

Bacharelado em Ciência e Tecnologia

BC -1501 Programação orientada a objetos

UFABC

Aula 01 – Introdução

Material Adaptado do
prof. André Balan
(CMCC)

Conteúdo

- Introdução à POO
- Introdução à linguagem Java

Introdução

- ▶ O termo Programação Orientada a Objetos (POO) foi criado por Alan Kay, no início da década de 70
- ▶ Também é o autor da linguagem de programação Smalltalk

Conceitos

- A POO é um **paradigma** de programação, que foi criado com o intuito de **modelar** o “mundo real” dentro do computador.
- Quando dizemos mundo real, queremos dizer uma porção limitada, ou alguma “coisa” que de fato existe, envolvendo seres vivos e objetos.
- Paradigma, no dicionário Aurélio, tem o significado de “**padrão**”, ou “**metodologia**”

Modelos

- Na POO, o programador é responsável pela criação de modelos computacionais que representem elementos do mundo real: pessoas, animais, plantas, veículos, edifícios, documentos, máquinas, estruturas de dados, etc...
- Estes modelos computacionais são representações simplificadas dos **elementos** reais, que buscam descrever principalmente:
 - Propriedades mais relevantes
 - Comportamentos mais relevantes

Modelos

➤ Exemplo:

➤ Um programador está interessado em criar uma aplicação para gerenciar pedidos em um restaurante

➤ Um modelo para “Mesa de restaurante”

➤ Propriedades mais relevantes

- Número da mesa
- Número de lugares
- Localização
- Estado (ocupada?)
- Pedidos entregues
- Pedidos Pendentes

➤ Comportamento (Ações)

- Adicionar pedido
- Fechar a conta

Modelos

- Outros modelos importantes na **modelagem** deste sistema de restaurante:
- Pedido
- Produto
- Funcionário
- Restaurante

LPOOs

➤As linguagens que seguem o paradigma de orientação a objetos são denominadas **Linguagens orientadas a objetos**

➤Exemplos

➤Java

➤Smalltalk

➤Object Pascal

➤C++, C#

➤Phyton

Classes

- ▶ Nestas linguagens, a criação de modelos é feita por meio das **Classes**
- ▶ “Classes são estruturas sintáticas das LOOs utilizadas para criação de modelos”
- ▶ Existem regras gramaticais para escrever as várias estruturas sintáticas
 - classes, loops, condições, atribuições, declarações, etc...

Classes

- .Classe: pode ser definida como a união de objetos com características semelhantes a partir de um domínio.
- .Cada classe possui métodos e dados que determinam as características comuns de um conjunto de objetos.



Classe dos
Animais

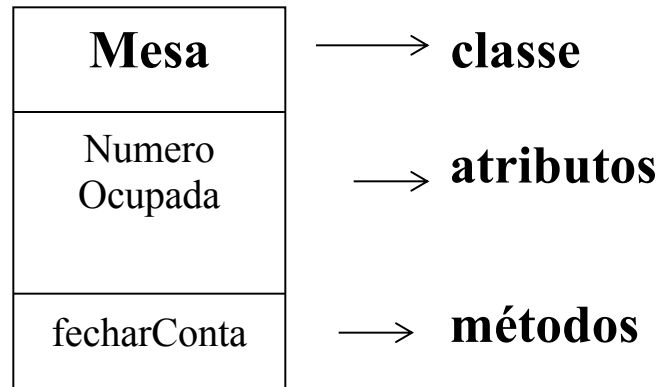
Classes

➤ Exemplo de classe em Java e C++

```
■ class MesaRestaurante {  
    char numero;  
    Pedido [] pendentes;  
    Pedido [] entregues;  
    bool ocupada;  
  
    void adicionaPedido(Pedido p);  
    void fechaConta();  
}
```

Atributos

Métodos



Objetos

O que são **objetos** no paradigma de programação orientada a objetos?

- “Objetos são instâncias das classes”
- ocupam uma determinada porção de memória, necessária para armazenar os valores dos atributos
- representa um elemento ou entidade do mundo real dentro de um determinado domínio com regras definidas.

Objetos

- Para se criar um objeto, o operador comum nas LPOOs é o operador **new**
- Na linguagem Java, o acesso a um objeto só pode ser feito por meio das variáveis de referência

Objetos

➤ Exemplo: Objetos na linguagem Java

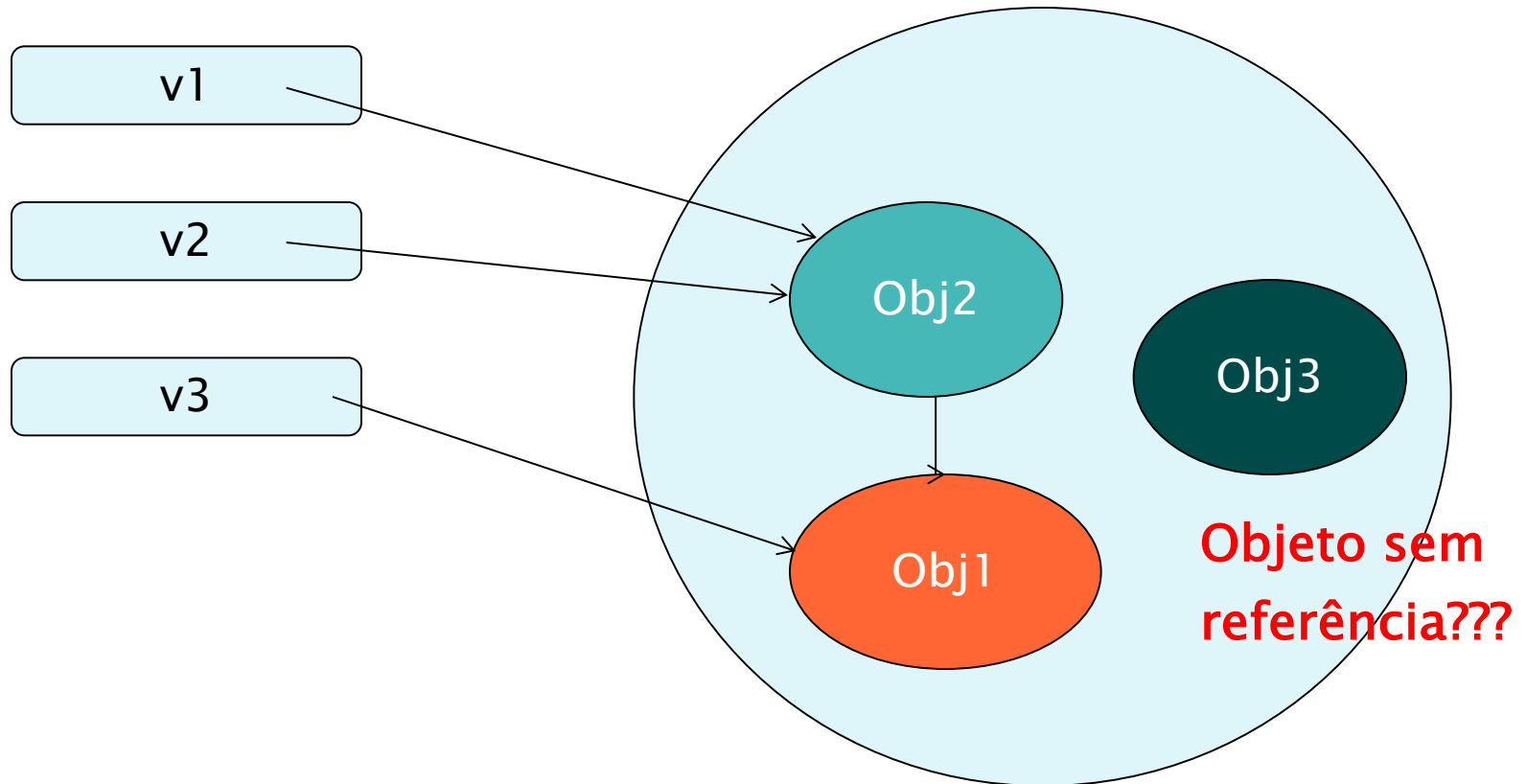
```
// declaração de uma variável de referência m1  
MesaRestaurante m1;
```

```
// Criação de um objeto! Aloca memória!  
m1 = new MesaRestaurante();
```

```
m1.numero = 10;    // acessando campos do objeto  
m1.fechaConta();   // chamando os métodos do objeto
```

Objetos

➤ Ilustração de objetos e variáveis de referência



Objetos

➤No exemplo anterior, o objeto que chamamos de “Obj3” está sem referência. Como isso acontece?

```
// Criação de uma variável de referência  
seguida da criação de um objeto
```

```
Mesa v1 = new Mesa();
```

```
v1 = new Mesa(); // v1 passa a referenciar  
um novo objeto.
```

O objeto anterior ficou sem referência. Uma porção de memória sem referência é considerada Lixo (garbage)

Objetos

➤ No exemplo anterior, o objeto que chamamos de “Obj2” é referenciado ao mesmo tempo por duas variáveis. Como isso pode ser feito?

```
// Criação de uma variável de referência  
seguida da criação de um objeto
```

```
Mesa v1 = new Mesa();
```

```
v2 = v1    // Agora v2 e v1 referenciam  
           o mesmo objeto recém criado.  
           Não há dois objetos! Somente um!
```

Máquina virtual Java

- Os programas da linguagem JAVA são executados por uma máquina virtual: *um sistema operacional desenvolvido pela SUN que roda em cima de outro sistema operacional (Windows, Linux, Mac....)*
- A máquina virtual JAVA possui um mecanismo de coleta/remoção de lixo: *Garbage Collector*, para desalocar porções de memória (objetos) que ficaram sem referência

Garbage Collector

- Programas desenvolvidos em linguagem C/C++, por exemplo, são executados diretamente pelo sistema operacional real (Windows, Linux...)
- Como os SOs atuais não possuem este mecanismo de coleta de lixo, os programadores de C/C++ **devem evitar a todo custo** que qualquer porção de memória atribuída ao seu programa fiquem sem nenhuma referência.

Ponteiros

- As variáveis funcionam como ponteiros, porém, o conceito de ponteiros não é explícito em JAVA, assim como é na linguagem C/C++
- Em JAVA, o programador não tem o poder de trabalhar explicitamente com endereços de memória (aritmética de ponteiros):
- Menor risco do programador cometer erros
- Menor a eficiência dos programas

Stack vs Heap

Stack vs Heap

- Variáveis comuns, variáveis de referência, ponteiros e objetos podem ocupar duas **regiões de memória**:
- **Stack**: Porção relativamente pequena de memória que pertence exclusivamente ao programa em execução.

Espaço pequeno, mas alocação muito rápida

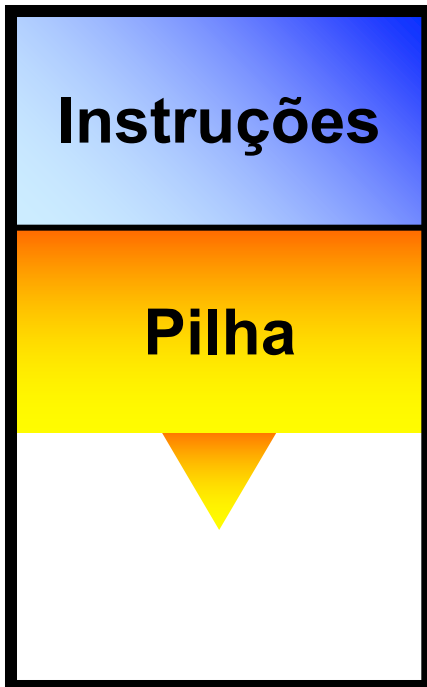
- **Heap**: Grande porção de memória gerenciada pelo SO que pode atender a qualquer processo. Alocação mais lenta

Processo

- Um programa, ao ser executado, torna-se um **processo** e ocupa uma porção contígua da memória interna do computador. Essa parte é dividida em 2 áreas:
- **Instruções** – armazena o código do programa em linguagem de máquina
- ***Stack* (Pilha)** – nela são armazenadas variáveis ao longo da execução do programa

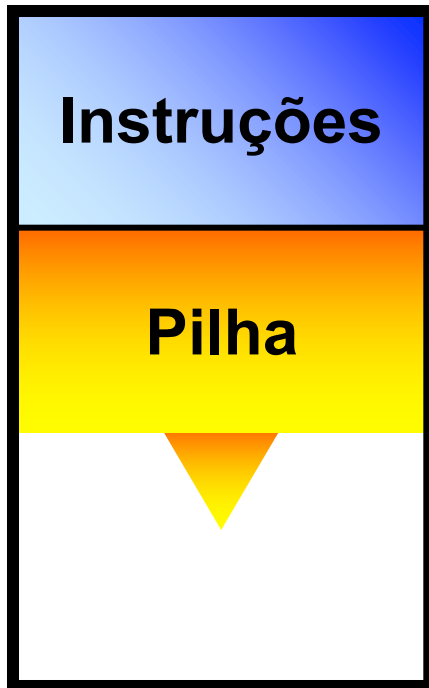
Processo

➤ Ilustração da porção de memória ocupada por um processo



Processo e a Heap

➤ Ilustração da porção de memória ocupada por um processo e a Heap



Porção de memória gerenciada pelo SO para ser distribuída entre os processos por meio de solicitações

HEAP

Programa em execução

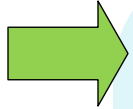
- ▶ Porque se chama pilha?

```
int a;  
  
void main() {  
    a = 4;  
  
    int b = 6;  
  
    int c = soma(a,b) ;  
  
}
```

Instruções

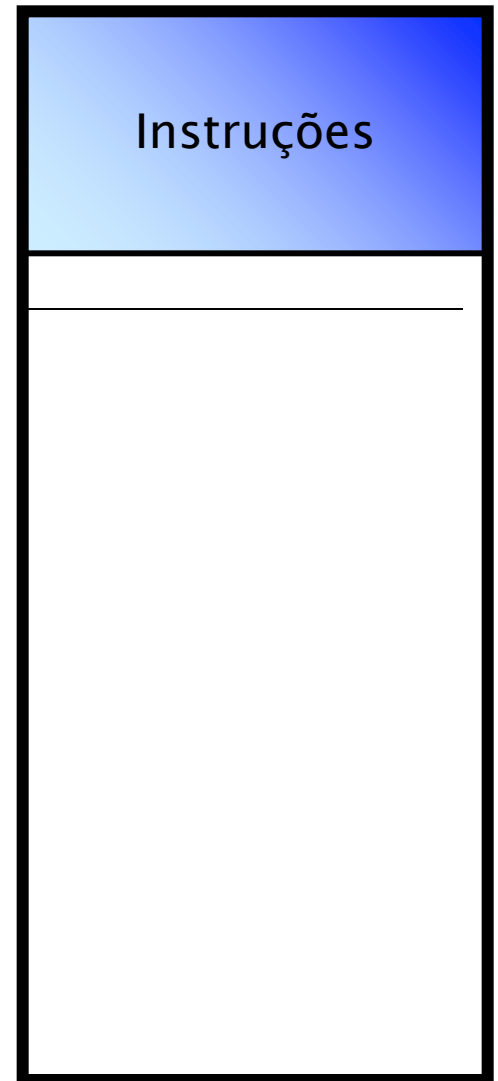
Programa em execução

► Porque se chama pilha?



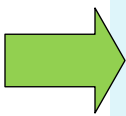
```
int a;  
  
void main() {  
    a = 4;  
  
    int b = 6;  
  
    int c = soma(a,b) ;  
  
}
```

a

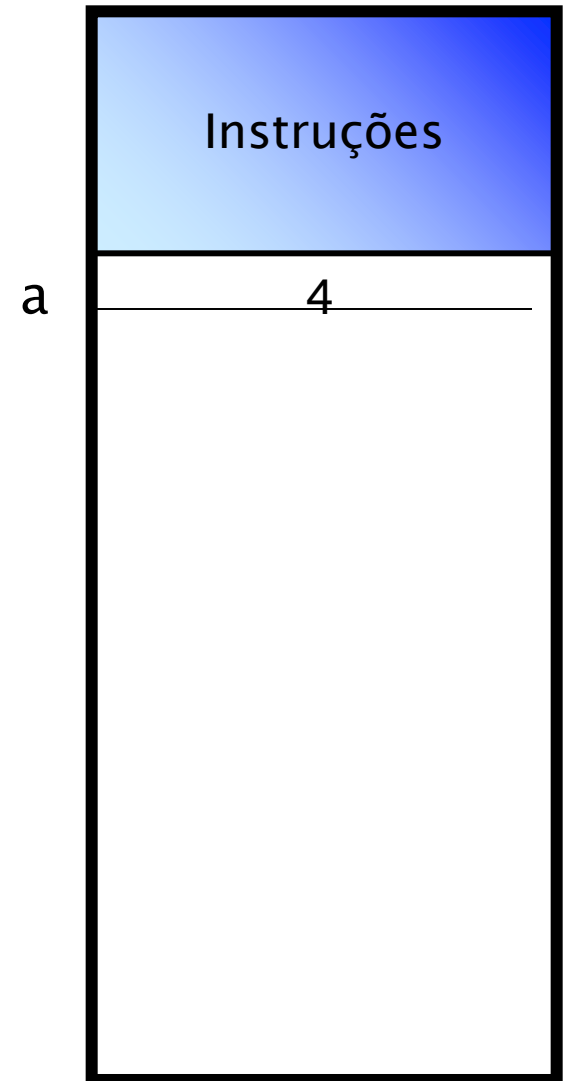


Programa em execução

► Porque se chama pilha?

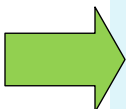


```
int a;  
  
void main() {  
    a = 4;  
  
    int b = 6;  
  
    int c = soma(a,b) ;  
  
}
```



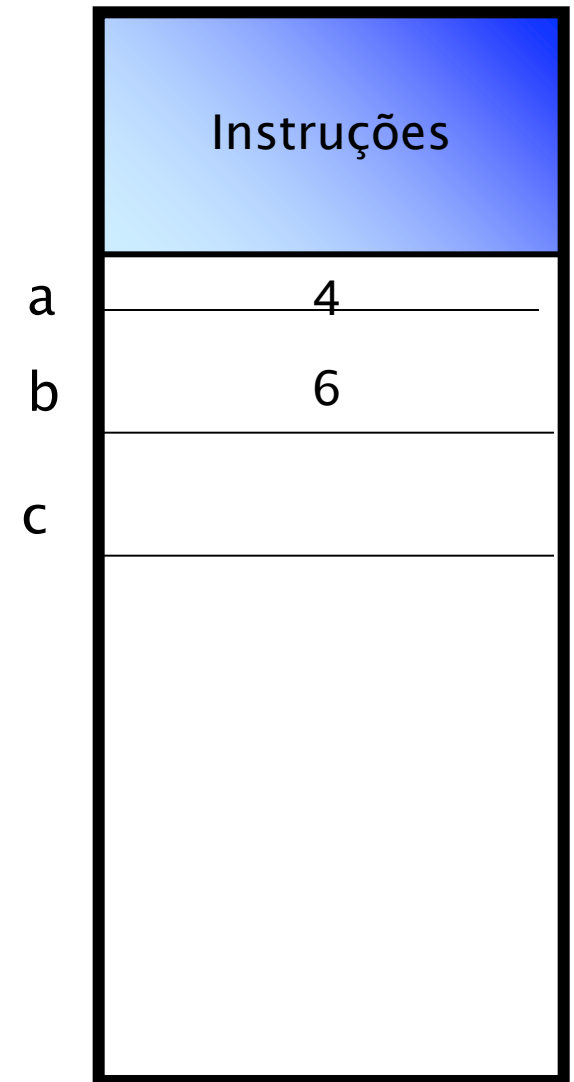
Programa em execução

► Porque se chama pilha?



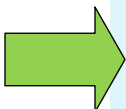
```
int a;  
  
void main() {  
    a = 4;  
  
    int b = 6;  
  
    int c = soma(a,b) ;  
  
}
```

Aloca espaço para c



Programa em execução

► Porque se chama pilha?



```
int a;  
  
void main() {  
    a = 4;  
  
    int b = 6;  
  
    int c = soma(a,b) ;  
  
}
```

retorno

a

4

b

6

c

a'

4

b'

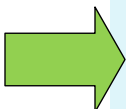
6

Instruções

Uma função soma é executada, ou seja, são criadas variáveis auxiliares, a e b “c

Programa em execução

► Porque se chama pilha?



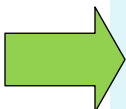
```
int a;  
  
void main() {  
    a = 4;  
  
    int b = 6;  
  
    int c = soma(a,b) ;  
  
}
```

O resultado $a+b$ é atribuído à retorno

Instruções	
a	4
b	6
c	
retorno	10
a'	4
b'	6

Programa em execução

► Porque se chama pilha?



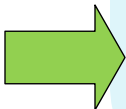
```
int a;  
  
void main() {  
    a = 4;  
  
    int b = 6;  
  
    int c = soma(a,b) ;  
  
}
```

retorno

Instruções	
a	4
b	6
c	
	10

Programa em execução

► Porque se chama pilha?



```
int a;  
  
void main() {  
    a = 4;  
  
    int b = 6;  
  
    int c = soma(a,b) ;  
}
```

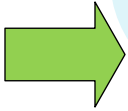
Instruções	
a	4
b	6
c	10

Neste momento, tudo que foi alocado para executar a função soma deixa de existir.

Programa em execução

► Porque se chama pilha?

```
int a;  
  
void main() {  
    a = 4;  
  
    int b = 6;  
  
    int c = soma(a,b) ;  
  
}
```



Instruções

Preste atenção!

► Em Java

`MyClass v1;` apenas cria uma variável de referência na stack

◦ Em C++

`MyClass v1;` cria um objeto que fica na stack!

Preste atenção !!!@\$#

- ▶ Em Java, qualquer **objeto**, **vetor** ou **matriz** são sempre alocados na **Heap**. Qualquer outro tipo de variável solitária (que não fazem parte de um objeto ou vetor ou matriz) (incluindo as variáveis de referência), permanecem na stack, e são desalocadas automaticamente
- ▶ Em Java, objetos não podem ser criados na Stack!
- ▶ Em C++ por outro lado, objetos podem ser alocados na Stack. (isto pode proporcionar maior eficiência em programas)

Encapsulamento, Herança e Polimorfismo

A Programação orientada a objetos possui 3 pilares

- Encapsulamento
- Herança
- Polimorfismo

Encapsulamento

- ❑ é o agrupamento dos dados e funções relacionados a uma classe, onde o acesso aos dados somente é permitido por meio dos próprios métodos.
- ❑ Está diretamente ligado a modularização do programa.
- ❑ Por questões de segurança, a classe passa a ser criada como uma caixa preta.
- ❑ Além de organizar a criação das classes o encapsulamento permite ocultar os dados e os métodos da classe quando necessário.

Encapsulamento

➤ Os programadores que utilizam POO podem exercer dois tipos de papéis:

➤ **Programador de classes** – projetam e implementam novas classes

➤ **Programadores usuários** – utilizam as classes já criadas para desenvolver aplicações/classes mais complexas

Encapsulamento

➤ *Quase sempre, o programador atua simultaneamente nos dois papéis, utilizando classes mais simples, para criar classes mais complexas*

Encapsulamento

Do lado do programador usuário de classes:

- Encapsulamento, é a capacidade dos objetos em ocultar detalhes que não são relevantes para o **programador usuário** (aquele que só está interessado em utilizar a classe)
- Este conceito está presente no mundo real:
 - Exemplo máquina fotográfica, carro, televisão
 - Os usuários finais desses produtos, têm acesso a apenas alguns comandos e informações, que constituem sua “**interface**”

Encapsulamento

➤ O encapsulamento é um processo natural da POO

➤ Exemplo:

- vamos supor que você é um programador que vai apenas utilizar as classes JOptionPane e THashList no seu novo programa. Não foi você quem desenvolveu essas classes.

➤ Naturalmente, **você não perderia muito tempo em estudar o mecanismo dessas classes, apenas se dedicaria a saber o que ela oferece para você**, afinal, você já sabe que essa classe funciona bem.

Encapsulamento

➤ Vantagens:

- O usuário tem acesso somente àquilo que lhe interessa e àquilo que ele pode mexer sem causar problemas;
- Ele não precisa ter conhecimento do funcionamento do objeto
- Ele apenas assume que o objeto “funciona” (abstração), e faz uso do mesmo

Encapsulamento

Do lado do programador desenvolvedor de classes:

ele conta com mecanismos da linguagem OO para esconder aquilo que, de fato o usuário não precisa ou não deve mexer, definindo tipos de **visibilidade** em Java:

Public : o programador pode usar/mexer a vontade

Private : o programador não vai nem saber que isso existe

Protected : o programador não vai nem saber que isso existe, a menos que ele esteja desenvolvendo uma **classe filha** para essa classe, ou uma classe do mesmo pacote (JAVA)

Encapsulamento

➤ Exemplo

```
■class Televisao {  
    private void setTensaoBase_T1(int c);  
    private void setResistencia_R5(int c);  
    public void mudaCanal(int c);  
    public void liga();  
    public void desliga();  
}
```

Neste classe, os **métodos privados só podem ser utilizados por métodos da própria classe.**

Por exemplo, o método `liga` é responsável por definir a tensão adequada na base do transistor T1 dependendo das configurações atuais de energia do aparelho. O usuário da classe, no entanto, não deve se atrever a mexer nessa tensão sem conhecimento.

Encapsulamento em Estruturas de Dados

- As **estruturas de dados** (ED) são mecanismos fundamentais na Ciência da Computação.
- Uma **estrutura de dados** consiste em um *container* de informações que deve fornecer, fundamentalmente, as seguintes operações sobre dados:
 - **Inserir**
 - **Remover**
 - **Consultar**

Encapsulamento em Estruturas de Dados

- A maioria das EDs precisam manter a todo momento a **organização e consistência de seus dados**, como um todo. Exemplo:
- Em uma **ED fila**, utilizamos tipicamente um vetor para armazenar elementos: **clientes, produtos, pedidos, mensagens, processos, etc.**
- A ordem dos elementos dentro do vetor determina a ordem de chegada dos elementos. “**O primeiro que entra é sempre o primeiro que sai**”
- Como desenvolvedor da classe Fila, você deve esconder este vetor do programador usuário, pelos seguintes motivos:
- O usuário poderia **alterar a ordem** dos elementos (não é permitido)
- O usuário poderia **inserir ou remover elementos no meio da fila** (não é permitido)

Encapsulamento em Estruturas de Dados

- A classe Fila deveria fornecer como interface para o usuário, somente as seguintes operações:
 - Inserer : insere um novo elemento no final da fila
 - Remove: remove o primeiro elemento da fila
 - Tamanho: retorna o número de elementos que estão atualmente na fila.
- Outras informações importantes para a ED fila que **não devem ser disponíveis** aos programadores usuários:
 - Índice do vetor que indica o começo da fila
 - Índice do vetor que indica o fim da fila
 - Contador do número de elementos inseridos na fila (o usuário não pode alterar)
 - Qualquer uma dessas informações, se forem expostas para o programador usuário, podem comprometer a consistência da ED fila. Como?

Por que encapsular?

- Neste ponto você pode estar com o seguinte pensamento:
- As classes que eu irei desenvolver só serão utilizadas por mim mesmo, nos meus próprios programas!
- Se você estiver com este pensamento, pense que:
- Quase sempre você trabalhará em equipe
- Para que seu projeto tenha uma vida longa, outras pessoas deverão dar continuidade, suporte e manutenção ao seu trabalho
- Com o tempo, você tende a esquecer o mecanismos das classes que você mesmo desenvolveu
- Sendo assim, desde já, devemos aprender a programar em POO com responsabilidade, utilizando adequadamente os mecanismos disponíveis por este paradigma, assim, faça o encapsulamento corretamente!