13.03.19

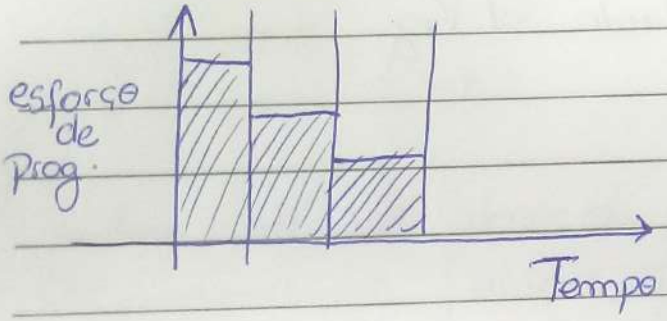
Introdução

Vantagens da Programação Modular (PM)

Problemas → Problemininhos

- Vencer barreiras de complexidade
  → "dividir e conquistar"
- distribuir tarefas em grupo
  ↳ Cuidado ▽ a gente tem que se organizar para decidir os módulos pequenos

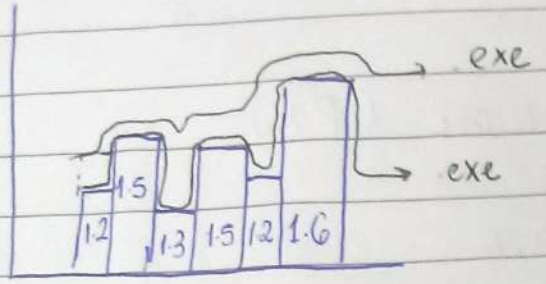
- reuso
  → Se eu Programo uma Pilha, Posso usar em outra aplicação
  ex: módulo Tabuleiro de damas serve para Xadrez.
  não precisa reinventar a roda



- Criação de um Acervo de módulos reutilizáveis
  É dentro do mesmo nicho, se você trabalha com pagamento vai ter um módulo de pagamento

- Permite trabalhar com baselines de módulos já testados

13.03.19



daí para Voltar atrás e Sabe onde foi o erro

- desenvolvimento incremental
  Criou um módulo? Testa } Criou o conjunto? Testa
  Criou outro? Testa

Separado

- Aprimoramento individual
- Reduz o Tempo de compilação.
  - Só preciso compilar tudo 1 Vez.

Princípios de Modularidade

1) Módulo
   - física: Unidade de Compilação independente
   - lógica: Possui um único conceito
     ↳ Se tiver muitos, na hora de consertar dá problema

13.03.19

Versionado em uma aplicação.

3) interface

M_1 — M_2

mecanismo de troca de dados, comandos e eventos entre elementos de programa.

dif de comando e evento? evento → "Sem conexão" "arquivo vazio"

Comando →

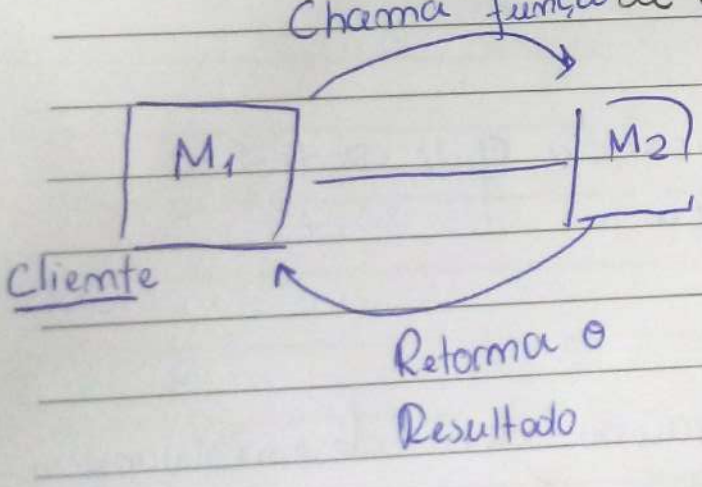
A interface sempre ocorre entre elementos de pogama do mesmo nível na hierarquia.

ex: Arquivo entre Sistemas
Funções entre módulos de acesso e seus Parâmetros

Variáveis globais entre blocos de código

Variáveis globais entre linhas? Sim

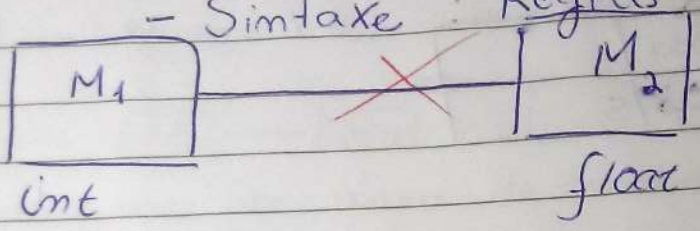
Relacionamento Cliente ↔ Servidor



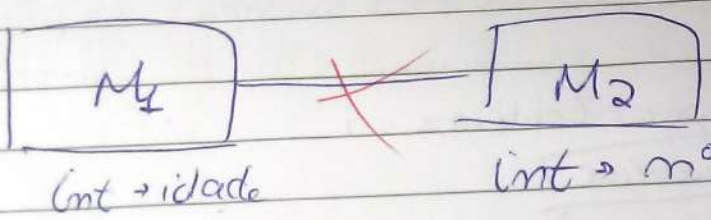
18.03.19

- interface em detalhe

- Sintaxe: Regras



- Semântica: Significado



Exemplo de interface

interface explícita → Assinatura ou protótipo

tpDadosAlunos * obterAluno (int id)

- interface esperada pelo cliente

ponteiro válido dado referenciando os dados de aluno

Ou null

- interface esperada pelo Servidor:

- id válido

- Acesso aos dados de aluno → interface implícita se o mesmo existir

- interface esperada por ambos.

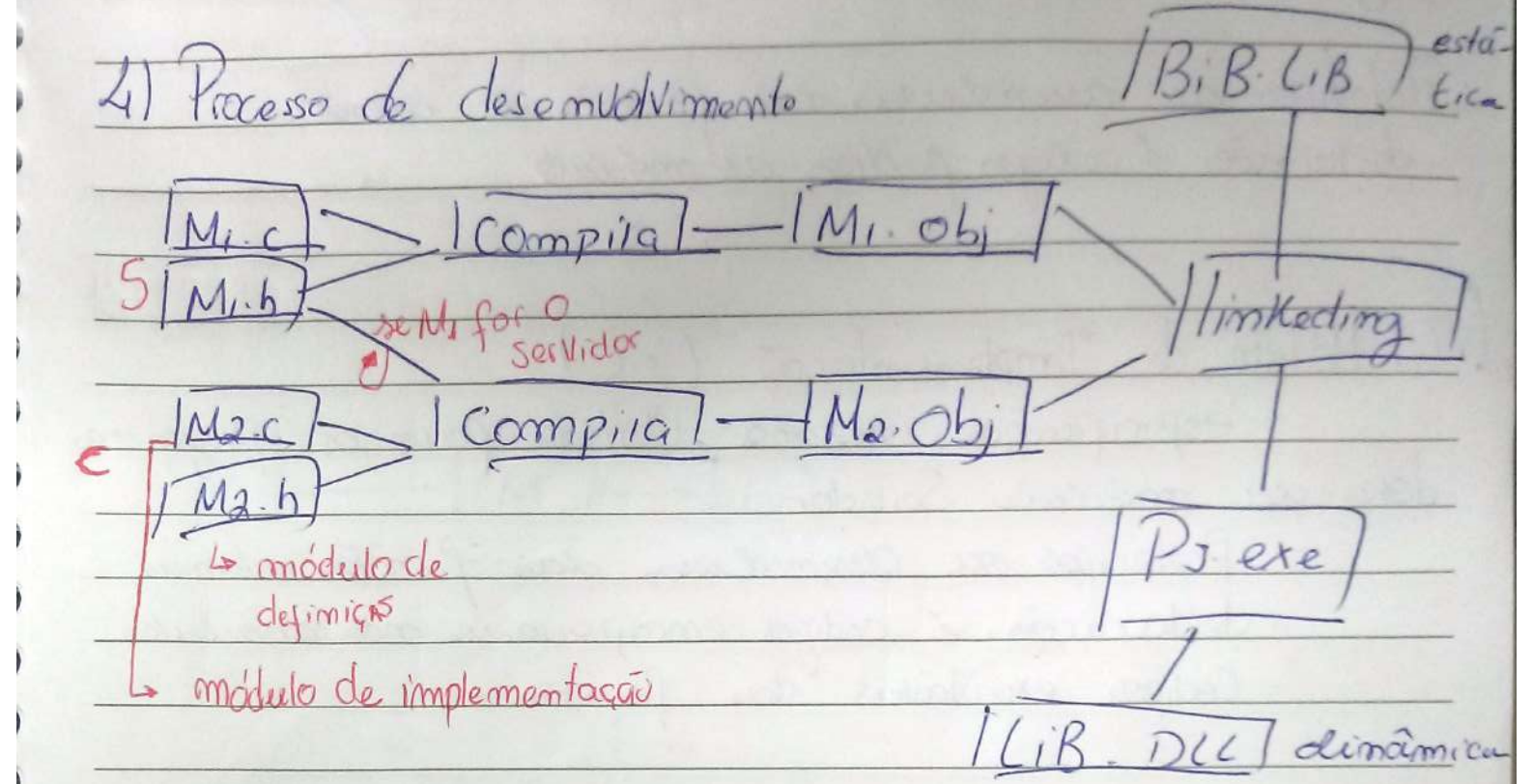
- tpDadosAluno → interface fornecida por 3º.

18.03.19

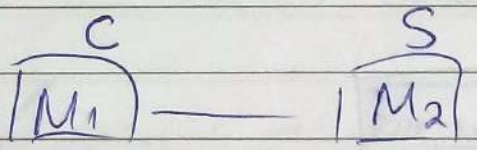
↳ na doc de cada funçaõ

Observação: Protocolo de uso: forma de utilizar os itens que compõem uma interface para que essa possa ser operada corretamente

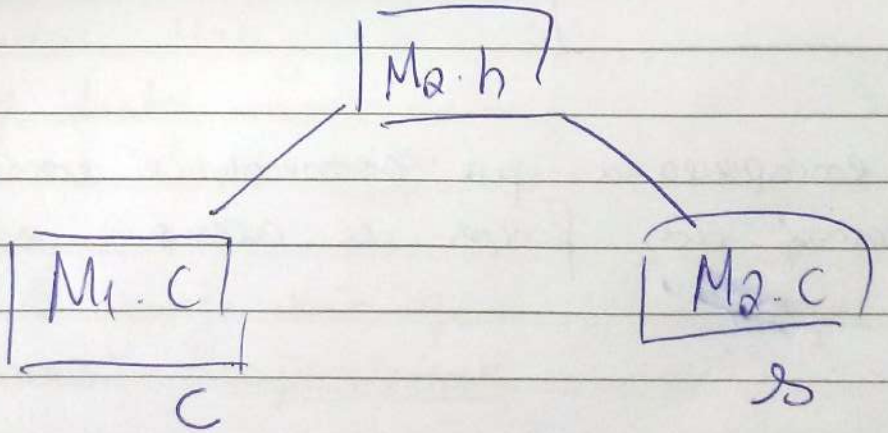
4) Processo de desenvolvimento



no passado:



agora:



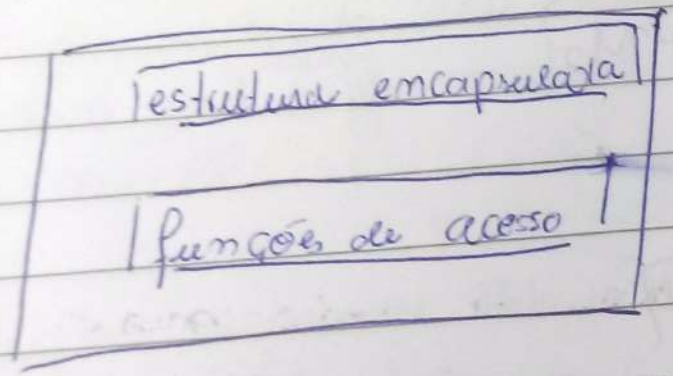
18.03.19

5) Módulo de Definição (.h)
- especificação externa voltada para os programadores do módulo Cliente

- Protótipos ou assinaturas das funções de acesso
- declarações e códigos públicos do módulo

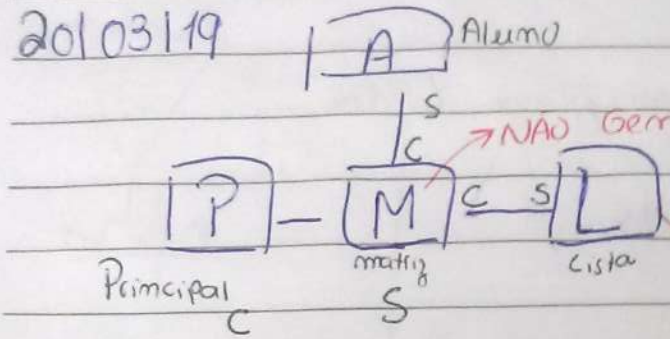
6) Módulo de Implementação (.c)
- especificação interna voltada para os programadores do módulo Servidor
- Protótipos ou assinaturas das funções internas
- declarações e códigos encapsulados nos módulos
- Códigos executáveis das funções

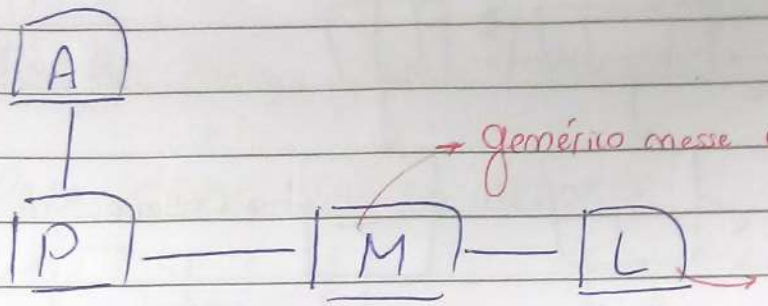
7) Tipo Abstrato de Dados (TAD)

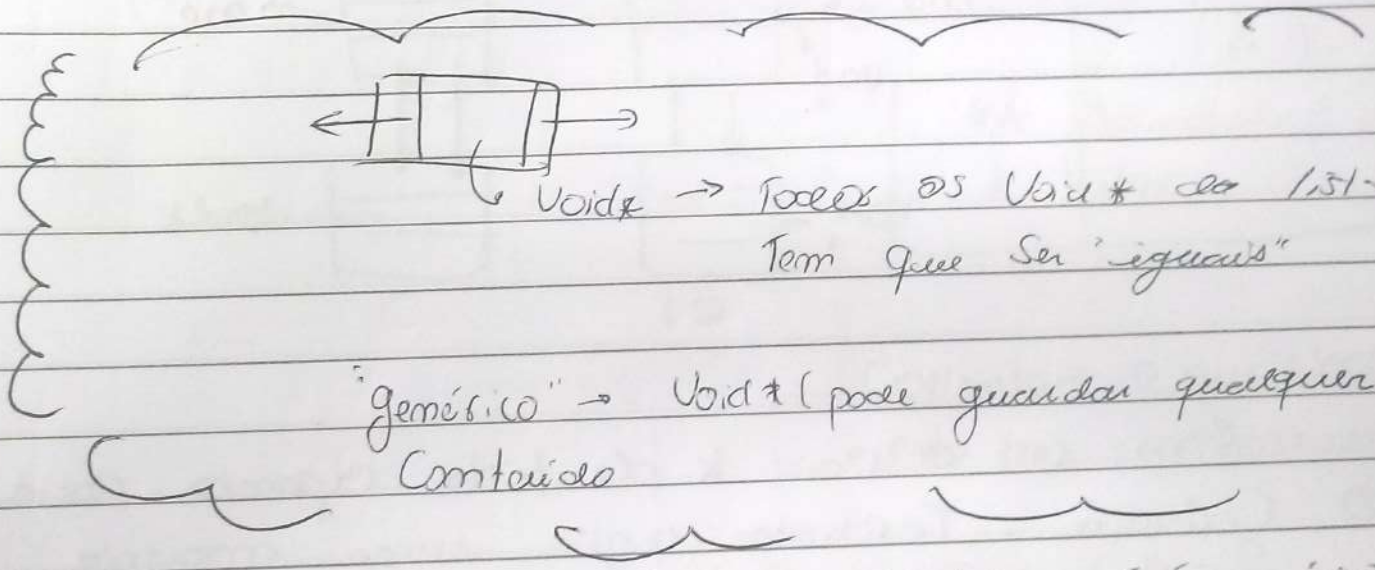


é a estrutura encapsulada que somente é conhecida pelos clientes através das funções de acesso.

20/03/19



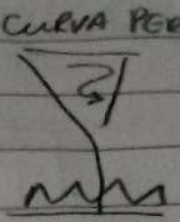




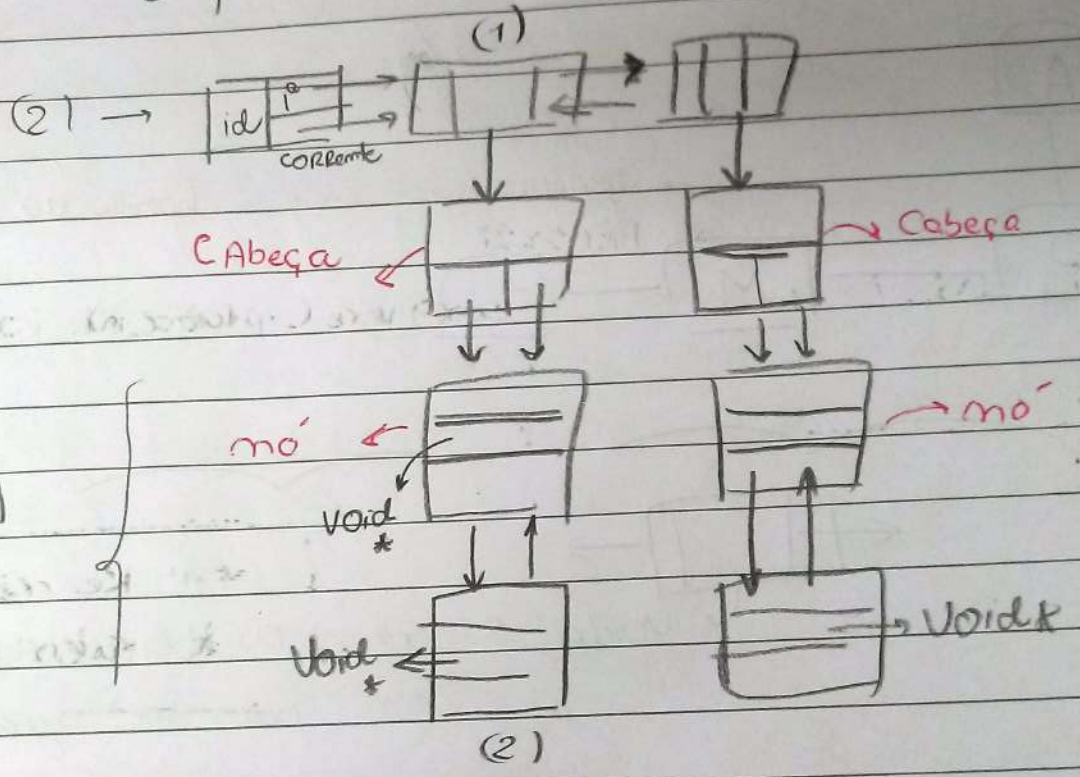
Contexto 1: Matriz de lista → Matriz não é genérico.
Se lista usar a definição de Matriz, lista não é
Caso lista não usa matriz → lista fica ge-
nérico.

A gente não quer Criar específicos porque
queremos Reaproveitar

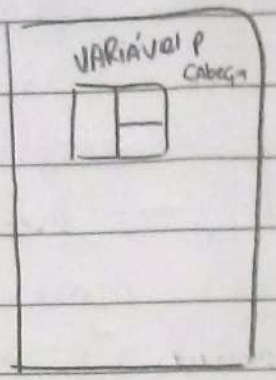
Cuidado: Matriz que armazena Lista ≠ Matriz de Lista



Matriz feita de Lista ⇝ genérica



em (1) o Void * de lista aponta para o Cabeça. Teríamos aqui uma matriz 2 colunas (2) e cada coluna com 2 linhas (4)

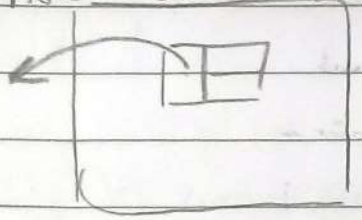


Módulo

CriarArvore( )
{
}

Só cria nesse caso 1 ÁRVORE

PARA CRIAR VÁRIAS ÁRVORES



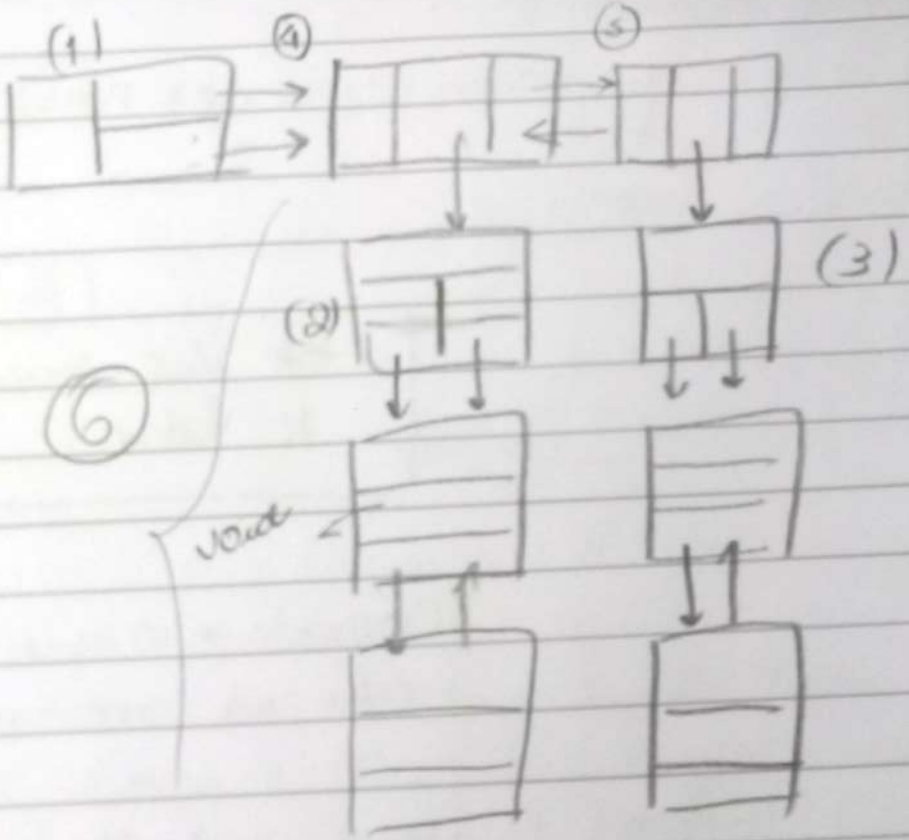
CriarArvore(tpcabec** Pcab)
{
}

** Referência
* Valor

Corrente → o que você está no momento

Como anda? irProx(Ptcabec* pcab)

CriaLista(&PtCols) (1)
CriaLista(&PtCol1) (2)
CriaLista(&PtCol2) (3)
CriaNO(PtCols, PtCol1) (4)
CriaNO(PtCols, PtCol2) (5)
Obter(PtCols, ponteiro)
CriaNO(ponteiro, Pvalor1)
CriaNO(ponteiro, Pvalor2)
} 6
AvançaProx(PtCols)



25103119

8) Propriedades da Modularização

- Encapsulamento - Proteção
- acoplamento - interface
- Coesão - Conceito

9) Encapsulamento

Propriedade relacionada com a proteção dos dados de um Componente de forma que este possa ser utilizado sem perder as suas características básicas.

ex: Televisão: Você recebe uma televisão, que tem vários circuitos, mas opera pelos botões.

Vantagens: • facilitar a manutenção, pois tudo que está relacionado com a estrutura ~~relacionada~~ encapsulada, está dentro do mesmo módulo.

• facilita a documentação do que se encontra encapsulado pois é um único Conceito.

desvantagem: exagero

25.03.19

Tipos de encapsulamento:

- de código: uma função de acesso contida no módulo de implementação e que não é vista pelo módulo Cliente.

```
. for ...
  {
  ========  } esse código está encapsulamento no for
  }
```

. while

. código de uma função está dentro

Obs: se a função usar ponteiro, pode ferrar o encapsulamento

- de variáveis: - static → encapsulada na classe ou módulo
  - local → bloco de código
  - Private → Objeto
  - Protected → estrutura de herança

- de documentação:
  . interna → ~~rotulada~~ documentação do (.c)
  . externa → documentação da interface (.h)
  . de uso → usuário

10) Acoplamento

Propriedade relacionada com a interface entre os módulos

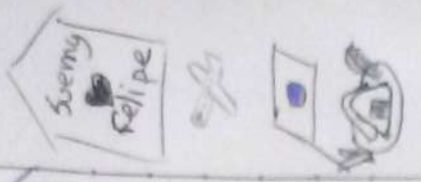
25.03.19

- Conector: é o item da interface
  ex: · Protótipo de função
  · arquivo
  · Variável global dos módulos

- Critérios de Qualidade de acoplamento
  I) Tamanho do Conector
  ex: função com 10 parâmetros
  X
  função com 1 parâmetro
  II) Quantidade de Conectores
  ↳ quantidade de items que são necessários e suficientes ex: .h com 40 funções

  III) Complexidade do Conector
Protocolo de uso (se a documentação é bem feita, o Conector difícil pode ser facilmente entendida)

11) Coesão
Propriedade relacionada com o grau de interdependência dos elementos que compõem o módulo.
(Conceito)
- Níveis de Coesão:
  I) Incidental: não há relação entre os vários conceitos inseridos no módulo
  II) Lógica. Os elementos possuem uma relação lógica entre os Conceitos de uma forma possivelmente genérica. ex: Módulo que calcula



III) Temporal: Os elementos estão relacionados pela necessidade de serem utilizados dentro do mesmo período de tempo

IV) Procedural: Os elementos estão relacionados pela necessidade de serem utilizados ~~dentro~~ em determinada ordem ex: .bat

V) Funcional: Os elementos estão relacionados por funcionalidade. Todos os elementos de uma mesma funcionalidade estão no mesmo módulo

VII) Abstração de Dados: Um único conceito.

03.04.19

Perdi a aula do dia 27

Especificação de Requisitos

1) Requisitos
   - O que deve ser feito
   - Nunca COMO deve ser feito

2) Características dos Requisitos
   - Curtos e diretos
   - linguagem Natural

3) etapas da especificação
   - elicitação
     - Técnicas
       - entrevista
       - brainstorm
       - questionário
   - documentação
     - requisitos genéricos
     - requisitos específicos
     - Contrato

- Verificação
  - Tem que ver se dar para fazer
  - análise se a documentação somente possui requisitos computáveis.
  - junto com a equipe técnica

SERPRO

- Validação de Requisitos
  - Cliente

4) Tipos de Requisito

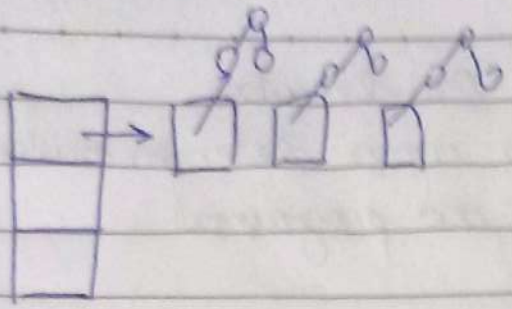
- Funcional
  Regras que devem ser implementadas na aplicação relacionadas com o negócio

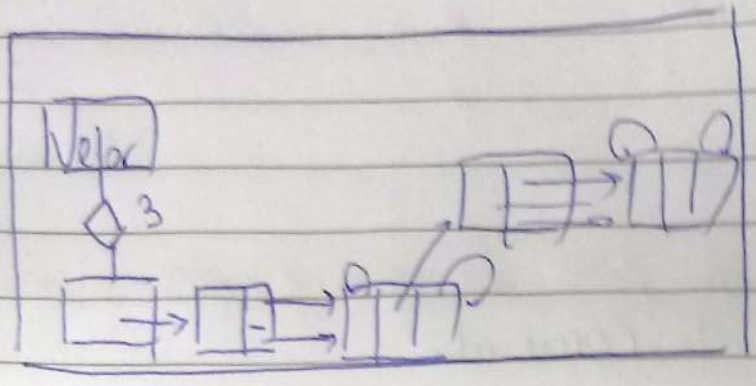
- Não funcionais
  Propriedades que a aplicação deve possuir e que não necessariamente estão relacionadas com o negócio
  ↳ login e Senha é um Requisito não funcional de Segurança
  ↳ disponibilidade: 24x7
  ↳ backup
  ↳ Velocidade: Todas as Consultas devem retornar resultado no máximo em 3seg.

- inverso
  - O que a equipe ~~não~~ se Compromete a não fazer

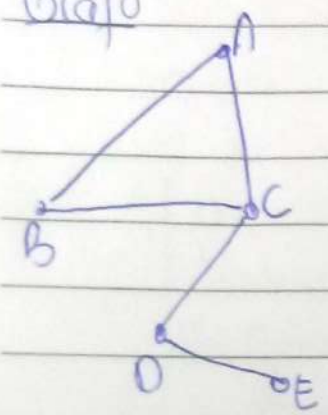
5) Exemplos de requisitos

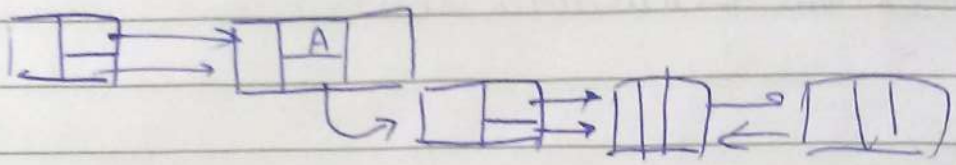
a) bem formulados
- Para cada aluno deve Ser Cadastrado ma-tricula e nome.

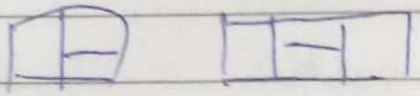


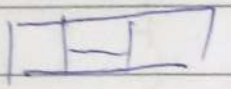


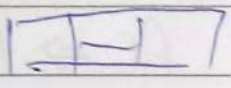
Grafo

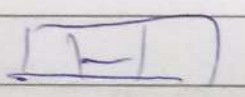


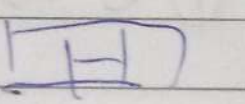


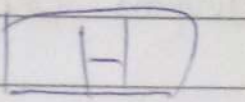






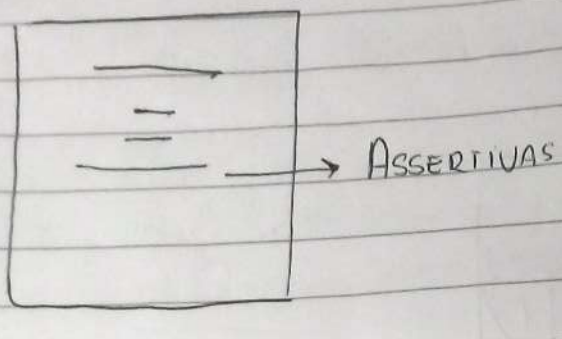






15.04.19

ASSERTIVAS ( ) Regras Consideradas válidas uma determinada parte do programa



São utilizadas em:
- Argumentação de Corretude
- instrumentação

ASSERTIVAS Estruturais são

ASSERTIVAS de entrada e saída

AE →

Bloco de Código

F V
F F
V V
V F

AE ⇒ AS

AS →

AE (+) B ⇒ AS

Obs: AS ASSERTIVAS devem Ser CORRETAS E Completas

AE → ✱

excluir nó corrente intermediário de uma lista duplamente encadeada com cabeça

As → ✱✱

✱ = { lista existe e possui pelo menos 3 nós
. Ponteiro corrente aponta para um nó corrente intermediário que se quer excluir
. Valem as assertivas estruturais da lista duplamente encadeada com cabeça

✱✱ := { nó foi excluído
. Valem as assertivas estruturais da lista duplamente encadeada com cabeça
. Ponteiro corrente aponta para o 1º Nó da lista.

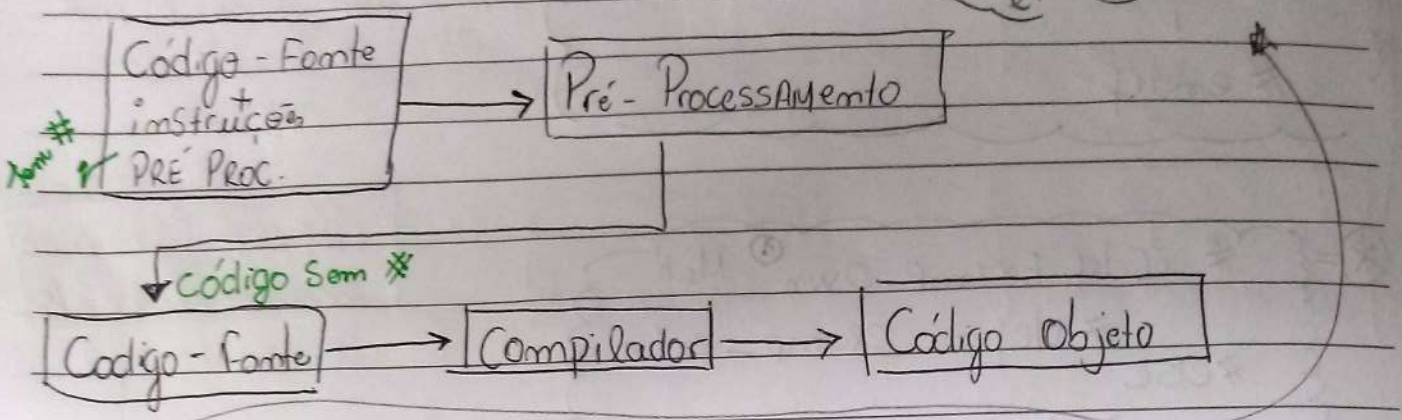
Ponto extra: Colocar ASSERTIVA de entrada e saída das funções de acesso (.h)

1 ponto extra

dia 29/04/19

6) Pré-Processamento

Por ex: se tiver o nome no comentário ele troca também. Se fizer #define nome ele troca por nada



Pré Processamento (não é comando de C)

define nome Valor → Substituidor de texto ~ Substitui nome por Valor

include < nome - arquivo > Copia o Texto do nome-arquivo

Procura na pasta padrão

nome - arquivo → Procura na mesma Pasta do .c

if defined (nome) ou # ifdef nome → Se tiver um #define nome ele entra no Texto V

Texto V

entraria aqui

else → se não estiver definido

Texto F

if ! defined

ou

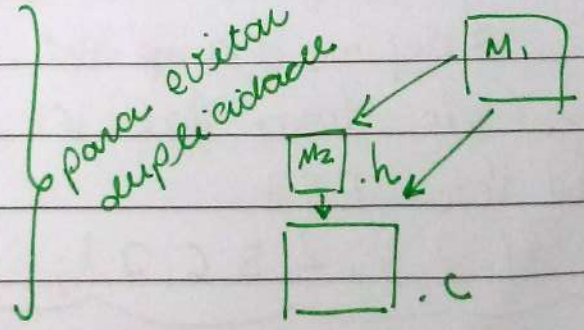
if ndef

endif

endef nome → Para de trocar

```
# if ! defined (Exemplo_Mod)
# define Exemp_Mod
      Corpo do .h
# endif
```

para evitar duplicidade



M0 M2 Tem um include

FORO

(2)

```
Exemp_ext  int Vetor [7]
   # if defined (exemp_own)
        = {1, 2, 3, 4, 5, 6, 7};
   # else
        ;
   # endif
```

(1) M1.h

```
# ifdef Exemp-Own
      # define exemp_exter
#else
      # define Exemp_exter extern
# endif
```

(3)

```
# define exemp_own
# include "M1.h"
# undef exemp_own
```

Como Seria Caso fosse ①, ②, ③ nessa ordem?

→ # define exemp-own ;

→ # include "M1.h"

{ # ifdef exemp-own (tá def. sim?

define exemp.ext

agora ele troca todos os exemp.ext por nada

→ int Vetor [7]

→ = {1, 2, 3, 4, 5, 6, 7};

aqui temos int Vetor [7] = {1, 2, 3, 4, 5, 6, 7};

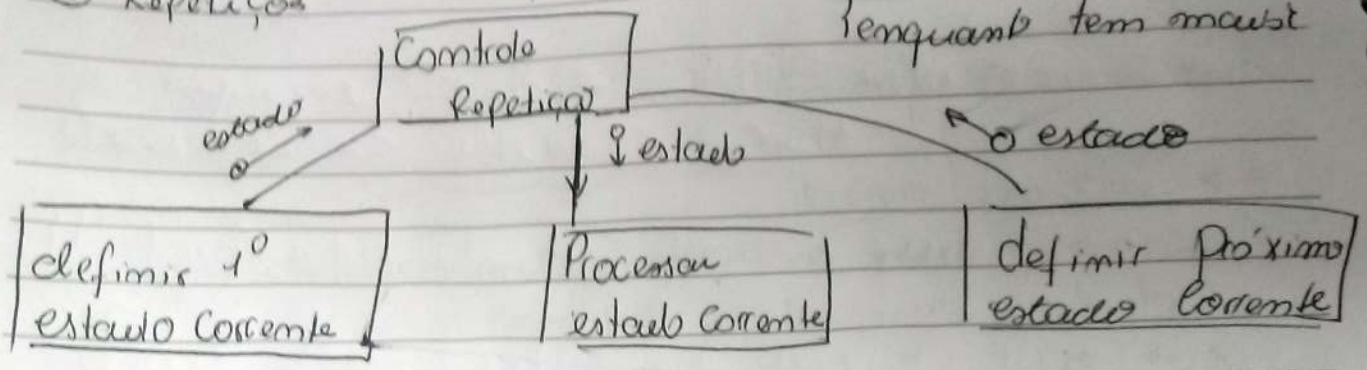
#include m.h ④

Como seria se fosse ①, ②, ④?

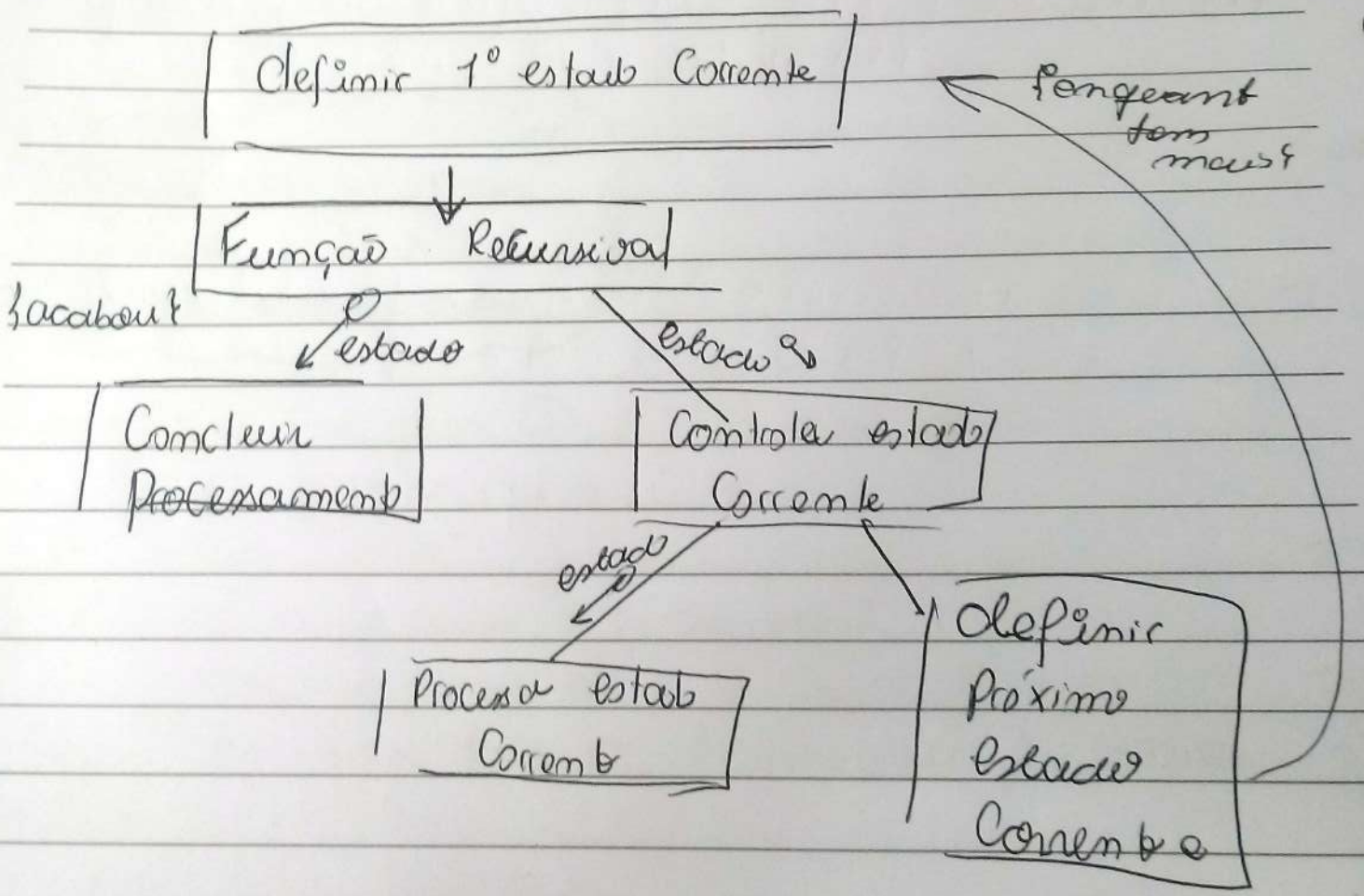
→ include m.h
{ → não temos a exemp_um definida, logo ele troca exemp-ext por extern
→ extern int Vetor [7] ;

08/09 (Passando a limpo)

⑦ Repetição



⑧ Recursão



(9) estado

· 1 Descritor de estado:

- Variável ou Variáveis que definem um estado

Ex:

busca sequencial → int i;
busca binária → int inf; int sup;

estado:

- Valoração do descritor de estado

(10) esquema de algoritmo

framework

```
{
  inf = ObterLimInf();      ——— Hot Spot
  sup = obterLimSup();
  while (inf <= sup) {
     meio = (inf + sup)/2;
     Comp = Comparar(Valorproc, obterValor(mei…
     if Comp == igual {break};
     if (Comp == Menor) {sup = meio-1}
      else {inf = meio + 1}
```

Obs: Esquema de algoritmo permitem encapsula… estruturas de dados utilizadas.

É correto, é incompleto e precisa ser in… ciado.

Normalmente Code em:
- POO
- Frameworks

se esquema Correto e hotsposts com assertivas validas => Processo Correto

(11) Parâmetros de tipo ponteiro para função

```
float areaQuad(float base, float altura)
    { return base * altura; }

float areaTriang(float base, float altura)
    { return base * altura /2; }

    :
    :
    :

CondRet = ProcessaArea(3,2, areaQuad);
CondRet = ProcessaArea(3,2, &areaTri);
```