Aluno: Nagib Moura Suaid 1710839

Introdução:

Vantagens da programação modular:

-Vencer Barreiras de complexidade
-Facilita trabalho em grupo(Paralelismo)
-Reúso de código
-Facilita a criação de um acervo(coleção)
-Desenvolvimento incremental
-Aprimoramento individual
-Facilita a administração de baselines(versões estáveis)

Princípios de Modularidade:

1- Módulo (caixa)

definição física: Unidade de compilação independente
definição lógica: Trata de um único conceito

2-Abstração de Sistema

Processo de considerar apenas o que é necessário em uma situação (definir o escopo)
(Reduzir entidades/objetos às suas características relevantes)

Níveis de abstração:

Sistema
|
V
Programas
|
V
Módulos
|
V
Funções
|
V
Blocos de código
|
V
Linhas de código

OBS: **Artefato**: é um item com identidade própria criado dentro de um processo de desenvolvimento. Pode ser versionado.

Construto (build): é um “resultado apresentável”.(necessariamente um artefato).

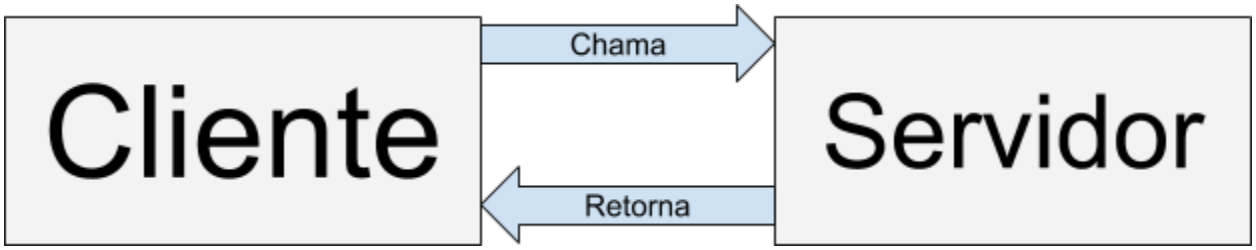
3-Interface

Mecanismo de troca de dados, estados e eventos entre elementos de um mesmo nível de abstração

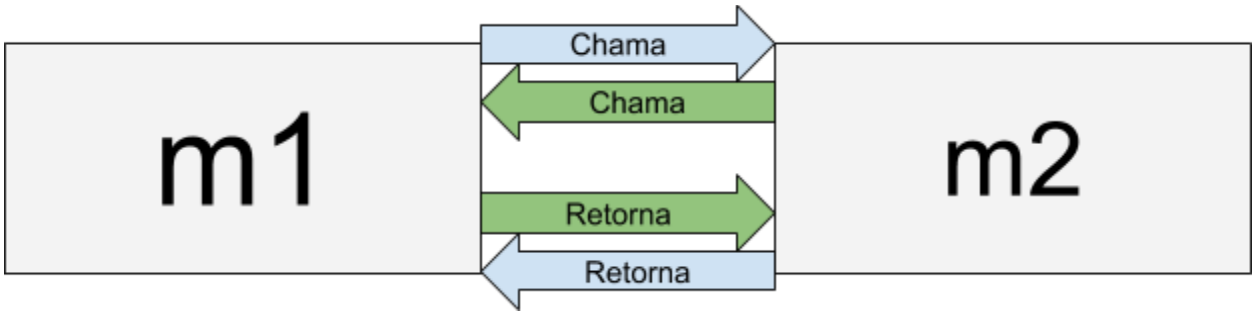
a)Exemplos:

-Entre Sistemas: Arquivos
-Entre Módulos: Funções de acesso
-Entre Funções: Parâmetros
-Entre Blocos: Variáveis globais

b)Relacionamento Cliente-Servidor

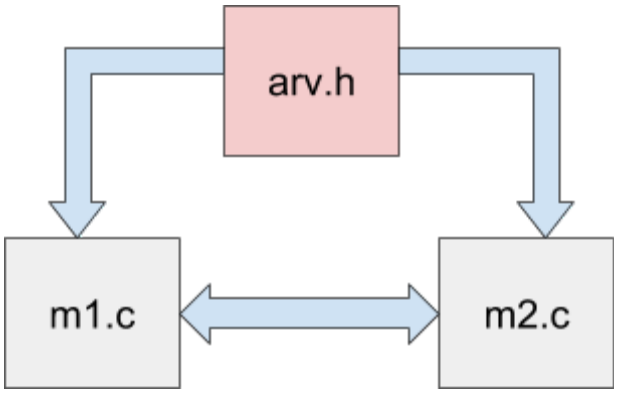


Callback: O servidor precisa de mais informação para realizar a sua tarefa, então a relação cliente-servidor inverte temporariamente:



c)Interface fornecida por terceiros

Módulos que utilizam a mesma estrutura precisam de uma interface fornecida por terceiros



d)Interface em detalhe

-Sintaxe(Regras)

MOD1 ←X→ MOD2
int float

-Semântica(Significado)

MOD1 ←X→ MOD2
int idade int cpf

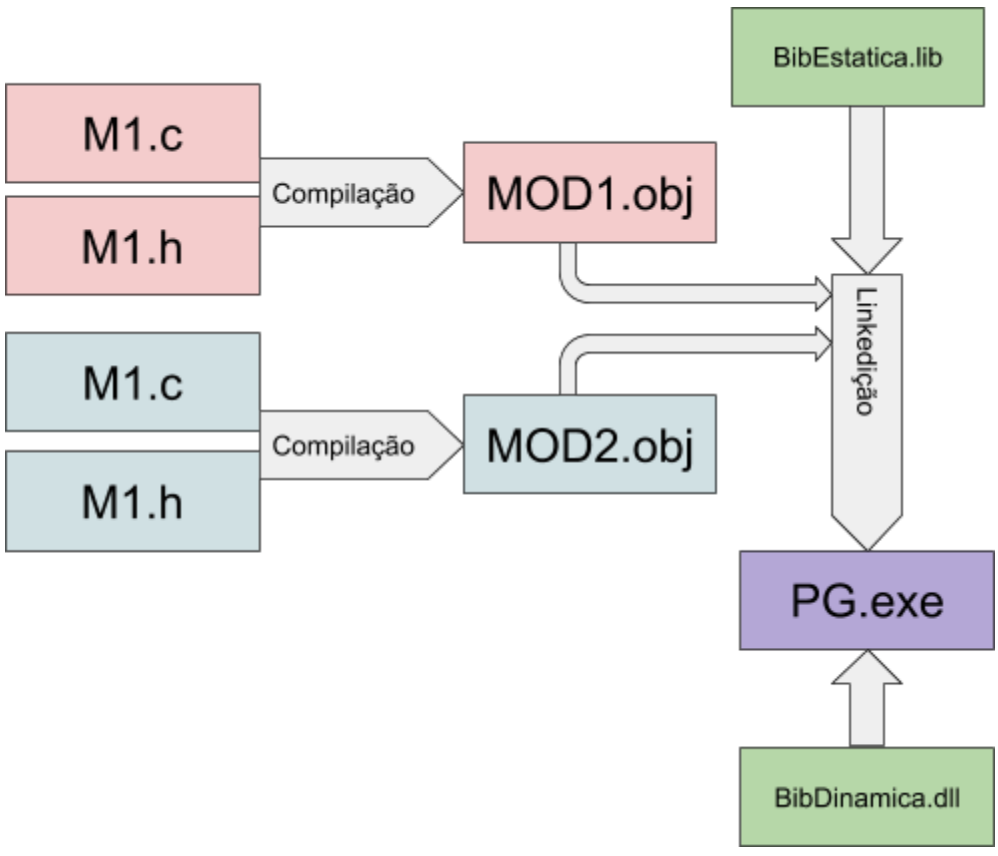
e)Análise de Interface

tpDadosAluno* ObterDadoAluno(int mat); (protótipo da função)

Cliente espera um ponteiro tpDadosAluno*
Servidor espera um inteiro int mat
Ambos esperam a estrutura tpDadosAluno

4-Processo de Desenvolvimento

OBS: .c→ Módulo de Implementação, .h→ Módulo de Definição
OBS2: TODO CLIENTE PRECISA DO .h DE SEU SERVIDOR



5-Bibliotecas Estáticas e Dinâmicas

Estática:

Vantagem:

.lib já é acoplado em tempo de linkedição à aplicação executável

Desvantagem:

Existe uma cópia da biblioteca estática na memória para cada executável que a utiliza

Dinâmica:

Vantagem:

Só é carregada uma instância em memória, independente do número de aplicações que a usam

Desvantagem:

.dll precisa estar na máquina para a aplicação funcionar (dependência externa)

6-Módulo de Definição

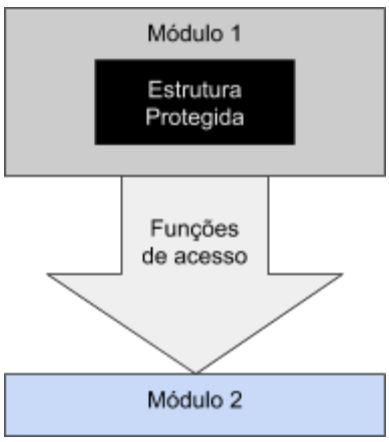
-interface do módulo
-contém os protótipos das funções de acesso, interfaces fornecidas por terceiros
-documentação voltada para o programador do módulo cliente

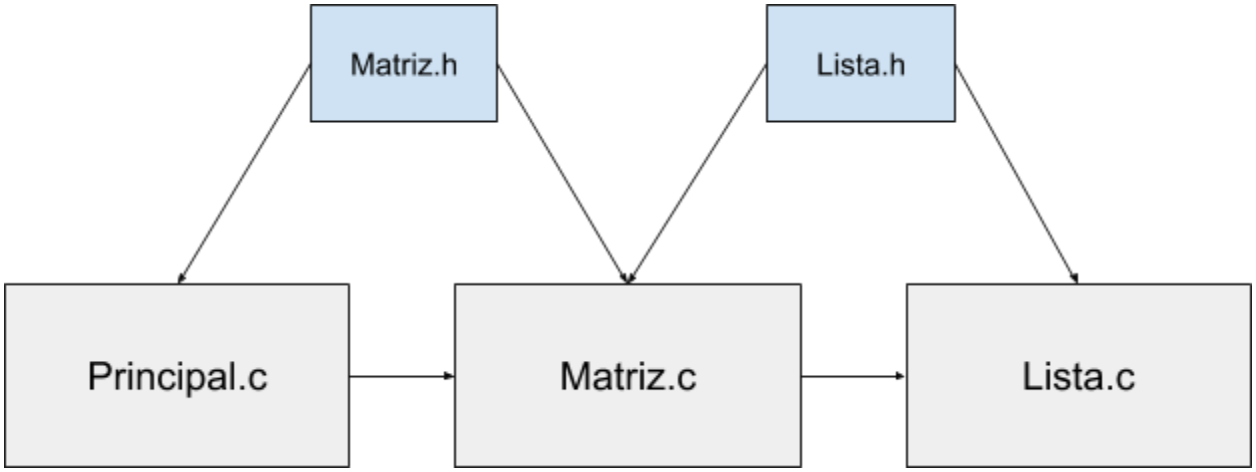
7-Módulo de Implementação

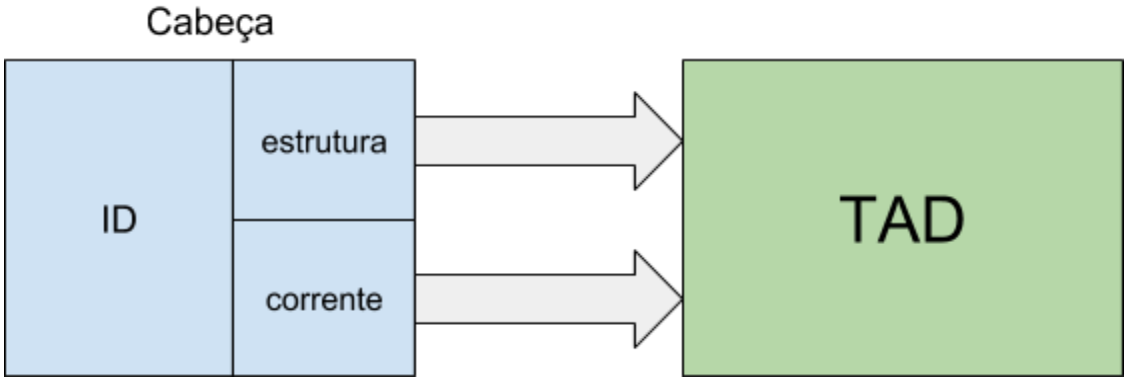
-código das funções de acesso
-protótipo e código das funções internas
-variáveis internas ao módulo
-documentação voltada para o programador do módulo servidor

8-Tipo Abstrato de Dados

É uma estrutura encapsulada em um módulo que somente é conhecida pelos módulos cliente através das funções de acesso disponibilizadas na interface.







Utilização e administração de um tad:
typedef struct ptcab* ptCab

criarLista(ptCab* plista) passagem de parâmetro por referência
inserirNo(ptCab, void *conteudo) passagem de parâmetro por valor

9-Encapsulamento

Propriedade relacionada com a proteção dos elementos que compõem um módulo
Facilita a manutenção e impede a utilização indevida

*encapsulamento de documentação:
- -Documentação interna → módulo de implementação
- -Documentação externa → módulo de definição
- -Documentação de uso → manual do usuário

*encapsulamento de código:
    -dentro de módulos
    -dentro de blocos de código
    -dentro de funções

*encapsulamento de variáveis :
    -private→ objeto
    -public→ mundo (orientado a objeto)
    -global→ mundo
    -static→ módulo/classe
    -protected→ estrutura de herança

10-Acoplamento

Propriedade relacionada com a interface entre módulos

conector==item da interface (ex.: função de acesso, variável global...)

Critérios de qualidade:
    -Quantidade de conectores→ necessidade X suficiência (preciso?, basta?)

    -Tamanho do conector→ quantidade de parâmetros

    -Complexidade de conector→ documentação e mnemônicos (nomes autoexplicativos)

11-Coesão

Propriedade relacionada com o grau de interligação dos elementos que compõem o módulo

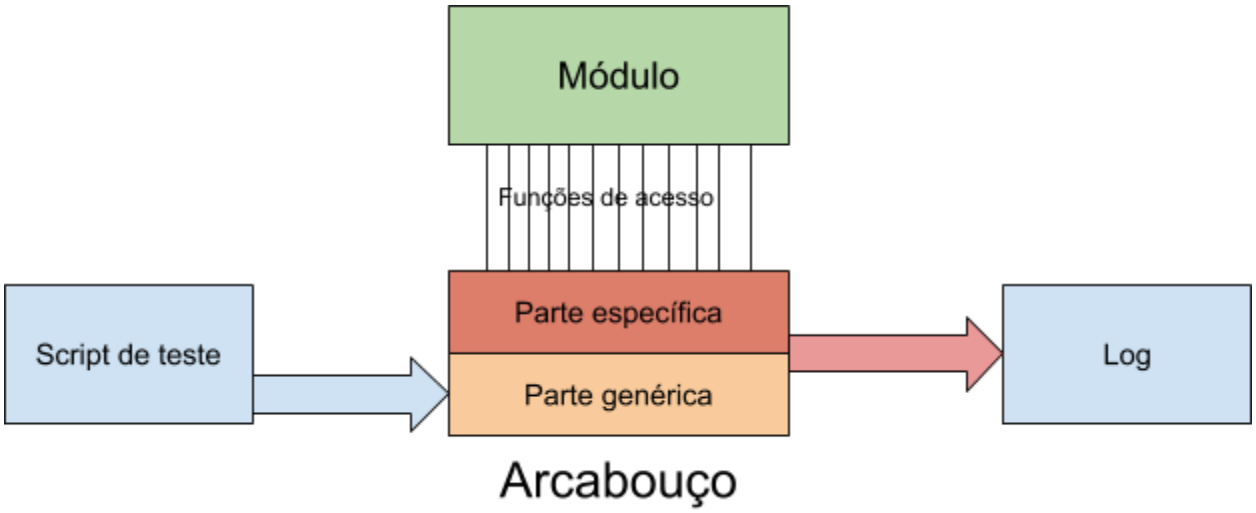
níveis de coesão:
    -incidental→ nada a ver
    -lógica→ relação lógica (ex.: processa)
    -temporal→ tarefas feitas no mesmo período de tempo
    -procedural→ tarefas feitas em sequência
    -funcional→ tarefas semelhantes
    -abstração de dados→ um conceito (o melhor)

Teste Automatizado

1) Objetivo:

Testar de forma automática um módulo, recebendo um conjunto de casos de teste (script) e gerando um log de saída com a análise entre o resultado obtido e o esperado.

2) Framework de teste



3) Script

== caso de teste → testa uma situação(contexto)(pode ter vários comandos)
=comando de teste→ chama uma única função de acesso
=recuperar→ ignora um erro esperado

Um script de teste completo deve testar todas as possíveis condições de retorno de cada uma das funções de acesso.

4)Log de saída

Arquivo de texto listando os casos de teste do script, onde houve algo inesperado e onde tudo ocorreu como previsto:

==caso x → OK
1>> Func esperava 0 e retornou 1 → ERRO

*Toda informação do Log após o primeiro erro não pode ser confiada.

5) Parte específica

Necessita ser implementada para que o framework possa acoplar no app: Hotspot

Ex: TESTEARV.c:

/* Tabela dos nomes dos comandos de teste específicos */

```
#define      CRIAR_ARV_CMD       "=criar"
#define      INS_DIR_CMD         "=insdir"
#define      INS_ESQ_CMD         "=insesq"
#define      IR_PAI_CMD          "=irpai"
#define      IR_ESQ_CMD          "=iresq"
#define      IR_DIR_CMD          "=irdir"
#define      OBTER_VAL_CMD       "=obter"
#define      DESTROI_CMD         "=destruir"
...
 TST_tpCondRet TST_EfetuarComando( char * ComandoTeste )
   {

      ARV_tpCondRet CondRetObtido   = ARV_CondRetOK ;
      ARV_tpCondRet CondRetEsperada = ARV_CondRetFaltouMemoria ;
                                 /* inicializa para qualquer coisa */
      char ValorEsperado = '?'  ;
      char ValorObtido   = '!'  ;
      char ValorDado      = '\0' ;
      int  NumLidos = -1 ;
      TST_tpCondRet Ret ;
      /* Testar ARV Criar árvore */
      if ( strcmp( ComandoTeste , CRIAR_ARV_CMD ) == 0 )
      {

            NumLidos = LER_LerParametros( "i" ,
                                 &CondRetEsperada ) ;
            if ( NumLidos != 1 )
            {
            return TST_CondRetParm ;
            } /* if */

            CondRetObtido = ARV_CriarArvore( ) ;

            return TST_CompararInt( CondRetEsperada , CondRetObtido ,
                                 "Retorno errado ao criar árvore." );

      } /* fim ativa: Testar ARV Criar árvore */
```

Processo de Desenvolvimento em Engenharia de Software

Demanda→ Analista de negócios→ Líder de Projeto

A demanda vem de um cliente.
O Líder deve estimar:

- Tamanho do projeto (ponto de função)
- Esforço para terminar o projeto
- Recursos necessários
- Prazo de término do projeto

Etapas:

- Requisitos (o que o Cliente quer):
  - -Elicitação
  - -Documentação
  - -Verificação
  - -Validação
- Análise e Projeto
  - -Projeto Lógico (Modelagem de dados, ...)
  - -Projeto Físico (Módulos, arquivos, BD, ...)
- Implementação
  - -Programas
  - -Teste Unitário
- Teste Iterado
  - -Teste integrado
- Homologação (Apresentação ao Cliente)
  - -Sugestão-> Retrabalho renumerado
  - -Erro-> Se ferrou, se vira
- Implantação

OBS:

- *Gerência de configuração
- *Gestor de qualidade de software