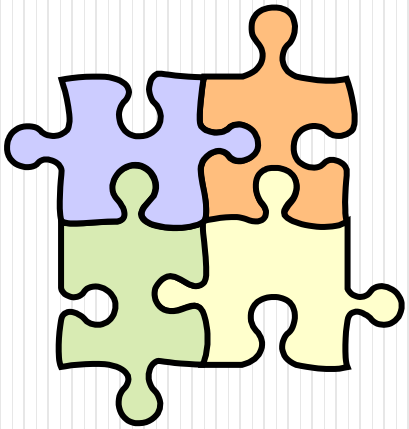
PUC Minas

PUC MINAS
ENGENHARIA DE SOFTWARE I
Profa: Luciana Mara F. Diniz

MODULARIZAÇÃO



ANÁLISE DE REQUISITOS

- Durante a análise, o objetivo é entender as funcionalidades que o sistema deve possuir sem preocupações em relação a características computacionais.

- É basicamente a transformação do conhecimento dos especialistas do domínio da aplicação (equipe) e dos clientes/usuários em um modelo que represente os requisitos/necessidades do sistema.

- Elaboração de um documento contendo os detalhes acertados (RFs, RNFs e RNs) – SRS ou ERSw.

PROJETO DE SISTEMAS

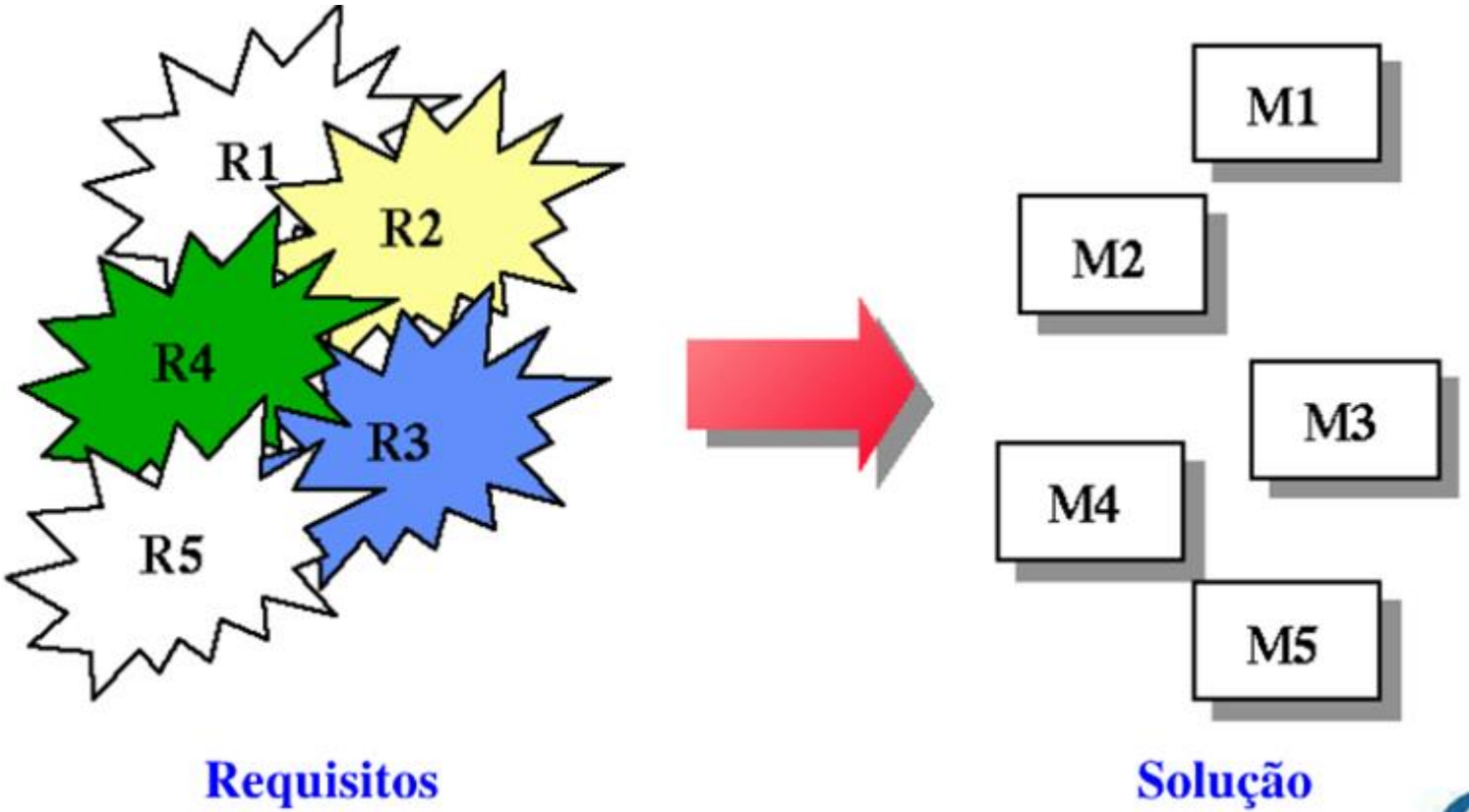
- Descreve uma solução para o problema identificado através da construção de um novo modelo, mais específico.

- Ele é parte do modelo inicial criado na análise de requisitos, introduzindo detalhes computacionais e possíveis mudanças e/ou adaptações no modelo pré-estabelecido.

ANÁLISE x PROJETO

Conjunto de requisitos descritos em uma linguagem compreensível aos clientes/usuários dos sistema, a fim de facilitar a compreensão.

Tradução em um conjunto de **módulos** passíveis de implementação.



Fonte:

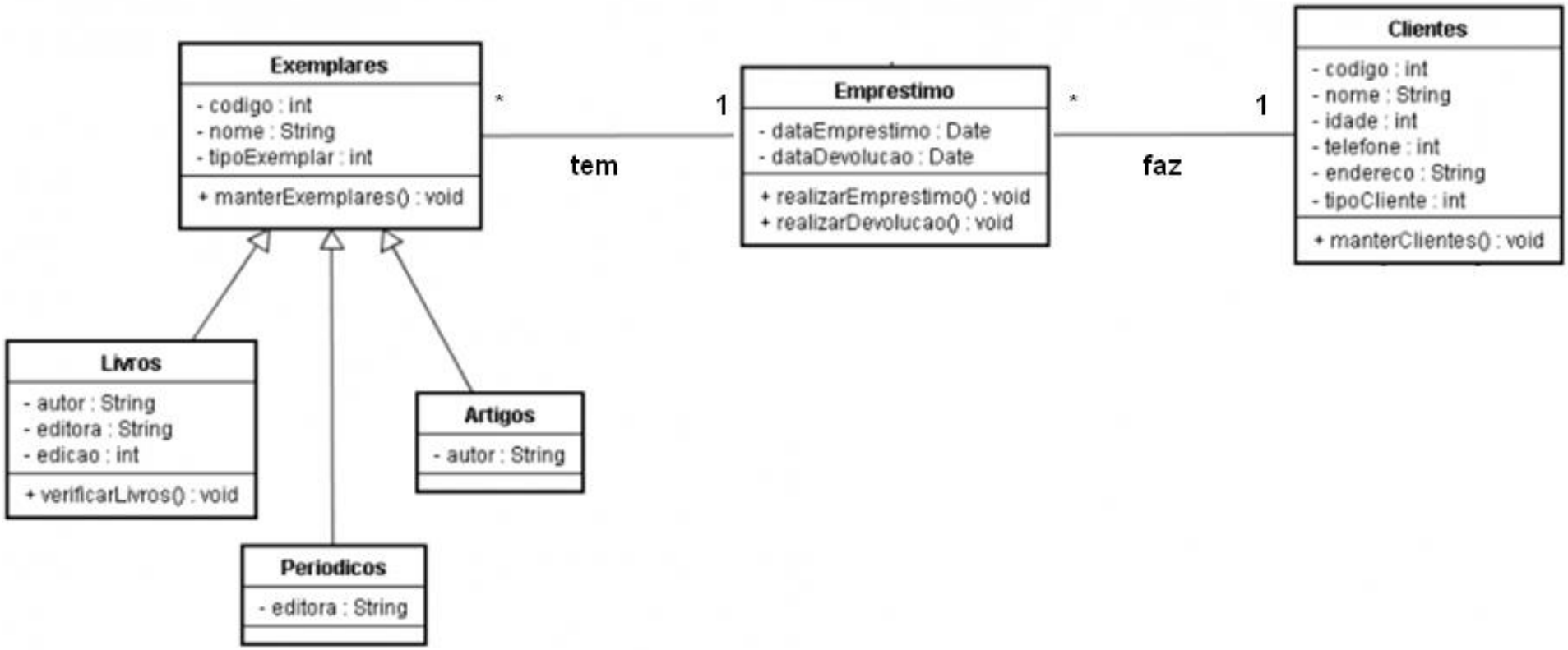
PROJETO DE SISTEMAS

- A principal característica do projeto é **a transformação** de **algo que é compreensível pelo usuário/cliente** para **algo que é compreensível pelo computador** (que posteriormente vai ser codificado/implementado em linguagem de programação).

- Esta transformação (solução) que traduz o problema inicial (geral/amplo) em algo computacional é o **objetivo da etapa de PROJETO DE SISTEMAS.**

- **O Diagrama de Classes (já visto anteriormente) faz parte desta** etapa.

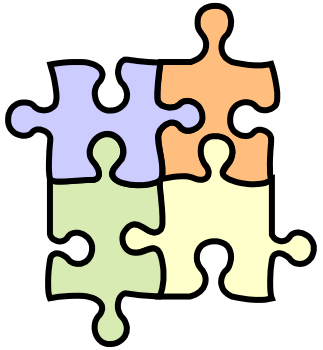
DIAGRAMA DE CLASSES DA UML
- EXEMPLO BIBLIOTECA -



MODULARIZAÇÃO

- Um **módulo** é um 'pedaço' do sistema que contém um conjunto de componentes bem definidos de um sistema: podem conter rotinas, classes, definições de tipos, etc.
  - Ex.: No desenvolvimento OO (orientado a objetos), um módulo se constitui por uma ou mais classes (sendo uma principal e outras acessórias).
- Fazendo-se uma analogia, podemos pensar um **módulo como uma peça** e, assim como os vários módulos compõem o software, as peças irão compor o quebra-cabeça (que seria o software como um todo).

MODULARIZAÇÃO



- **OBJETIVO:** facilitar o desenvolvimento do sistema, a manutenção e a reutilização de código!

- **MÓDULOS** são construídos com base em partes bem definidas do sistema.

- Funções independentes, partes independentes podem ser separadas (com base na implementação da lógica – usando linguagens de programação específicas) para posterior integração.

EXEMPLO 1 - MÓDULOS

Recursos humanos

Financeiro

Locação

Compras

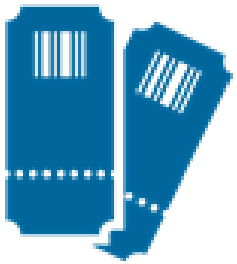
Comercial

Orçamentário

Cadastro

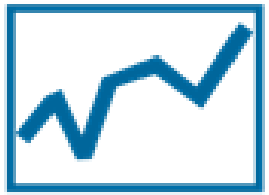
Fiscal

Projetos



Cartão de ponto

Controle de eventos



Faturamento

EXEMPLO 2
MÓDULOS por DEPARTAMENTO

DIAGRAMA DE PACOTES

SÍMBOLOS

- Diagrama da UML bastante útil para a modelagem de subsistemas e subdivisões da arquitetura de uma linguagem.

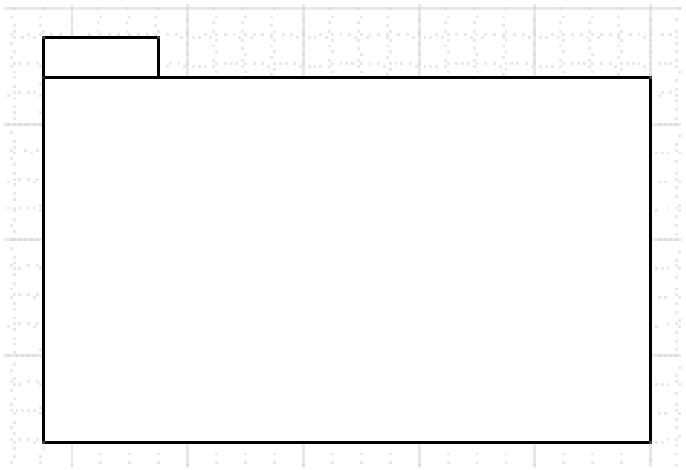
- Representa também submódulos englobados por um sistema.

- **Dependência**

DIAGRAMA DE PACOTES

SÍMBOLOS

- Pacote



- Dependência

DIAGRAMA DE PACOTES

- **EXEMPLO**

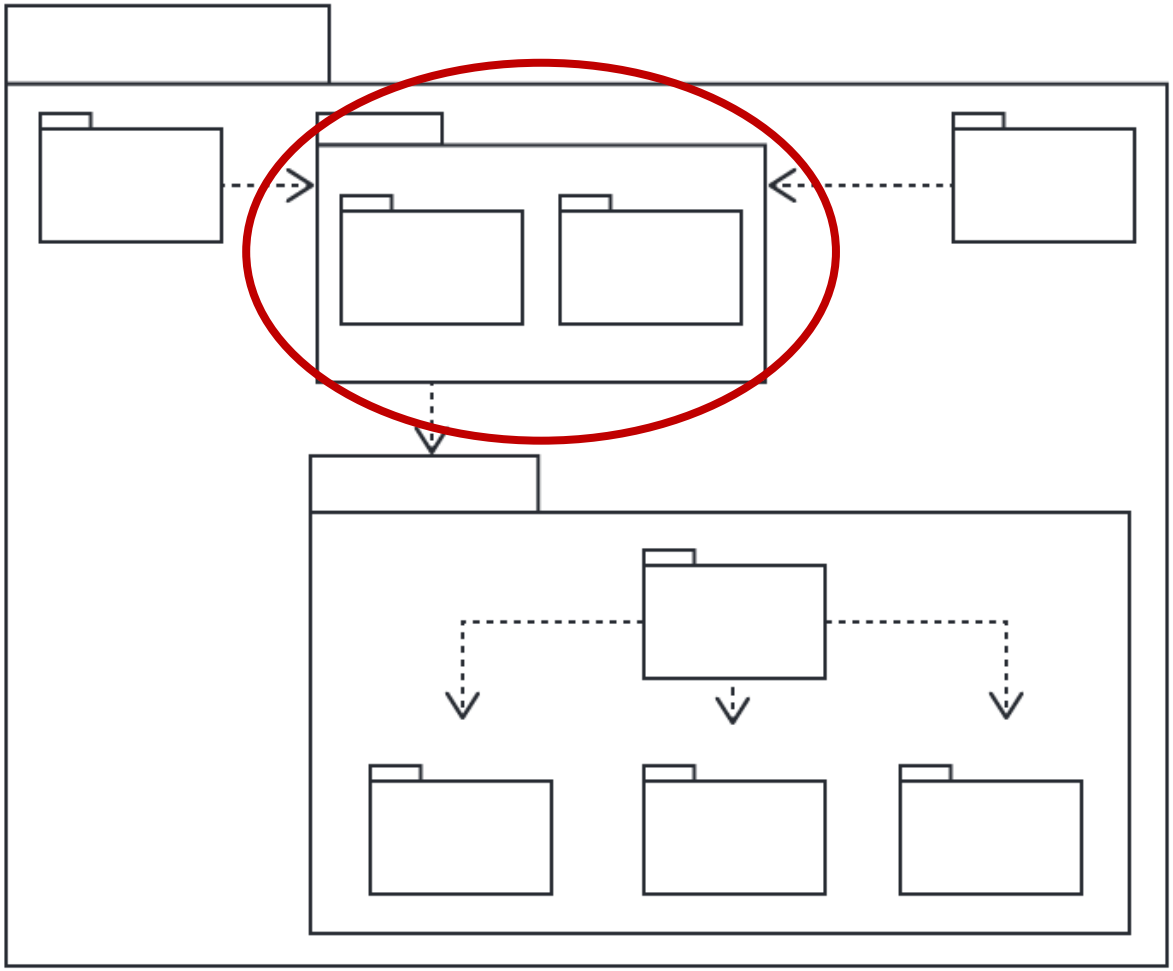
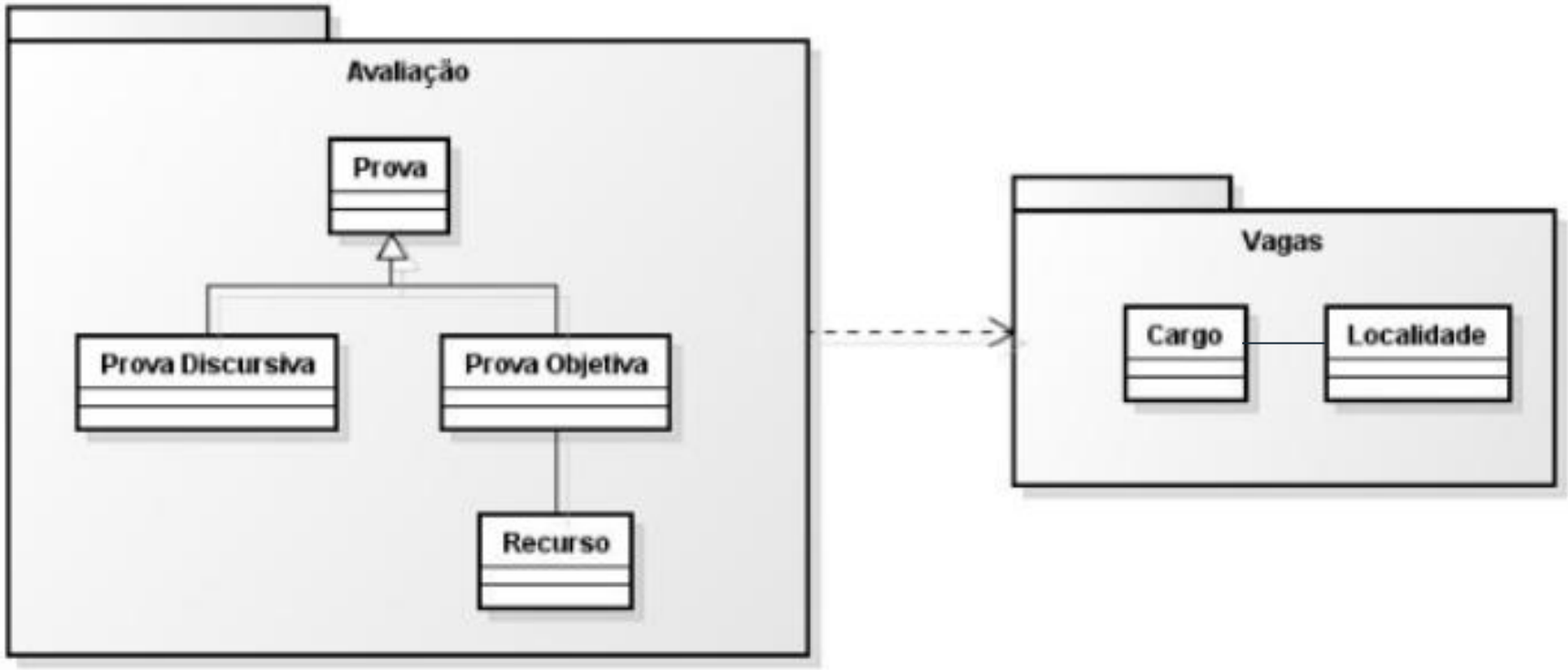
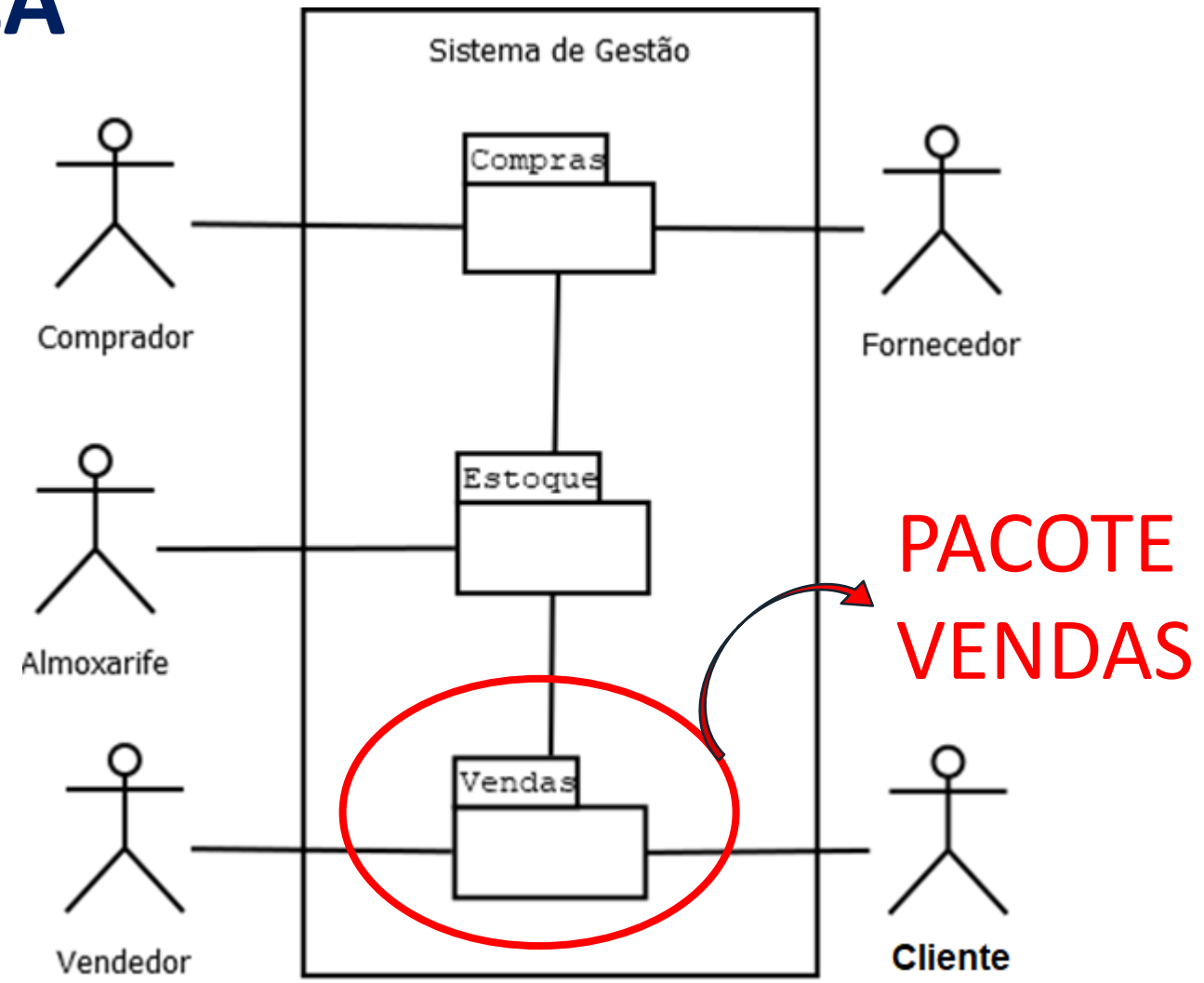


DIAGRAMA DE PACOTES

- **EXEMPLO - CONCURSO**

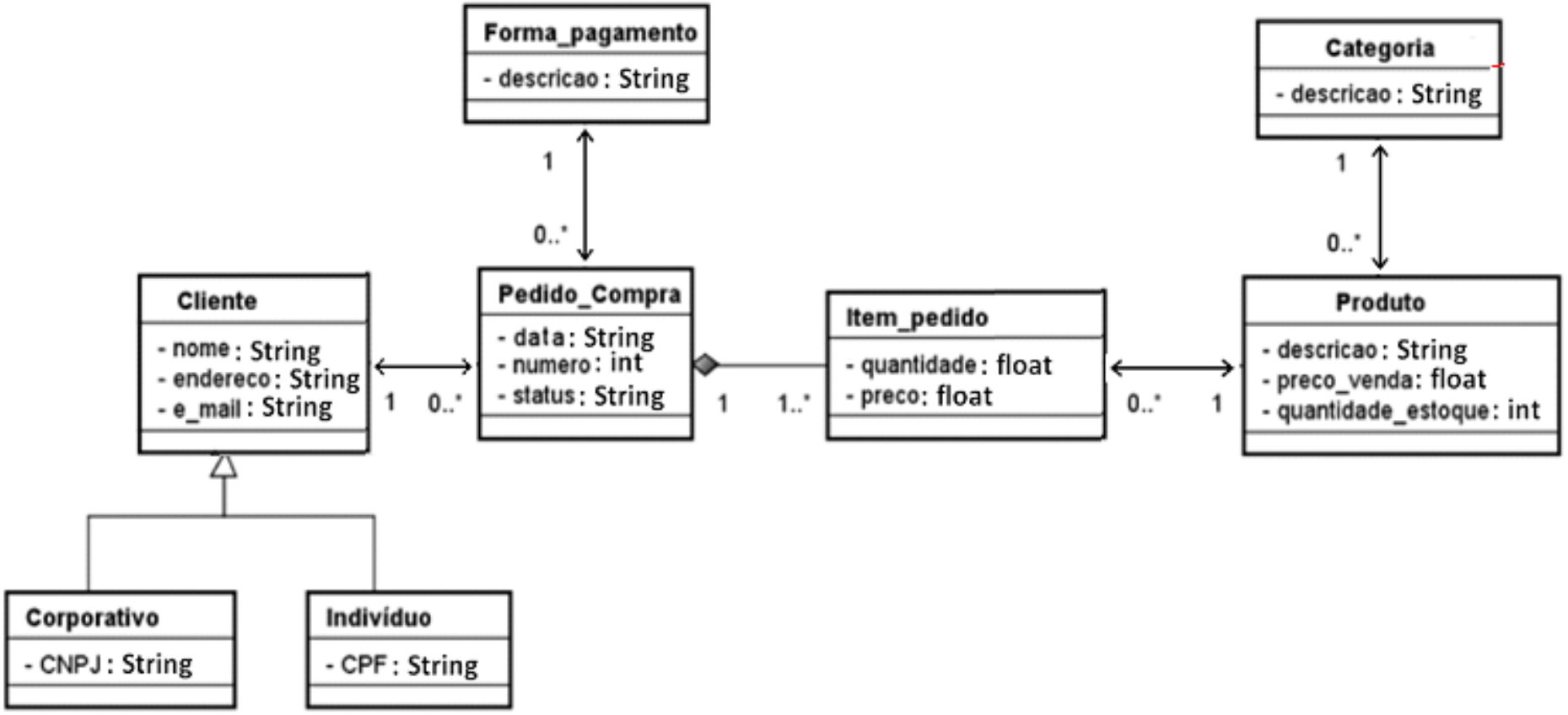


PACOTES
LOGÍSTICA

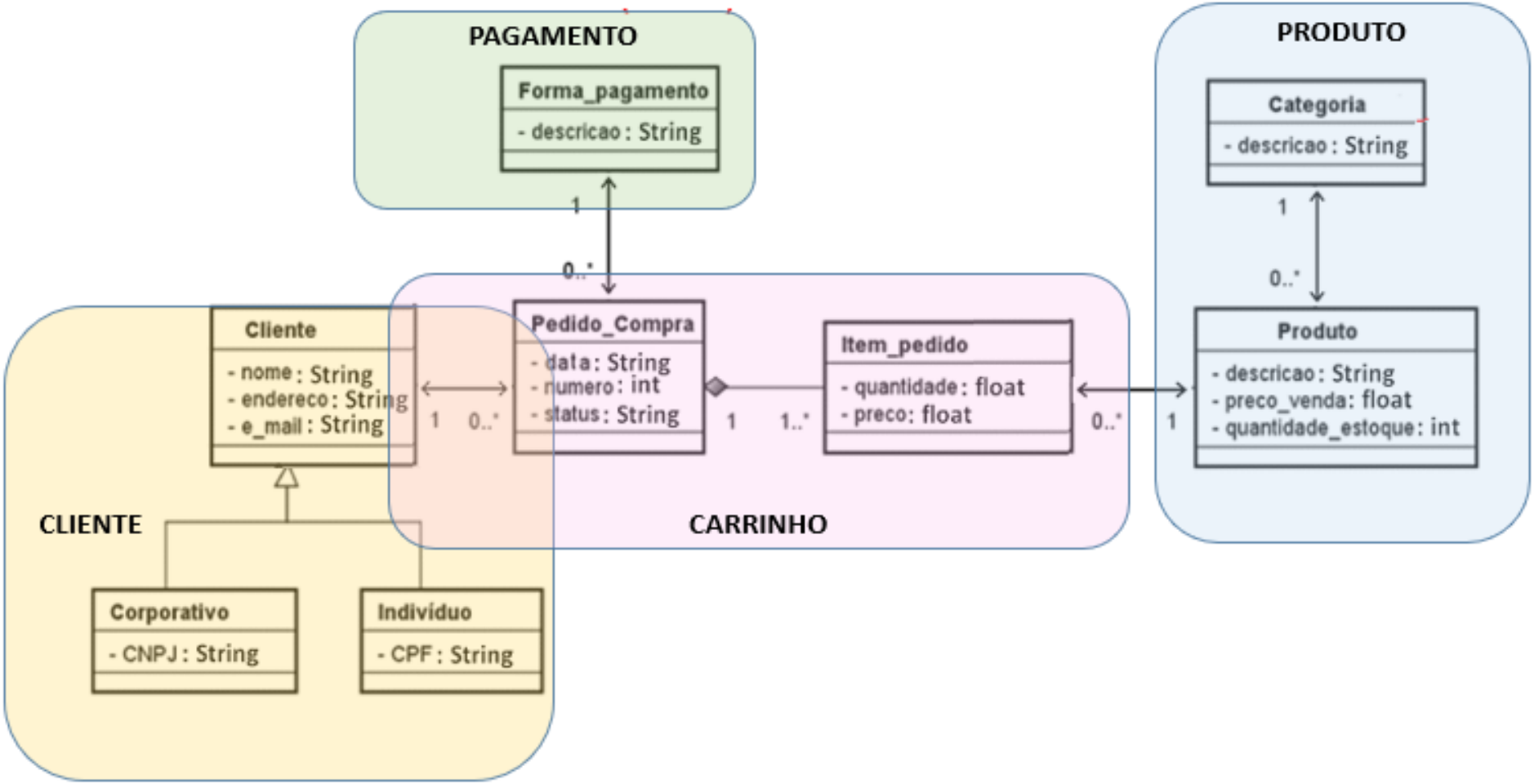


EXEMPLO
SISTEMA e-COMMERCE - VENDAS

DIAGRAMA SUGESTIVO DE RESPOSTA (ENADE):

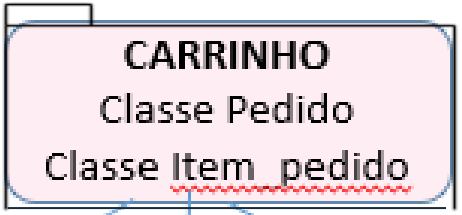


EXEMPLO
MODULARIZAÇÃO - SISTEMA e-COMMERCE

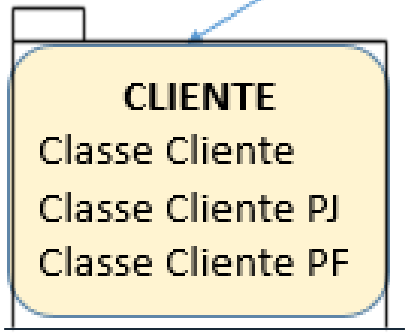


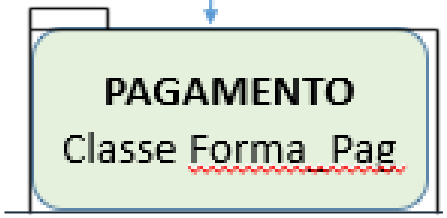
EXEMPLO
MODULARIZAÇÃO - SISTEMA e-COMMERCE

NÍVEL 1



NÍVEL 2



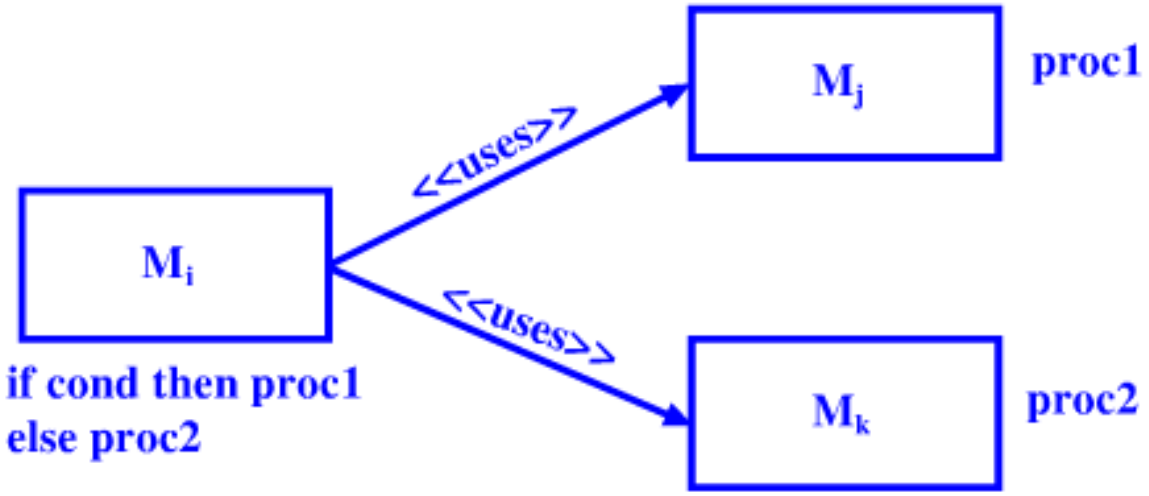


SE EXISTISSEM

NÍVEL 3

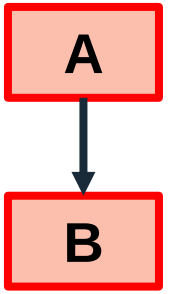
MODULARIZAÇÃO

- **Relação de uso entre módulos**: pra um módulo funcionar é possível que ele utilize um recurso de outro módulo (retorno de métodos, p.ex.).
  - **Seta**: sai do módulo que usa para o módulo que é usado (exemplos de modularização e generalização, aula anterior).



  - Módulo Mi usa o módulo Mj. Mi depende de Mj e Mk.

MODULARIZAÇÃO



- Na **modularização**, quando há uma **relação de uso** entre módulos, um módulo pode ser visto como cliente e o outro como fornecedor.

- O comportamento de um módulo 'cliente' deve ser compreensível sem a análise detalhada de seus 'fornecedores' (encapsulamento/abstração). **Como** está implementado, não interessa a quem **usa** os objetos (instâncias das classes).

- **Princípio fundamental**: se eu tenho um módulo B que fornece um serviço para o módulo A, eu não preciso saber como ele executa tal ação.
  - O módulo A deve ser implementado sem que seja necessário conhecer como o módulo B foi feito (só precisará do valor retornado, fornecido) → ASSINATURA.

MODULARIZAÇÃO

- **Implementação do módulo:** código que realiza determinado serviço. Serviço ofertado para o resto do sistema através de uma **interface....**
  → Conceito da OO.

- **Interface do módulo**: conjunto de serviços oferecidos por um módulo a seus clientes, ou seja, **assinaturas dos métodos** (funções) que implementam determinado tipo de serviço.

- **Possuem**: tipo de retorno, nome do método (função), parâmetros recebidos!

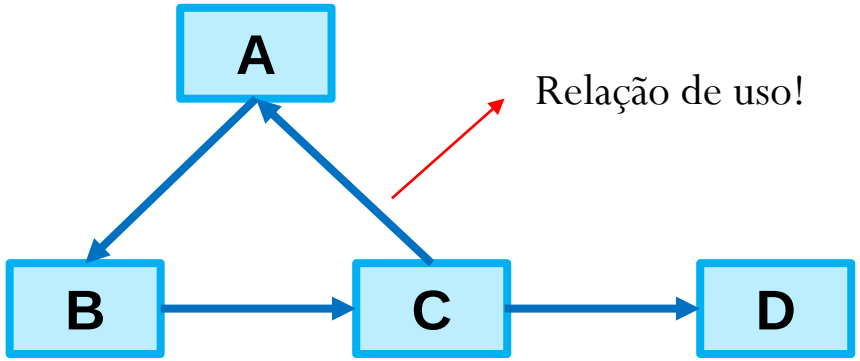
```
double calcula_media (int a, int b);
```

MODULARIZAÇÃO

- Implementação: código que implementa métodos.
- Interface: assinatura de métodos públicos dessa classe (princípio de ocultamento de informações).
- Manutenção/ mudança → o que importa para o sistema são as assinaturas dos métodos, no caso de uso de determinado retorno de método, etc.
- **Desenvolver interfaces fortes, para quando vier alguma mudança, mude a implementação, mas não a assinatura. O resto do sistema fica "blindado", pois não 'conhece' detalhes (PADRÃO DE PROJETOS – CLEAN CODE)**

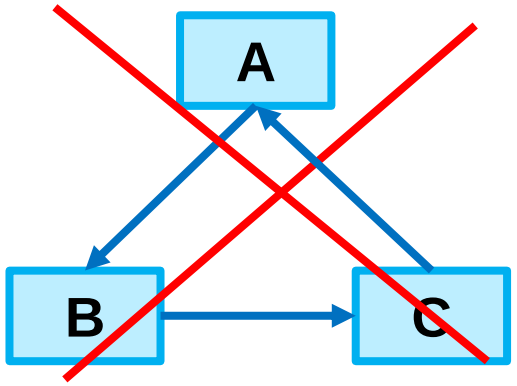
MODULARIZAÇÃO

- **TÉCNICA DE MODULARIZAÇÃO**: reduzir relações de usos entre módulos para que um **grafo (que contenha ciclos)** vire uma **árvore (que contém hierarquia).**



- **Princípio geral**:
  - Se o grafo não for reduzido a uma árvore, podemos ter um sistema onde "nada funcione até que tudo funcione".

MODULARIZAÇÃO

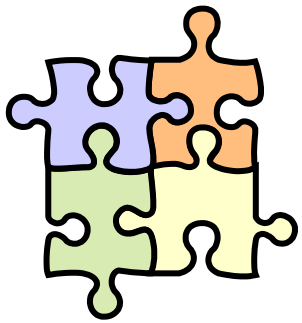


- Numa estrutura de módulos, devemos sempre buscar estrutura de uma **árvore**, isto é, devemos evitar ciclos (grafos).

- Em **estruturas de árvores há uma hierarquia**, facilitando os testes e integrações.

- **Ciclos são problemáticos**: Modulo A depende do Modulo B que depende do Modulo C que depende do modulo A.
  - Como eu testo o modulo A?
  - Pra eu ter o A correto tenho q ter o B e o C... e por ai vai...
  - Tenho que ter tudo proto pra testar. Não posso testar por partes.

MODULARIZAÇÃO



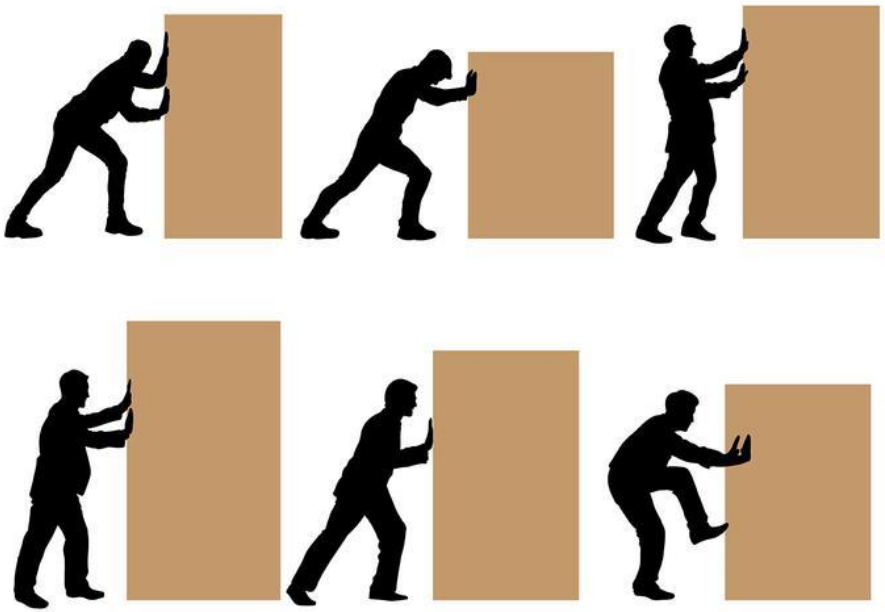
- Tem como embasamento a estratégia **"dividir para conquistar" ou "DeC - Divisão e Conquista" na programação**: consiste na definição que problemas complexos devem ser divididos em problemas menores e mais simples (**mergsort – algoritmo de ordenação**)

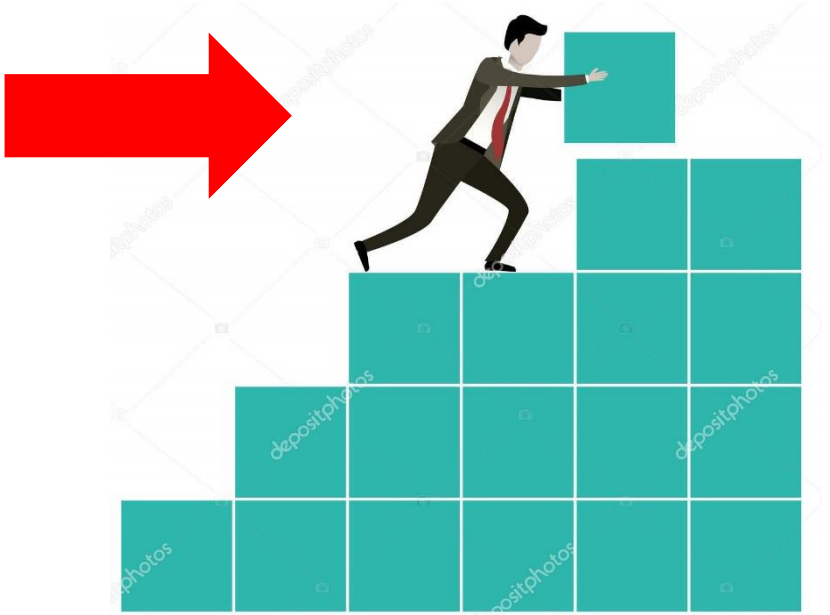
A técnica de divisão e conquista consistem de 3 passos básicos:

1. **Divisão:** Dividir o problema original, em subproblemas menores.
2. **Conquista:** Resolver cada subproblema recursivamente.
3. **Combinação:** Combinar as soluções encontradas, compondo uma solução para o problema original.

MODULARIZAÇÃO

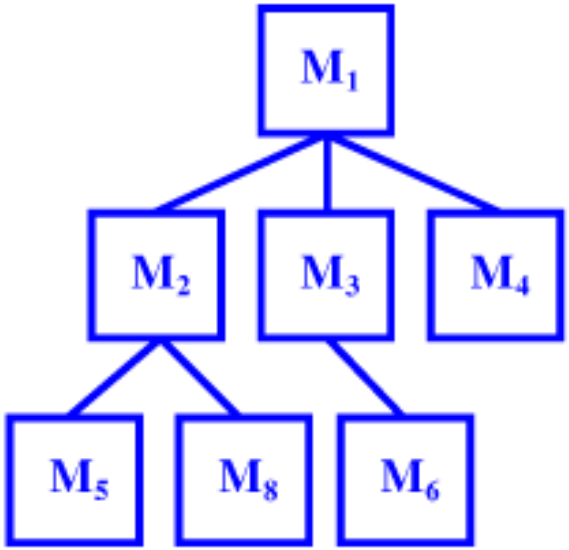
- **“DIVIDIR PARA CONQUISTAR” EM OUTRAS PALAVRAS...**





MODULARIZAÇÃO

- Diagrama apresenta relações de uso entre os módulos.
- Pode não representar ordem de uso (setas), apenas suas dependências.



PROPRIEDADES

→ **MÉTRICAS UTEIS NA AVALIAÇÃO DE UM PROJETO DE SISTEMA:**

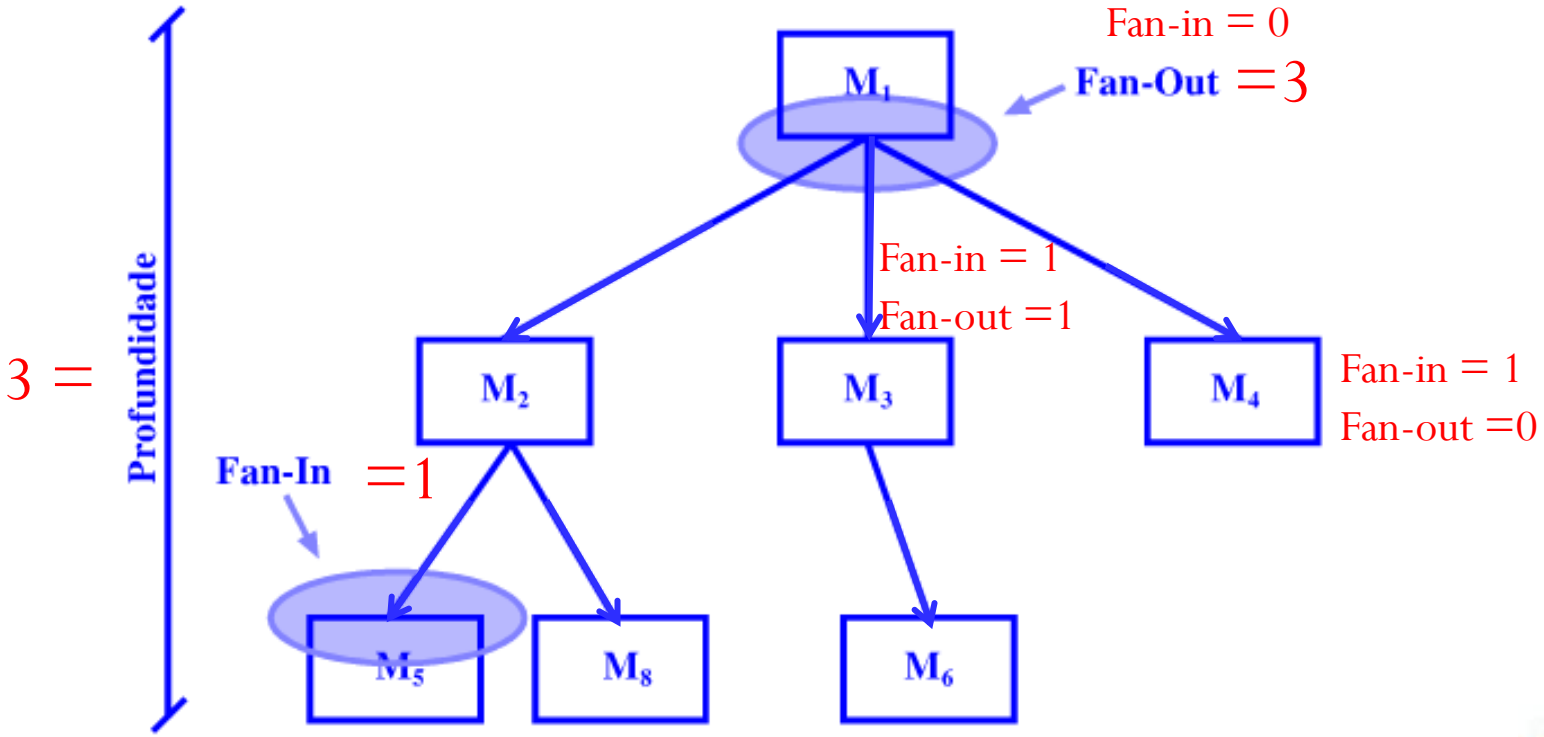
- **1. PROFUNDIDADE**: quantos níveis eu tenho da raiz até a ponta mais baixa da árvore - **hierarquia (número de níveis).**

- **2. FAN-OUT**: indica a quantidade de módulos que determinado módulo utiliza, ou seja, com quem ele tem relações de uso.
  - OBS.: Um módulo deve ter o mínimo possível de relações com outros módulos.

- **3. FAN-IN**: quantos módulos utilizam determinado módulo.
  - OBS.: Um módulo deve ter o mínimo possível de relações com outros módulos.

PROPRIEDADES

- **FAN-OUT**: número de saídas que o módulo tem.
  FAN-IN: número de entradas que o módulo tem.
  PROFUNDIDADE: número de níveis



Fonte:

PROPRIEDADES

ATENÇÃO!

- **IDEAL:** baixos Fan-in e Fan-out.

- **No diagrama de classes**: se uma classe depende de muitas outras, ou se uma classe é muito utilizada por outras, pode ser necessário a reestruturação da arquitetura do sistema!

- **IMPORTANTE:** não que estejam errados, mas pensar se é a melhor solução de projeto em questão de desempenho, reutilização de código e manutenção.

PROPRIEDADES
Coesão

- **Coesão**: 'força' que uma tarefa é executada dentro de um módulo.
  - Ao olhar para as partes componentes do módulo, devemos identificar se elas estão ali por acaso ou se elas estão muito relacionadas a uma mesma tarefa.

- **Ideal:** Módulos tenham coesão alta, isto é, os componentes que os compõem possuem relações fortes e cada qual cumprindo bem o seu papel.

- **RESUMO**: a coesão 'olha pra dentro' do módulo e verifica se as partes estão relacionadas a um mesmo objetivo.

PROPRIEDADES

Coesão

- **EXEMPLO: c**onsidere um sistema de *e-commerce* que possua as classes CLIENTE e VENDA, e que a classe CLIENTE esteja implementada assim:

```
1  public class Cliente {
2
3      private String name;
4      private String CPF;
5      private String endereco;;
6
7      //outros atributos ...
8      //construtor
9
10     public double CalcDesconto(Venda venda){
11
12         double x = 0;
13
14         //descontos à vista...
15         // atribuir valor à variável x
16
17     return x;
18     }
19 }
```

PROPRIEDADES
Coesão

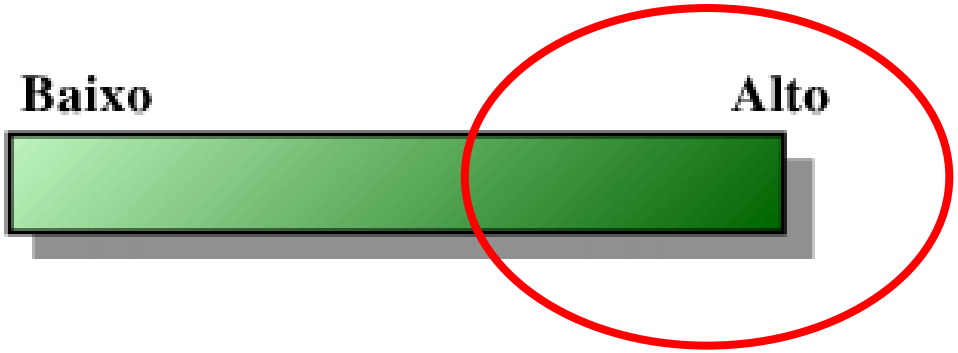
- **EXEMPLO:**
- O método CalcDesconto() não precisa de nenhum dado do cliente e sim da venda!
- Neste caso, este método não faz sentido estar dentro da classe CLIENTE.
- Caso continue dessa forma, **possuirá baixa coesão**, o que compromete os princípios do projeto de sistemas.
- A responsabilidade da classe CLIENTE é manter os dados relacionados que façam sentido para esta classe; da mesma maneira para a classe VENDA.

PROPRIEDADES

Coesão

- **IDEAL:**

  Cada módulo deve ter um objetivo muito bem definido.



PROPRIEDADES
Acoplamento

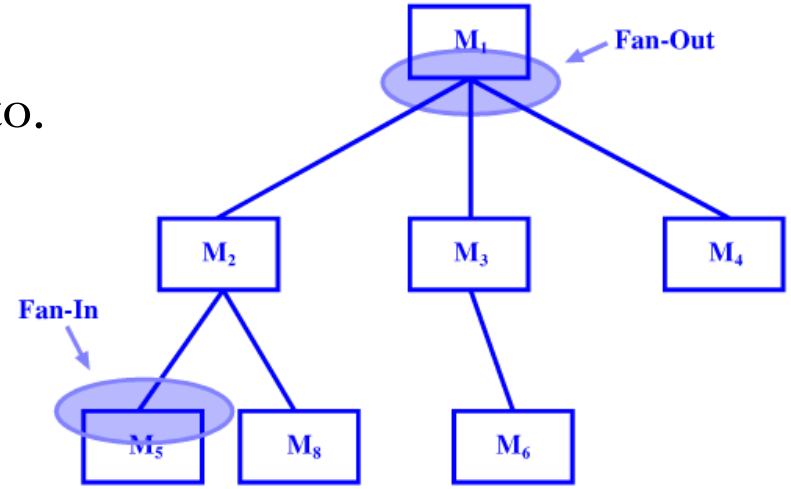
- **Acoplamento:** medida de interconexão entre os módulos do sistema.
  - É medido entre dois módulos, definindo o grau de acoplamento entre eles (diretamente relacionado com FAN-IN e FAN-OUT).

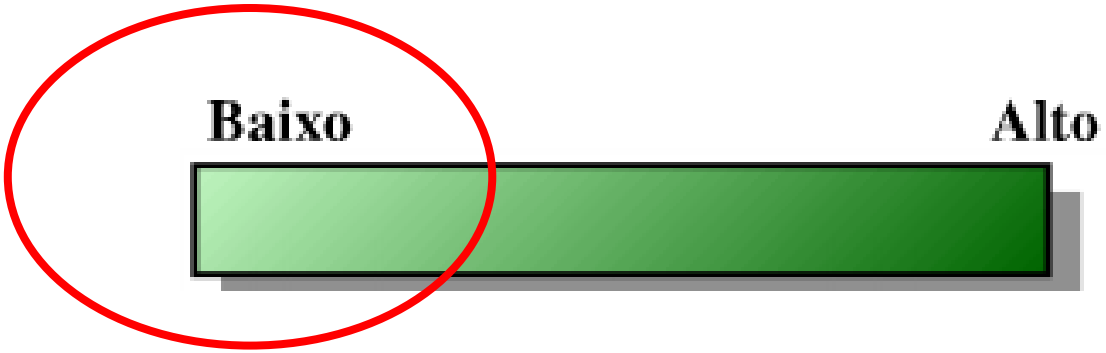
RESUMO: no acoplamento "olhamos para fora" do módulo e verificamos como este módulo se relaciona com os demais.

PROPRIEDADES
Acoplamento

- **IDEAL:**

  Módulos devem ter baixo acoplamento.





PRATICANDO...

1. (BNDES – 2005 – Análise De Sistemas – Desenvolvimento – Questão 40 - adaptada) A tabela abaixo mostra os pares da relação de uso (*USES*) entre os módulos de um sistema de informação:

Módulo	Usa Módulo
1	2
1	3
2	4
3	5

I - Esta relação não forma uma hierarquia.
II - O módulo 1 possui *fan-in* igual a 0 e *fan-out* igual a 2.
III - A profundidade do módulo 3 é igual a 3.

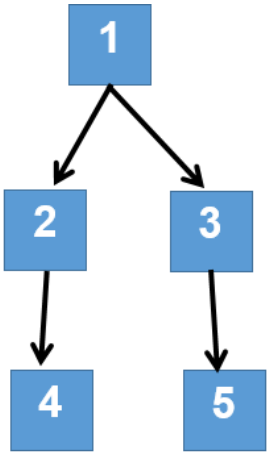
- As assertivas CORRETAS são:

  (A) somente I; (B) somente II; (C) somente III;
  (D) somente I e II; (E) I, II e III.

PRATICANDO... resposta

1. (BNDES – 2005 – Análise De Sistemas – Desenvolvimento – Questão 40 - adaptada) A tabela abaixo mostra os pares da relação de uso (*USES*) entre os módulos de um sistema de informação:

Módulo	Usa Módulo
1	2
1	3
2	4
3	5



I - Esta relação não forma uma hierarquia. (Falso, forma pois não tem ciclos)
II - O módulo 1 possui *fan-in* igual a 0 e *fan-out* igual a 2. (V)
III - A profundidade do módulo 3 é igual a 3. (Falso, módulo não tem profundidade)
As assertivas CORRETAS são:

(A) somente I; **(B) somente II;** (C) somente III;
(D) somente I e II; (E) I, II e III.

PRATICANDO...

2. (CESGRANRIO – 2011 – Petrobras - Analista de Sistemas Jr). No contexto de qualidade de software, **coesão e acoplamento** são medidas:

a) intramodulares, sendo a primeira inversamente proporcional, e a segunda proporcional à qualidade.

b) intramodulares e diretamente proporcionais à qualidade.

c) intramodular e intermodular, respectivamente, sendo a primeira proporcional, e a segunda inversamente proporcional à qualidade.

d) intermodulares e diretamente proporcionais à qualidade.

e) intermodular e intramodular, respectivamente, sendo a primeira proporcional, e a segunda inversamente proporcional à qualidade.

PRATICANDO... resposta

2. (CESGRANRIO – 2011 – Petrobras - Analista de Sistemas Jr). No contexto de qualidade de software, **coesão e acoplamento** são medidas:

INTRA = DENTRO; INTER = ENTRE

Diretamente proporcional = um aumenta, outro aumenta

Inversamente proporcional = um aumenta, outro diminuiu OU um diminui, outro aumenta

a) intramodulares, sendo a primeira inversamente proporcional, e a segunda proporcional à qualidade.

b) intramodulares e diretamente proporcionais à qualidade.

c) intramodular e intermodular, respectivamente, sendo a primeira proporcional, e a segunda inversamente proporcional à qualidade.

d) intermodulares e diretamente proporcionais à qualidade.

e) intermodular e intramodular, respectivamente, sendo a primeira proporcional, e a segunda inversamente proporcional à qualidade.

REFERÊNCIAS BIBLIOGRÁFICAS

- Material de aula do Prof. Márcio Barros – Fundação CECIERJ/Consórcio CEDERJ

- Especificação e projeto do sistema de gerência e manipulação de mídias para produção de aulas. Disponível em: http://slideplayer.com.br/slide/1219965/