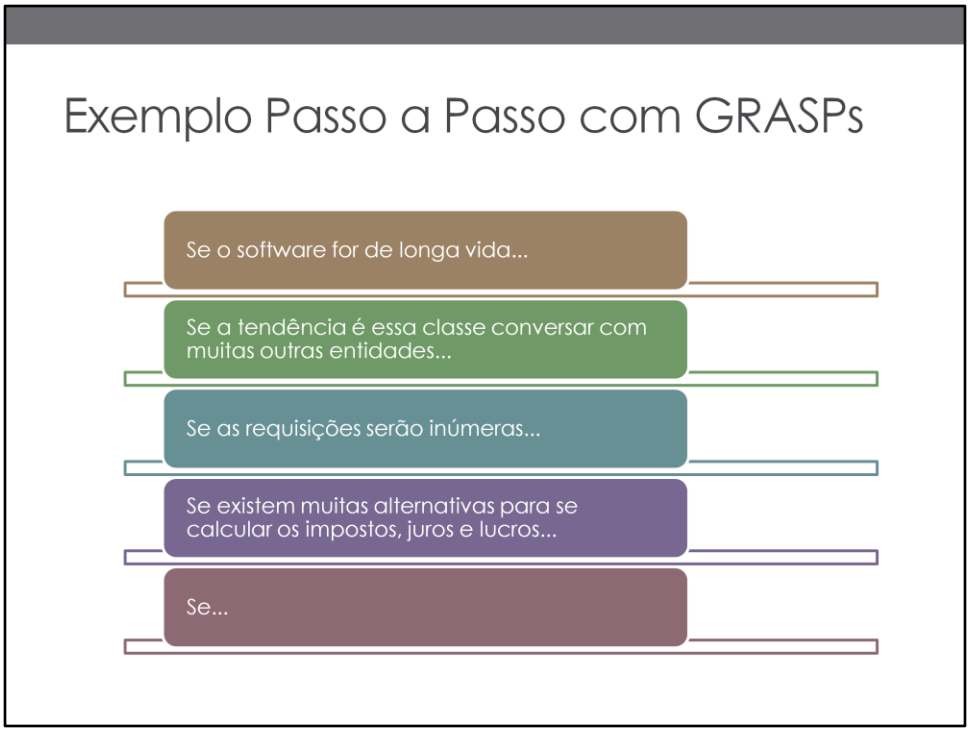
Exemplo

FOCO: Demais Padrões GRASPs

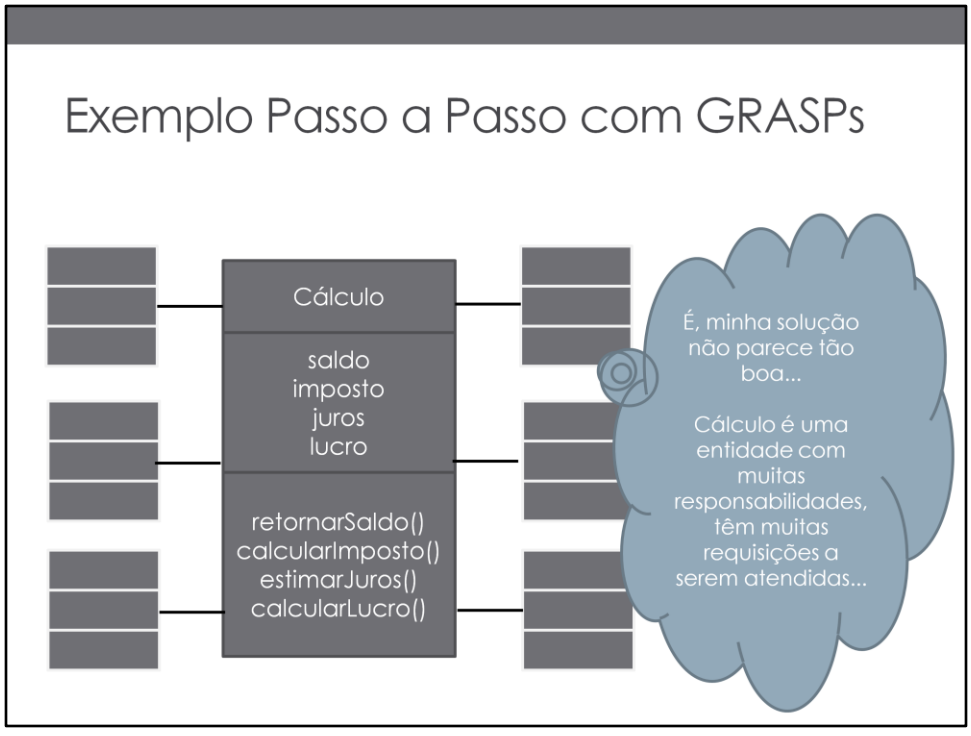
Como a ideia é entender a aplicabilidade dos GRASPs, iremos focar nessas boas práticas...



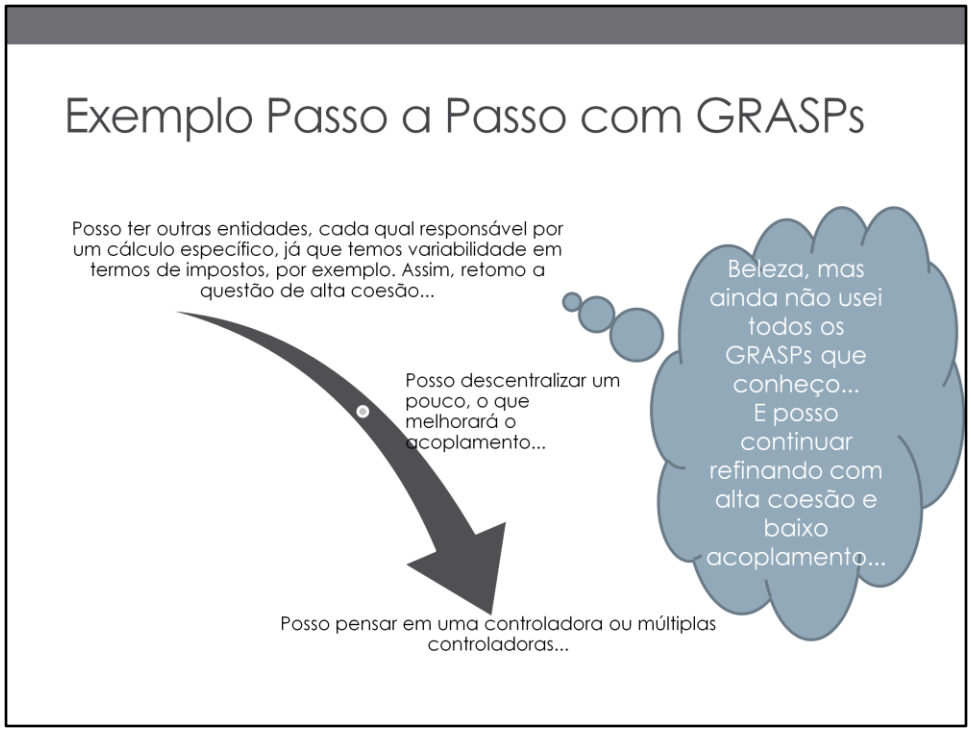
Começando por desenhar uma classe mais genérica, responsável por cálculos... Afinal, no contexto financeiro, temos cálculos envolvendo vários aspectos, dentre eles: impostos, juros, lucros e outros.



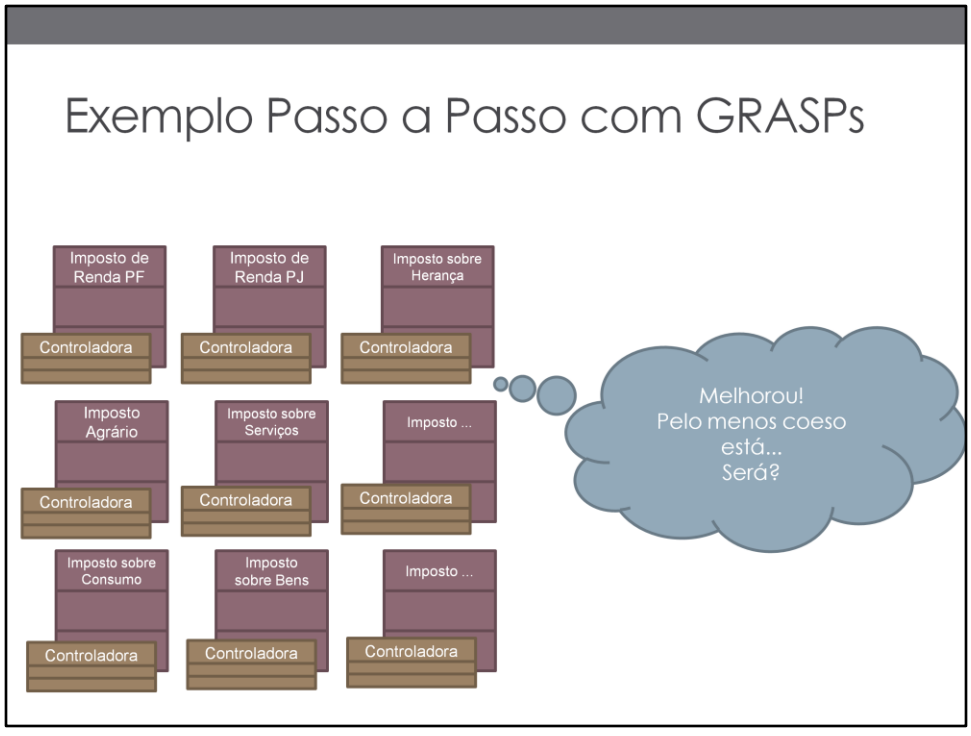
Portanto, o que parecia coeso e nada acoplado, quando nos aprofundamos mais um pouco no domínio em questão, notamos que ainda não estamos representando bem esse domínio, o qual é complexo e requer muito mais entidades, relacionamentos, requisitos não funcionais e outras demandas.



Agora, o acoplamento, por exemplo, está bem mais alto...
Trata-se de uma classe que centraliza muitas demandas e requisições de terceiros...
Tenho como melhorar?
Posso modularizar algo?
Há alguma boa prática que posso aplicar?



Reparem que alta coesão e baixo acoplamento permeiam várias das refatorações ao longo da modelagem...
Apoiem-se nessas boas práticas, pois os modelos tendem a melhorar bastante. Ok?

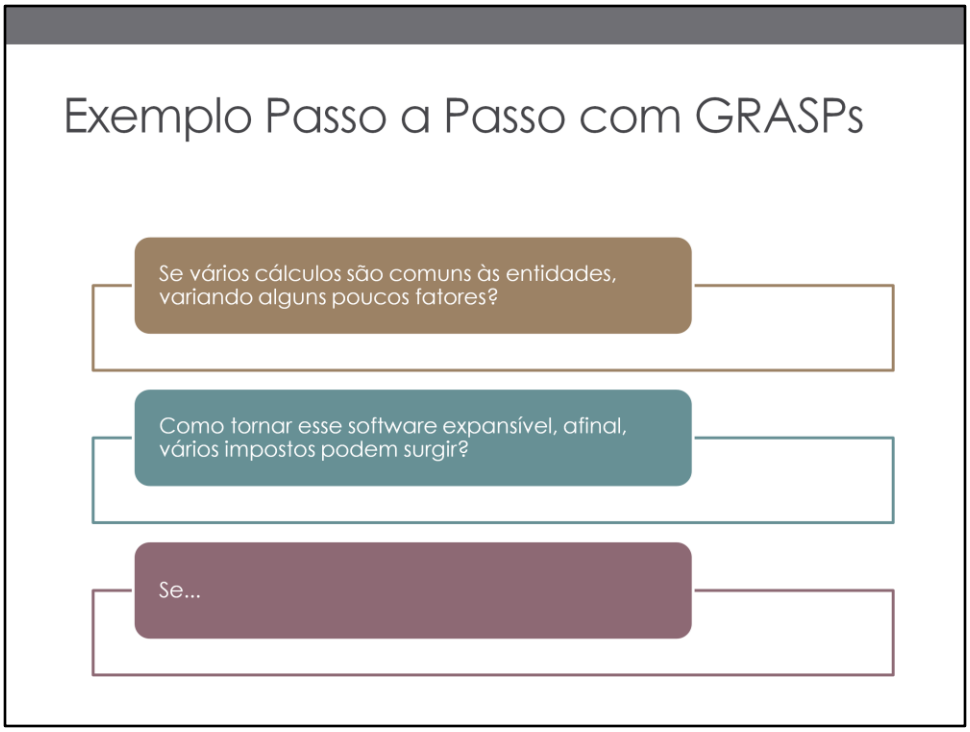


Controladora de Fachada
Controladora por Caso de Uso
Múltiplas Controladoras

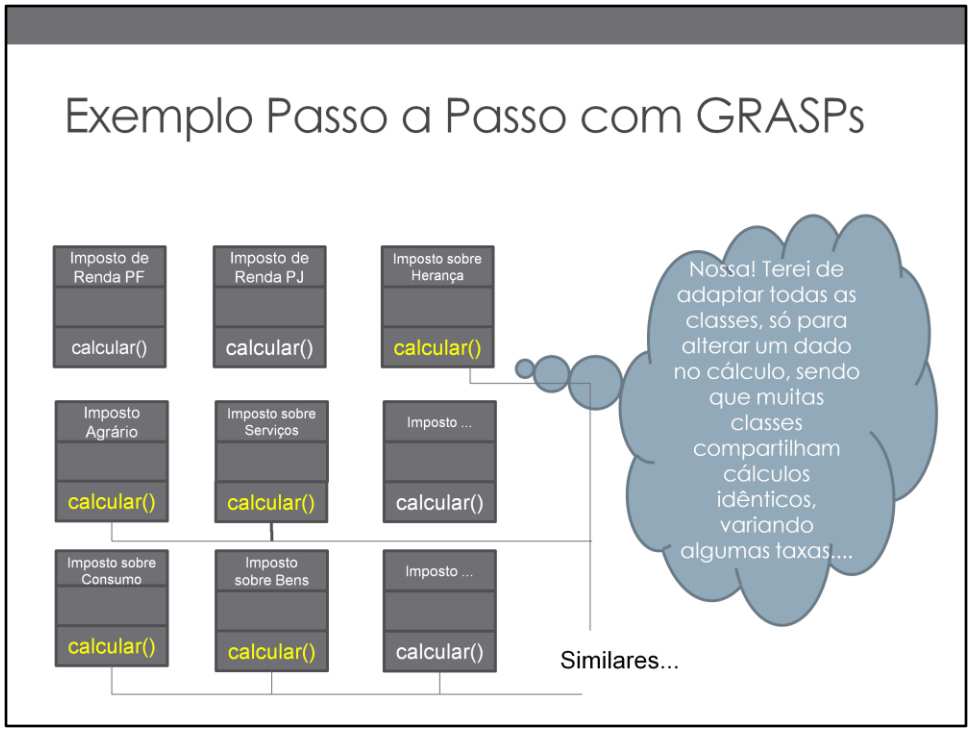
Debater sobre:
Quando optar pelo uso de cada uma dessas abordagens?

- Normalmente, quando o sistema é menor, com poucos requisitos e poucas operações de entrada, uma controladora de fachada pode ser suficiente.
- Mas, quando os requisitos demandam mais esforços, o sistema cresce, opta-se, preferencialmente, por mais controladoras.
    - Pode ser uma por caso de uso. Essa é uma abordagem que vale para alguns casos.
    - Podem ser múltiplas controladoras, uma para cada classe de domínio. Essa é uma abordagem mais atual. Em plataformas mais atualizadas, como Ruby On Rails e Grails, temos o uso desse tipo de abordagem.

CONTROLLER – Mais um padrão GRASP!



Mas, esse sistema ainda é bem mais complexo.
Portanto, vamos continuar revelando suas particularidades, o que demandará novas refatorações na modelagem...

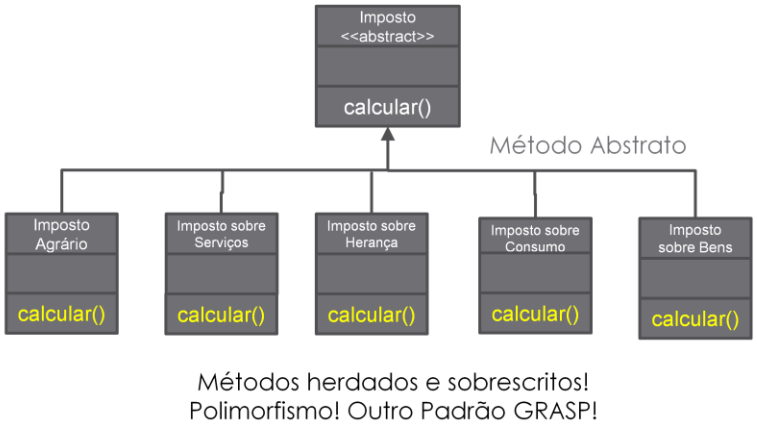


Em amarelo, estou considerando que são métodos com comportamentos similares, mas que diferenciam dependendo da instância, seja essa:

- Um imposto sobre herança;
- Um imposto agrário;
- Um imposto sobre serviços;
- Um imposto sobre consumo, ou
- Um imposto sobre bens.

Tem como melhorar esse aspecto, facilitando a interatividade, a reutilização bem como a manutenção do sistema?
Mais GRASPs para colaborar nesses aspectos...
Agora, vamos introduzir o POLIMORFISMO...
Esse é um princípio de OO bem como um padrão GRASP muito relevante, sendo base para os padrões GoFs comportamentais.

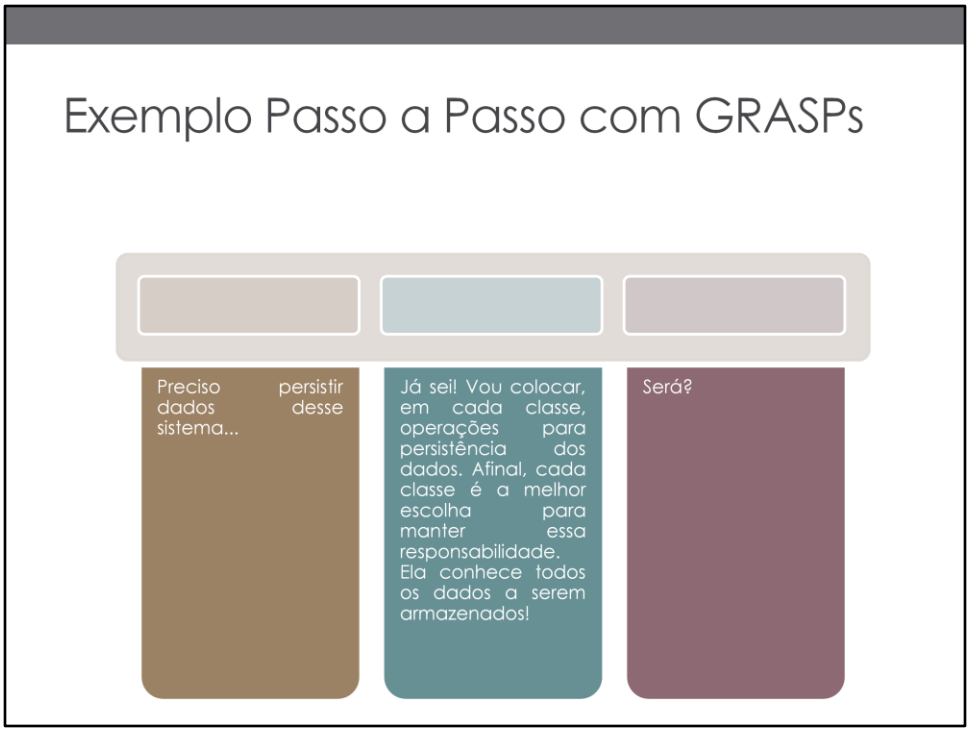
Exemplo Passo a Passo com GRASPs



Métodos herdados e sobrescritos!
Polimorfismo! Outro Padrão GRASP!

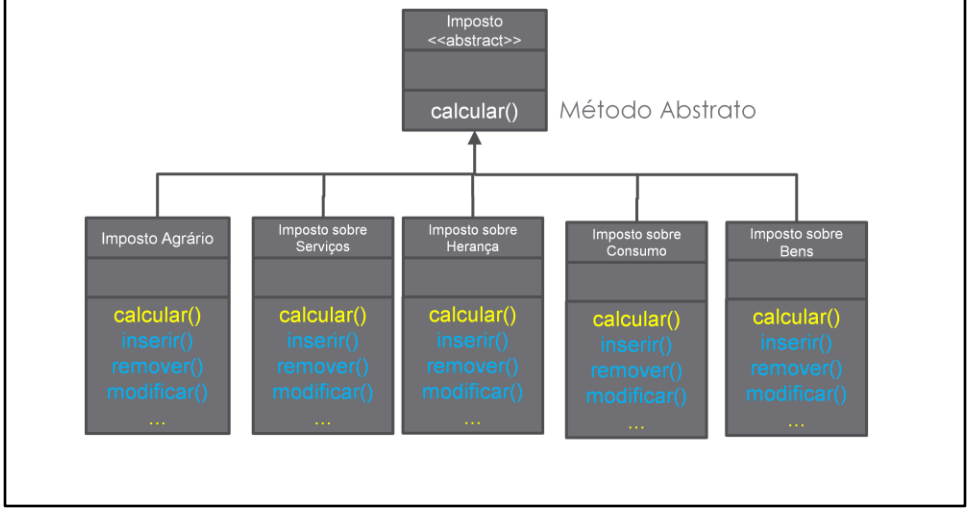
Debater sobre particularidades, tais como:

- Quando utilizar o conceito de *interface* <<java>>?
  - Vantagens?
  - Desvantagens?
  - Modelagem?
  - Implementação?
- Quando utilizar o conceito de classe abstrata <<java>>?
  - Vantagens?
  - Desvantagens?
  - Modelagem?
  - Implementação?
- Quando utilizar o conceito de uma classe concreta no nível superior da herança?
  - Vantagens?
  - Desvantagens?
  - Modelagem?
  - Implementação?
- Quais as vantagens e desvantagens de se usar o conceito de herança?



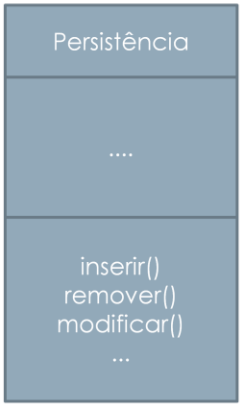
Ah! O domínio ainda demanda maior atenção, agora, na questão de persistência de dados. Ok?
Então, vamos introduzir outros GRASPs para alinhar essas demandas.
No caso, focaremos em INVENÇÃO PURA e INDIREÇÃO...

Exemplo Passo a Passo com GRASPs

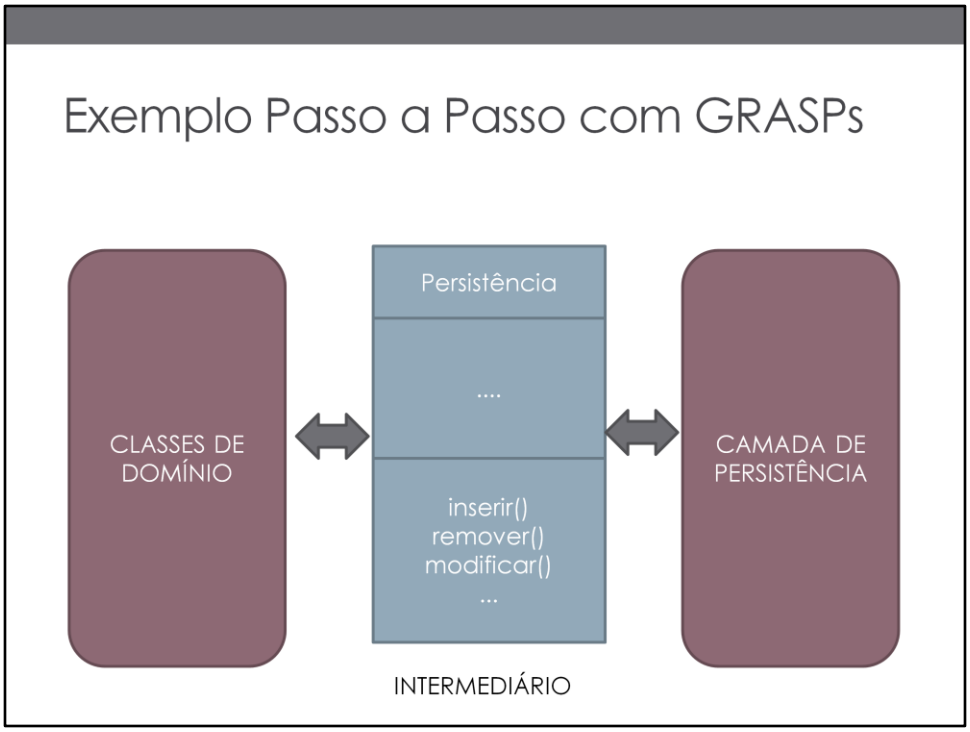


Em azul, métodos para garantir a persistência dos dados...
A distribuição de responsabilidades parece estar coerente...
Mas, está mesmo?
Quais problemas temos na modelagem?
<<debater>>

Exemplo Passo a Passo com GRASPs

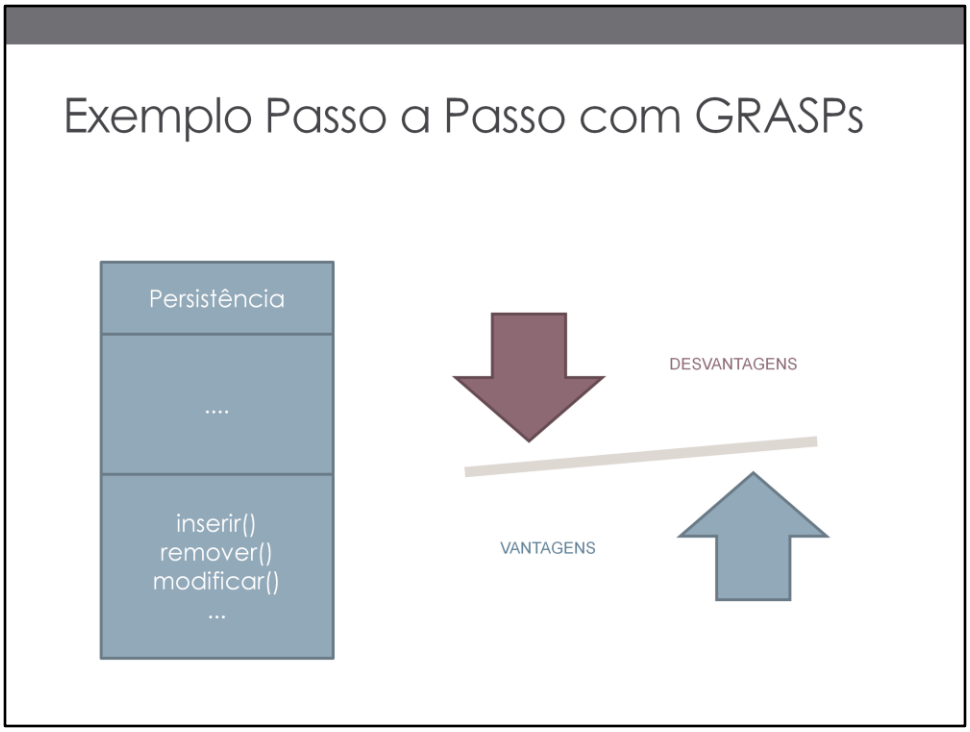


Já sei!
Mantenho como estava, em termos de hierarquia...
Mas, criarei uma classe técnica específica para conter essas operações de persistência...
Estará coeso, pelo menos...
Criar uma classe específica para lidar com uma preocupação, tornando a modelagem mais coesa, é uma boa prática.
Trata-se de um padrão GRASP chamado: Invenção Pura, Fabricação ou Fabricação Própria!



Essa classe tem ainda mais uma particularidade, no caso, está intermediando algo, como se comportasse como um mediador.
Nesse contexto, está intermediando as conversas das classes de domínio com a camada de persistência.
Portanto, temos a aplicação de mais um padrão de projeto GRASP, chamado: Indireção.
****

Portanto, os intermediários das indireções costumam ser invenções puras.
Mas, nem toda invenção pura é indireção.
****



Ainda debatendo sobre problemas e melhorias...

Tem que pensar sobre as demandas do sistema...
Se o sistema estiver sobrecarregando essa classe, seria mais adequado não centralizar essa conversa entre classes de domínios e camada de persistência em uma única classe.
Uma abordagem mais modularizada caberia nesse caso...

Cuidados…
-Usar essa estratégia em conformidade com as boas práticas de cada tecnologia.
-Dependendo da tecnologia, se a comunidade já utiliza uma estratégia – que difere dessa prática – optar por não realizar essa separação das operações de persistência.
-Estar de acordo com os padrões já reconhecidos por uma comunidade de programação – por exemplo – é mais relevante do que aplicar um padrão GRASP só para constar ou dizer que está usando-o.
Fica a dica! :)

Por fim, como *calcular()* é um método com variabilidade alta...
Portanto, usando polimorfismo (como ilustrado), escopo protegido (com atributos/métodos devidamente declarados como *private,* e acesso controlado por mecanismos de autenticação e autorização) e indireções e invenções (melhorando questões como acoplamento e coesão), podemos atingir um padrão GRASP muito recomendado quando se deseja extensibilidade, manutenção melhorada e reutilização de software.
Trata-se de Variações Protegidas...

Considerações Finais

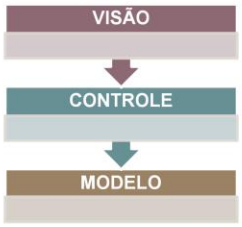
Adicionalmente, ao final da aula, recapitulamos sobre cada padrão de projeto GRASP (9 no total):
Criador, Especialista, Alta Coesão, Baixo Acoplamento, Controller, Polimorfismo, Invenção Pura, Indireção e Variações Protegidas.
Continuem os estudos, implementem os padrões nos trabalhos.

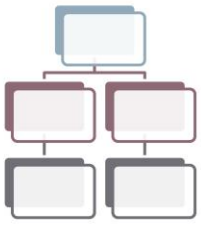
Estudem, adicionalmente, usando um exemplo modelado em UML e implementado em Java, em um domínio cognitivo de interesse.

Referências

Referências

- Aulas 10B e 10C, disponíveis no Moodle, nesse mesmo tópico.





FIM

Dúvidas?

CONTATO:
mileneserrano@unb.br
ou
mileneserrano@gmail.com

Sugestões?

UnB Gama