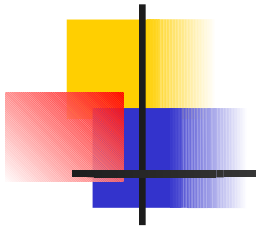


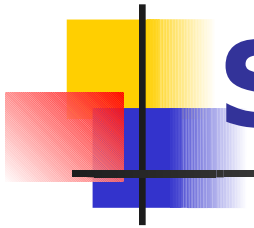
Padrões de Projeto em Desenvolvimento Web

SCC-266

Prof. Renata Pontin M. Fortes -
renata@icmc.usp.br
PAE: **Willian** Watanabe **(watinha@gmail.com)**

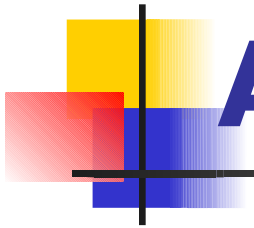
2.semestre 2010

Instituto de Ciências Matemáticas e de Computação ICMC USP



Sumário

- Apresentação da Disciplina
- Princípios de *Design* (projeto)



A Disciplina

- Requisitos “formais” da Disciplina
  - SCC0204 - Programação Orientada a Objetos
  - SCC0263 - Técnicas de Programação para Web

  - Requisito informal:
  - SCC-265 Sistemas Interativos Web

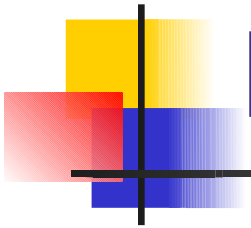
Ementa Disciplina

- **Objetivos**

  introduzir **padrões de projeto** para o desenvolvimento de aplicações Web envolvendo as etapas de navegação, processamento de formulários, banco de dados, autenticação e manipulação de exceções e erros. **Esses padrões são documentos** que, formalmente, descrevem abordagens para solucionar problemas envolvidos no projeto de sistemas Web.

- **Programa Resumido**

  O que é padrão de projeto. Descrição de padrões. Como projetar padrões de projeto. Como escolher e implementar um padrão de projeto. Padrões mais comuns para criação, estruturais e comportamentais: Abstract Factory, Builder, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Facade, Proxy, Iterator, Mediator, Observer, State e Strategy. Padrões para desenvolvimento WEB.



Programa Disciplina

- Introdução aos Padrões de Projeto
  - ***Design Patterns (DP)***
- Classificações
- Gang of Four (GoF)
- Interativos (Welie)
- Ajax

Proposta Disciplina

- Aulas teóricas
- 2 **S**eminários (em duplas)
- 2 **P**rojetos (em grupos de 4 alunos)

Avaliação Disciplina

- Os 2 Seminários: S1 e S2 → **MS** (média aritmética)

- Os 2 Projetos: P1 e P2 (parcial e final):
  - P1p, P1f, P2p, P2f → **MP**
  - **MP = 20%P1p + 30%P1f + 20%P2p + 30%P2f**

Média Final MF:

Se MS ≥ 5 e MP ≥ 5,

então **MF = (30% MS) + (70% MP)**

Caso contrário, **MF = min(MS, MP)**

Freqüência mínima (presença) **70%** *i.é,* ***4 faltas*** *no máx.*

Sites Disciplina

CoteiaWIKI

http://wiki.icmc.usp.br

Tidia-Ae

http://agora.tidia-ae.usp.br

Visitem sempre!!!!!!

Biblio apoio Disciplina

Design Patterns Java™ Workbook (*Steven John Metsker*)
http://proquestcombo.safaribooksonline.com/ 0201743973

Patterns in Interaction Design (*Martijn van Welie*)
http://www.welie.com/patterns/

Patterns in Interaction Design (*Mahemoff*)
http://proquestcombo.safaribooksonline.com/0596101805

Estudem SEMPRE!!

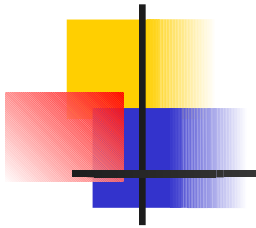
Calendário Disciplina

03/ago: aula
10/ago: aula
17/ago: reunião PET → *sem aula*
24/ago: aula + 5 seminários
31/ago: aula + 5 seminários

07/set: recesso
14/set: **P1parcia**l (sorteio 7 grupos)
21/set: aula + 5 seminários
28/set: SIGDOC → *sem aula*

05/out: **P1final** (8 grupos)
12/out: recesso
19/out: aula
26/out: aula + 5 seminários

02/nov: recesso
09/nov: **P2parcial** (sorteio 7 grupos)
16/nov: aula + 5 seminários
23/nov: aula + 5 seminários
30/nov: **P2final** (8 grupos)

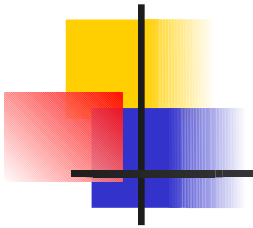


Design Patterns

Design Principles
and Design Patterns
(Robert C. Martin, 2000)

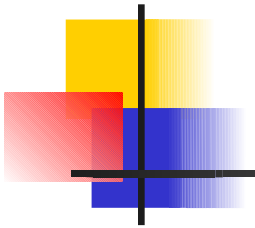
Arquitetura do Software?

- Multicamadas!
- No alto nível, estão os **padrões arquiteturais**, que definem a forma geral e a estrutura do software
- No nível mais abaixo, está a **arquitetura** que está especificamente relacionada com o propósito da aplicação
- No nível mais baixo, a arquitetura dos módulos e suas interligações.



Arquitetura e Dependências

- *O que acontece de errado com software?*
- O projeto (design) de muitas aplicações de software inicia com uma **imagem** na mente dos projetistas
- → clara, elegante e convincente
- .... até a primeira entrega (*release*)



Arquitetura e Dependências

- Conforme os requisitos evoluem, o software começa a 'apodrecer'.
- Necessidade de re-projeto!!
- Os re-designs raramente são bem sucedidos...
- Embora os projetistas iniciem com boas intenções, eles sentem como se estivessem “atirando no alvo em movimento”

Sintomas de Projeto apodrecendo

- *Os quatro principais sintomas:*

(1) Rigidez
(2) Fragilidade
(3) Imobilidade
(4) Viscosidade

Sintomas de Projeto apodrecendo

- *Os quatro principais sintomas:*

(1) Rigidez

é a tendencia do software ser dificil para mudar, mesmo nas partes mais simples

→ cada mudança causa uma cascata de mudanças subsequentes nos módulos dependentes

Sintomas de Projeto apodrecendo

- *Os quatro principais sintomas:*

(1) Rigidez

(2) Fragilidade

é a tendencia do software quebrar em muitos locais cada vez que é modificado

→ geralmente a quebra ocorre em áreas que não têm relacionamento conceitual com a área que foi alterada

Sintomas de Projeto apodrecendo

- *Os quatro principais sintomas:*

(1) Rigidez

(2) Fragilidade

(3) Imobilidade

é a incapacidade de reusar o software de outros projetos ou de partes do mesmo projeto. Geralmente acontece que um engenheiro discobrirá que ele precisa de um módulo similar a um que outro desenvolvedor escreveu.

Sintomas de Projeto apodrecendo

- *Os quatro principais sintomas:*

(1) Rigidez

(2) Fragilidade

(3) Imobilidade

(4) Viscosidade

quando desenvolvedor se depara com diferentes formas /modos para mudar o software (algumas preservam o projeto, outras não)

→ qdo as formas que preservam os projetos são mais difíceis : alta viscosidade!

→ “é facil fazer a coisa errada, mas difícil a coisa certa”

CAUSAS do Projeto apodrecendo

- *Os quatro principais sintomas:*
  - (1) Rigidez
  - (2) Fragilidade
  - (3) Imobilidade
  - (4) Viscosidade

- As causas:
  - Mudança de requisitos
  - Gerenciamento das dependencias

Princípios de projetos OO

- Princípios de Projeto de Classes:
  - OCP – *open closed principle*
  - LSP – *Liskov Substitution principle*
  - DIP – *Dependency Inversion Principle*
  - ISP – *Interface Segregation principle*

Princípios de projetos OO

- Princípios de Projeto de Classes:
  - OCP – *open closed principle*

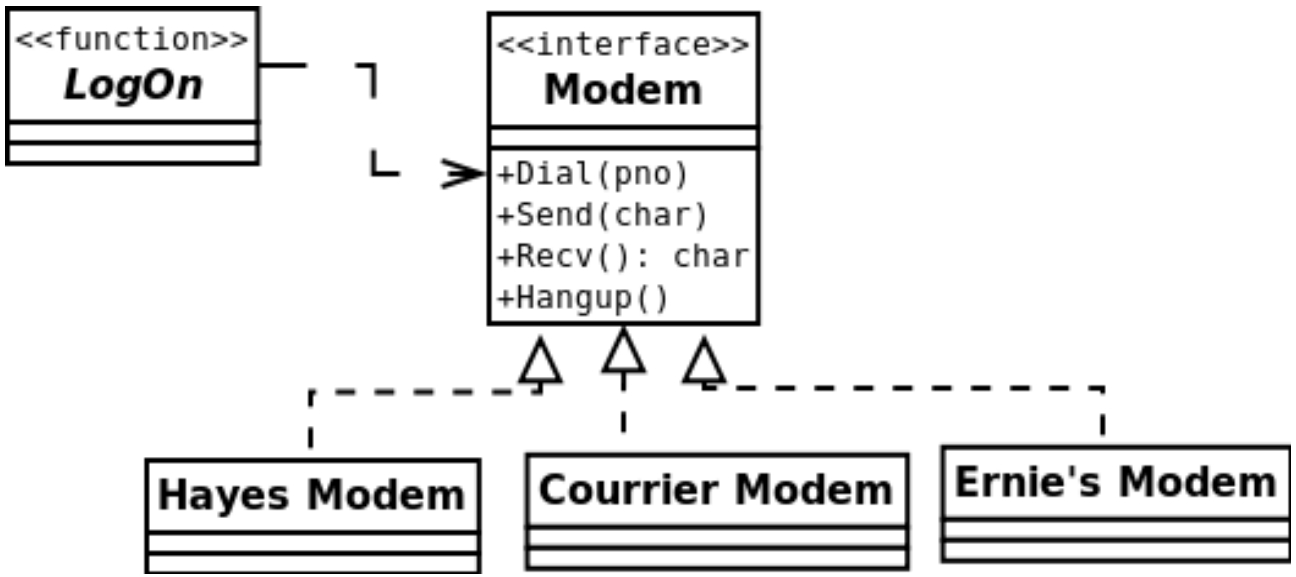
    “A module should be open for extension but closed for modification”

    → módulo deve estar aberto para ser estendido, e fechado para modificação
  - Queremos ser capazes de mudar o que o módulo faz, sem mudar o código fonte dos módulos
  - → *abstração!!*

Princípios de projetos OO

- Princípios de Projeto de Classes:
  - OCP – *open closed principle*

    → *polimorfismo*
  -



Princípios de projetos OO

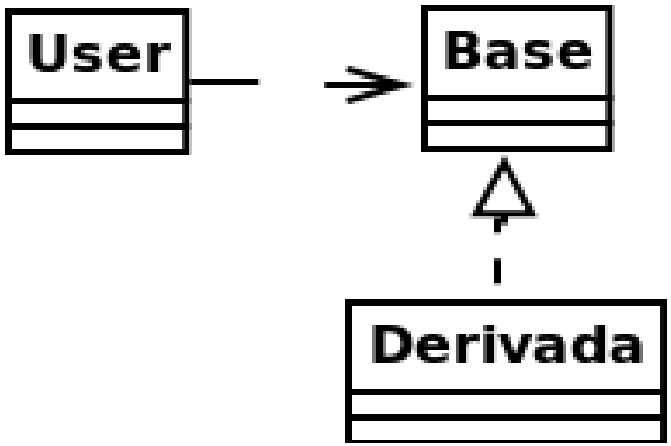
- Princípios de Projeto de Classes:
  - LSP – *Liskov Substitution principle*

    "*Subclasses should be substitutable for their bases classes*"

    → as classes derivadas devem ser substituiveis por suas classes base

Princípios de projetos OO

- Princípios de Projeto de Classes:
  - LSP – *Liskov Substitution principle*



  - → se alguma função **User** tem um argumento do tipo **Base**, deveria ser legal passar uma instancia de **Derivada** para a função.

Princípios de projetos OO

- Princípios de Projeto de Classes:
  - DIP – *Dependency Inversion Principle*

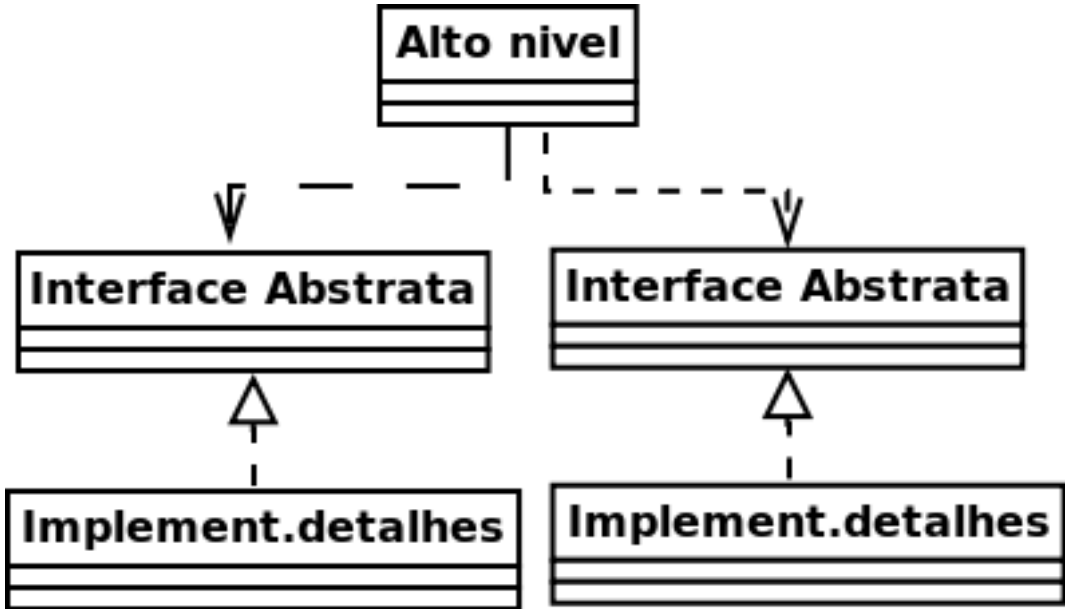
    “Depend upon Abstractions. Do not depend upon concretions.”

    → *dependa de abstrações e não de concretizações.*
  - Inversão de dependência.

    A razão é de que coisas concretas mudam muito, e as asbtratas menos frequentemente.

Princípios de projetos OO

- Princípios de Projeto de Classes:
  - DIP – *Dependency Inversion Principle*



Princípios de projetos OO

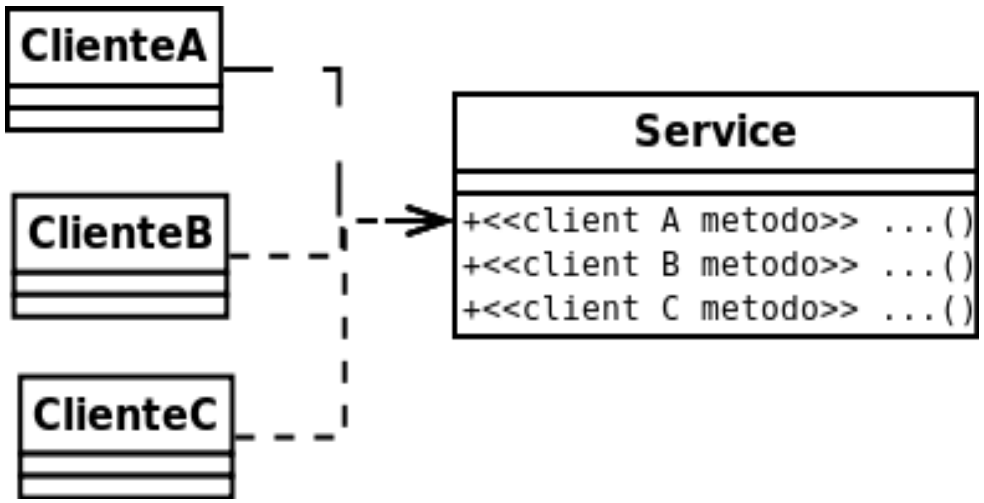
- Princípios de Projeto de Classes:
  - ISP – *Interface Segregation principle*

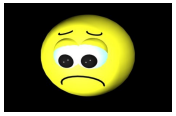
    “Many client specific interfaces are better than one general purpose interface”

    → muitas interfaces clientes específicas são melhores do que uma interface de propósito geral.

Princípios de projetos OO

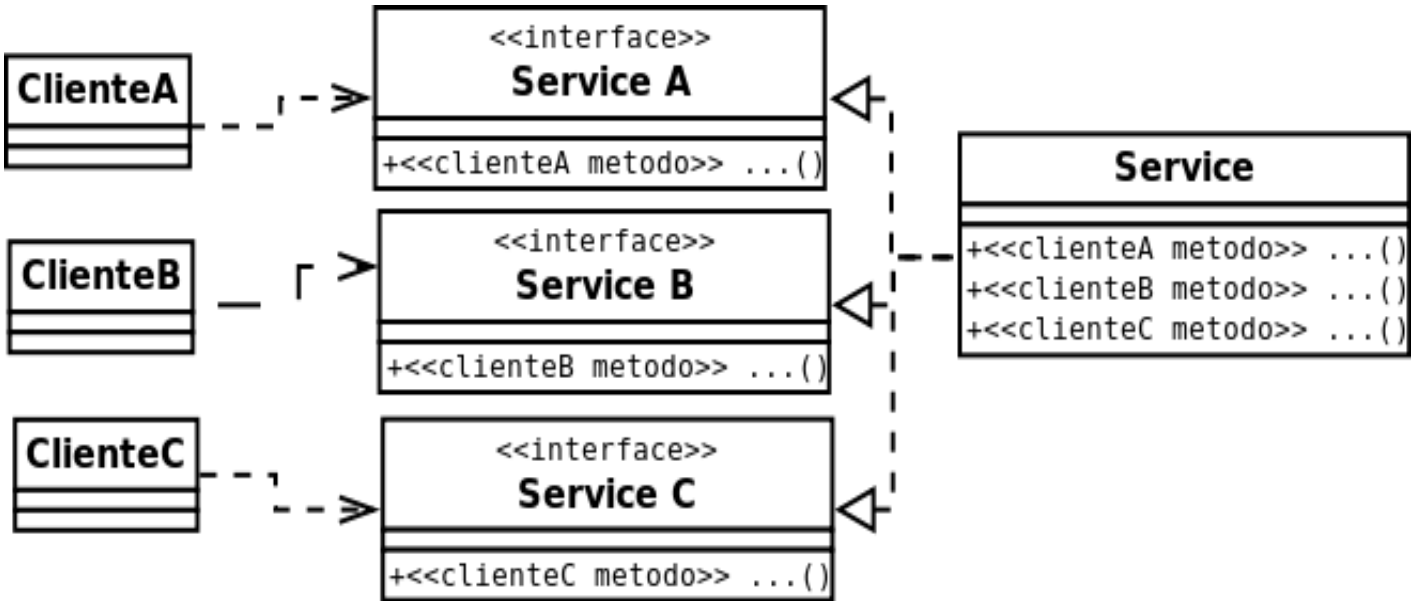
- Princípios de Projeto de Classes:
  - ISP – *Interface Segregation principle*

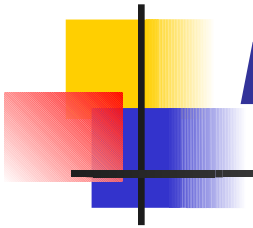




Princípios de projetos OO

- Princípios de Projeto de Classes:
  - ISP – *Interface Segregation principle*

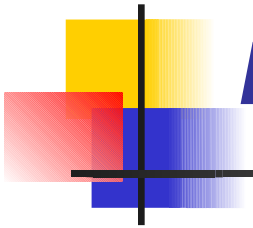




Princípios de projetos OO

- Princípios de Arquitetura de **Pacotes**:
  - REP – *release reuse equivalency principle*
  - CCP – *Common Closure principle*
  - CRP – *Common Reuse Principle*

  - ADP – *Acyclic Dependencies principle*
  - SDP – *Stable Dependencies principle*
  - SAP – *Stable Abstractions principle*

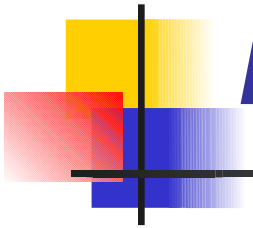


Princípios de projetos OO

- Princípios de Arquitetura de Pacotes:
  - **REP** – *release reuse equivalency principle*

    → *a granulosidade de reuso é a granulosidade da release (entrega)*
  - Assim, um critério para agrupar classes é o REUSO.
  - Uma vez que pacotes são as unidades da release, eles são também a unidade de reuso.

    → *facilita para reuso!*

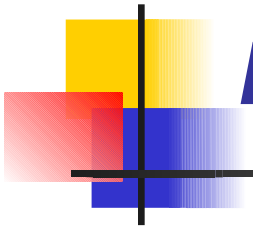


Princípios de projetos OO

- Princípios de Arquitetura de Pacotes:
  - **CCP** – *Common Closure principle*

    → *classes que mudam junto, permanecem juntas*
  - Assim, um critério para agrupar classes é a partir da previsão de quais irão mudar no mesmo momento.

    → *facilita para manutenção!*

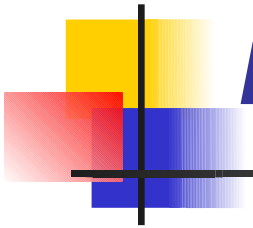


Princípios de projetos OO

- Princípios de Arquitetura de Pacotes:
  - **CRP** – *Common Reuse principle*

    → *classes que que não são reutilizadas juntas não deveriam ser agrupadas juntas*
  - Uma dependencia de um pacote é uma dependencia de TUDO que está incluído no pacote.

    → *facilita para reuso!*

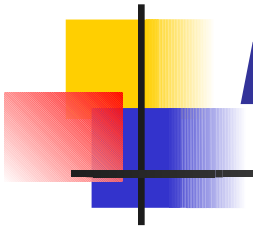


Princípios de projetos OO

- Princípios de Arquitetura de Pacotes:
  - REP – *release reuse equivalency principle*
  - CCP – *Common Closure principle*
  - CRP – *Common Reuse Principle*

    → ***COESÃO***
  - ADP – *Acyclic Dependencies principle*
  - SDP – *Stable Dependencies principle*
  - SAP – *Stable Abstractions principle*

    → ***ACOPLAMENTO***



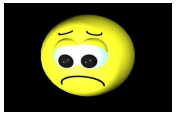
Princípios de projetos OO

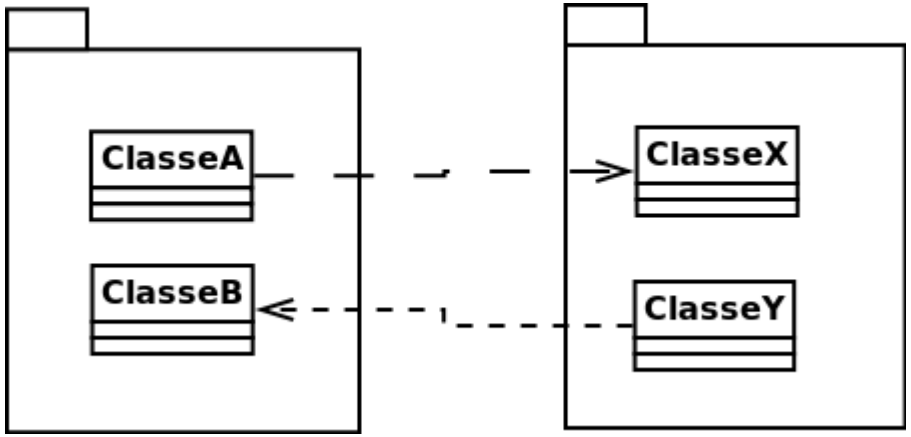
- Princípios de Arquitetura de Pacotes:
  - **ADP** – *Acyclic Dependencies principle*

    → *as dependencias entre os pacotes não devem formar ciclos*

Princípios de projetos OO

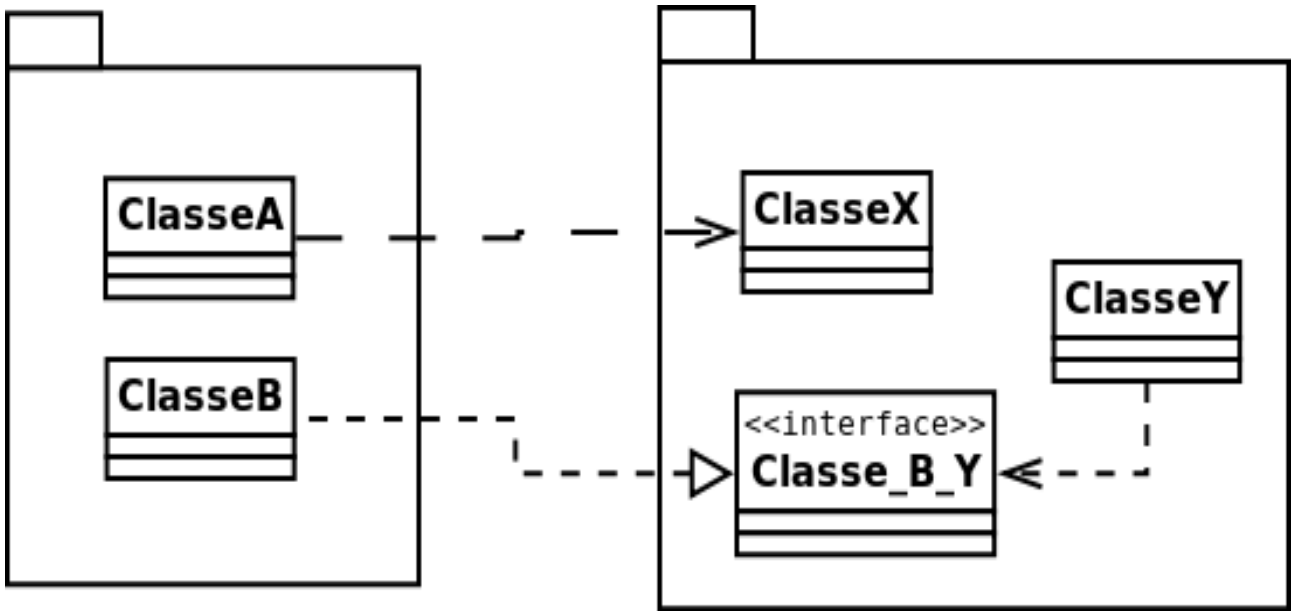
- Princípios de Arquitetura de Pacotes:
  - **ADP** – *Acyclic Dependencies principle*





Princípios de projetos OO

- Princípios de Arquitetura de Pacotes:
  - **ADP** – *Acyclic Dependencies principle*



Princípios de projetos OO

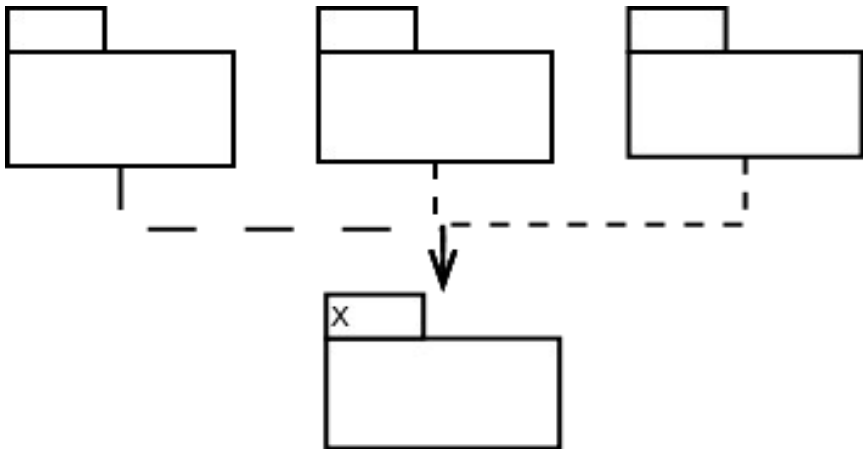
- Princípios de Arquitetura de Pacotes:
  - **SDP** – *Stable Dependencies principle*

    → *dependa na direção da estabilidade*
  - Estabilidade não significa baixa frequencia de mudanças
  - Estabilidade significa A QUANTIDADE de trabalho para fazer a mudança
  - Fatores que afetam quanto de trabalho é requerido para mudar: tamanho, complexidade, clareza, etc.
  - *Fan-in* & *fan-out*

Princípios de projetos OO

- Princípios de Arquitetura de Pacotes:
  - **SDP** – *Stable Dependencies principle*

    → *dependa na direção da estabilidade*

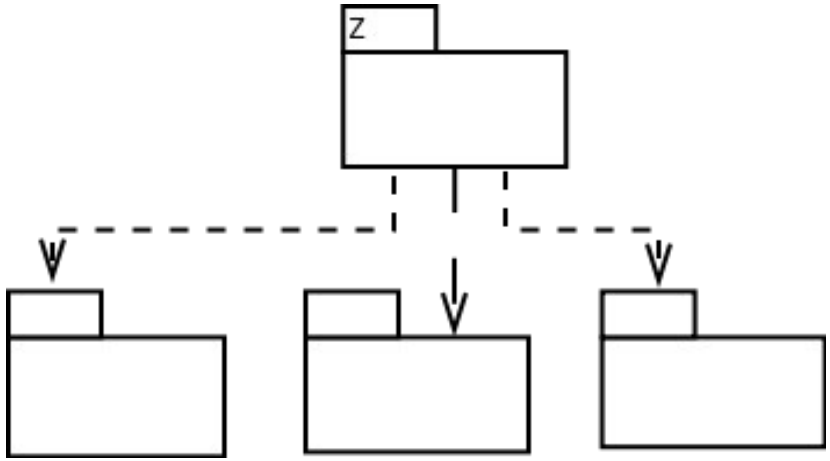


→ *X é estável*

Princípios de projetos OO

- Princípios de Arquitetura de Pacotes:
  - **SDP** – *Stable Dependencies principle*

    → *dependa na direção da estabilidade*

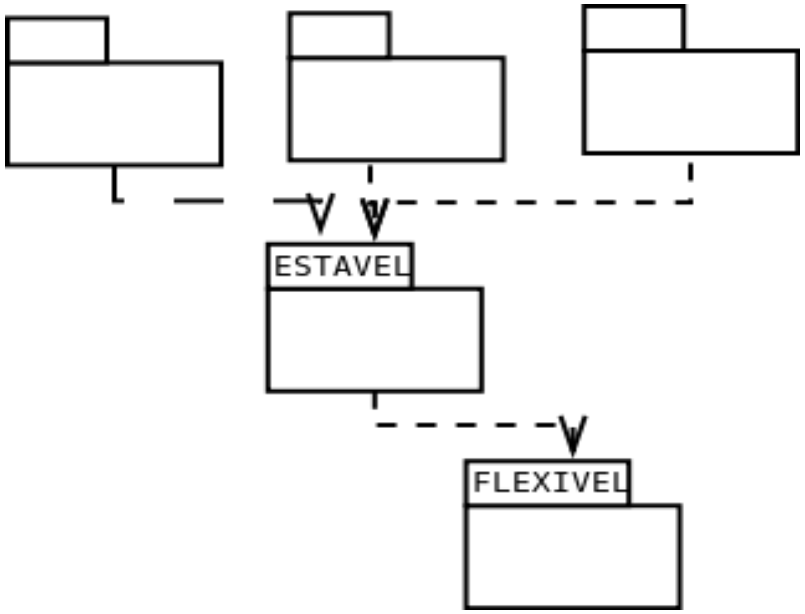


→ *Z é instável*

Princípios de projetos OO

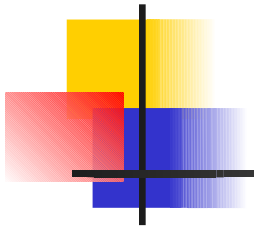
- Princípios de Arquitetura de Pacotes:
  - **SAP** – *Stable Abstractions principle*

    → *pacotes estáveis deveriam ser pacotes abstratos*



Princípios de projetos OO

- Princípios de Arquitetura OO:
  - Abstract server
  - Adapter
  - Observer
  - Bridge
  - Abstract Factory



iniciando

- Próxima Aula
  - Sortear os DP a serem apresentados nos seminários
  - Iniciar o Projeto