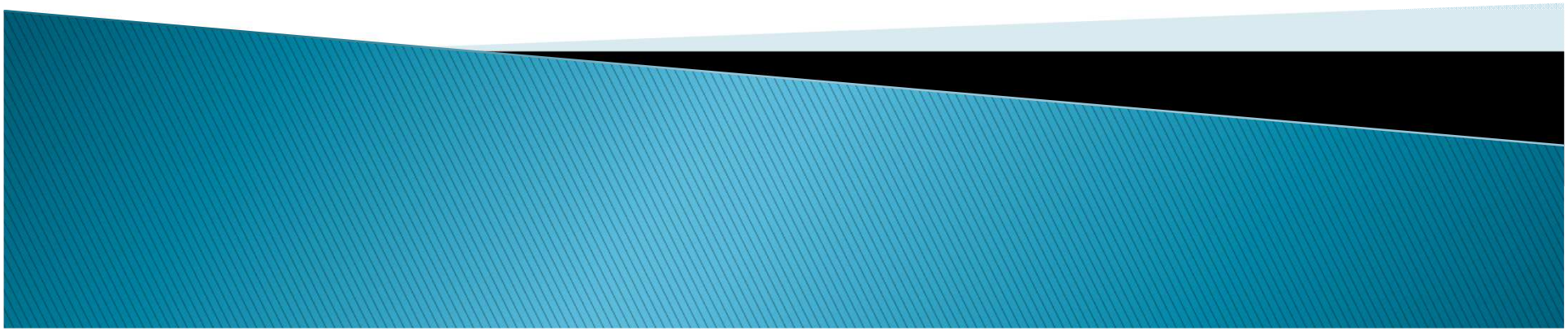
Estrutura de Dados

Listas Encadeadas Dinâmicas

Prof. Jesus José de Oliveira Neto



Listas Encadeadas
:: Representação por referências

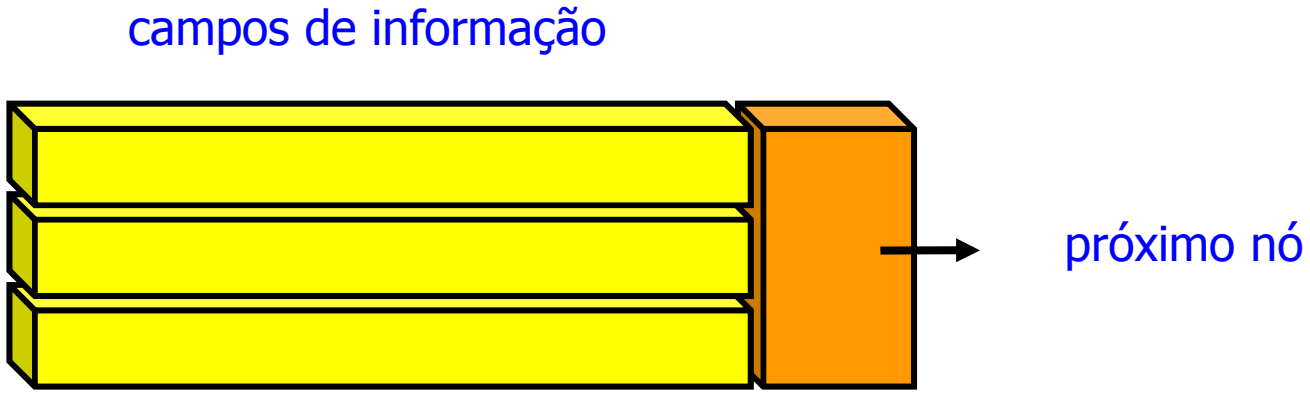
- Segunda forma: referências
  - Estruturas de dados dinâmicas: estruturas de dados que contém referências para si próprias.

```
class NoLista {
   String nomeTarefa;
   double duracao;
   String responsavel;
   ...
   NoLista prox;
};
```

Referência para a própria classe Lista

Listas Encadeadas
:: Representação por referências

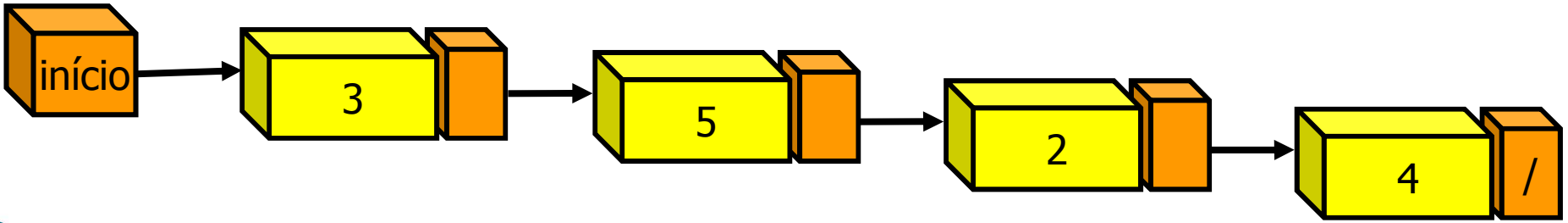
- Representação gráfica de um elemento da lista:



  - Cada item é encadeado com o seguinte, mediante uma variável do tipo referência.
  - Permite utilizar posições não contíguas de memória.
  - É possível inserir e retirar elementos sem necessidade de deslocar os itens seguintes da lista.

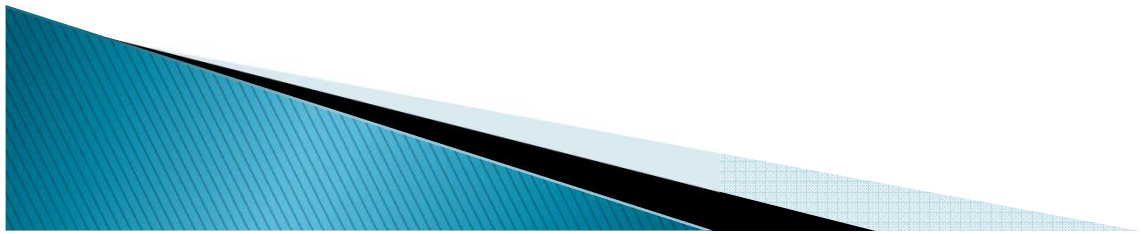
Listas Encadeadas
:: Representação por referências

- Cada item em particular de uma lista pode ser chamado de elemento, nó, célula, ou item.
- O apontador para o início da lista também é tratado como se fosse uma célula (cabeça), para simplificar as operações sobre a lista.
- O símbolo / representa a referência nula (null), indicando o fim da lista.



Criar Lista

```
public class NoLista {

   double  info;
   NoLista proximo;

   public NoLista(double valor)
   {
      this.info = valor;
      this.proximo = null;
   }

}
```



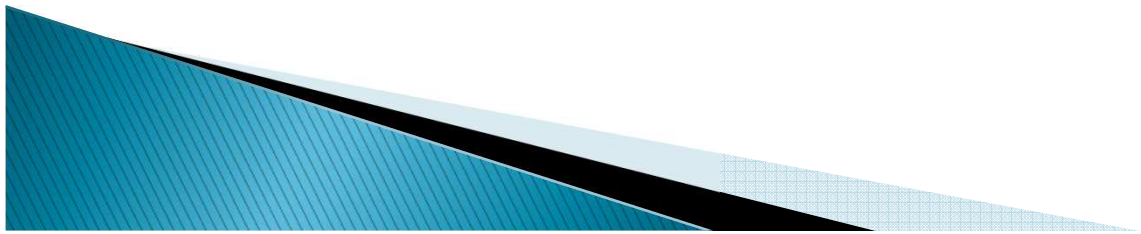
Listas Simplesmente Encadeadas

- Para criar a lista propriamente dita, criaremos a classe Lista, que manipula objetos do tipo NoLista

```
public class Lista {
  NoLista inicio;

  public Lista() {
    this.inicio = null;
  }

   // insere valor no começo da lista

  public void inserir(double valor) {...}

   // insere valor no fim da lista

  public void inserirNoFim(double valor) {...}
}
```

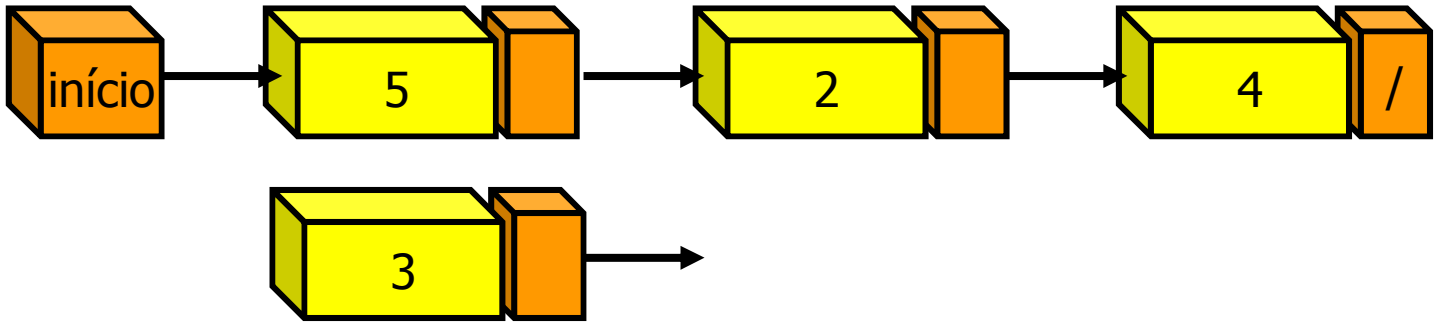
Operações sobre lista encadeada :: Inserção de itens

- Podemos inserir itens:
  - No início de uma lista
  - No final de uma lista
  - No meio de uma lista



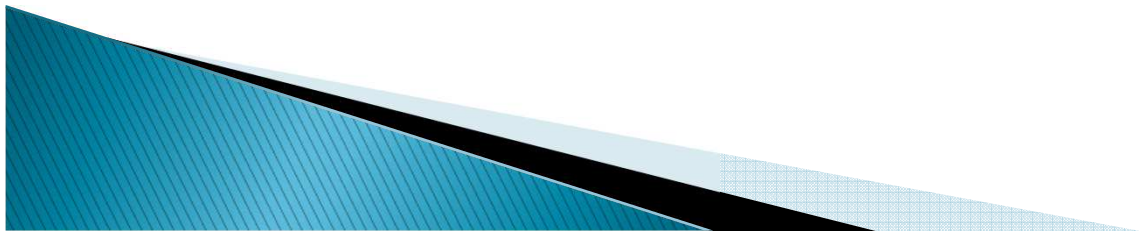
Operações sobre lista encadeada :: Inserção de itens no início

- O endereço armazenado na referência **`início`** deve ser alterado para o endereço do item a ser acrescido à lista.



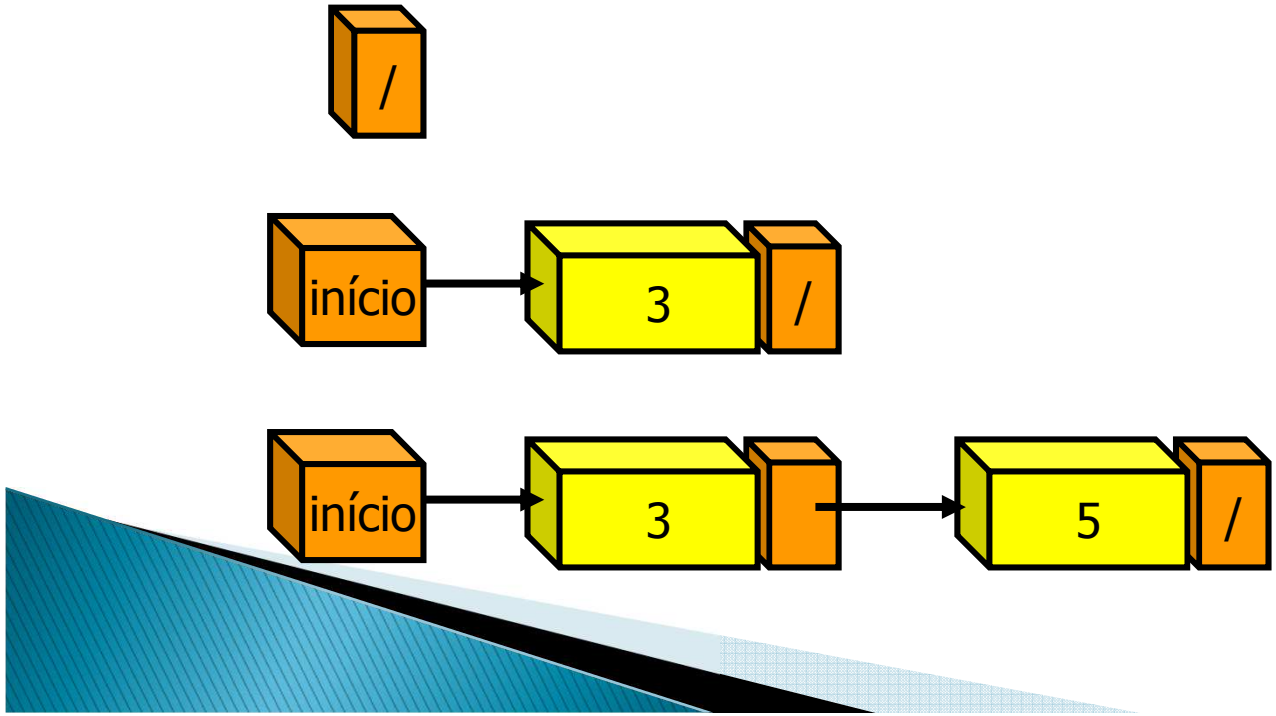
Inserção de itens no início

```
public void inserirNoInicio(double valor) {
  if (this.inicio == null) {
    // lista vazia, então só é preciso criar o nó
    this.inicio = new NoLista(valor);
  } else {

    // cria-se novo no e atualiza o NoLista inicio
    NoLista novoNo = new NoLista(valor);

    novoNo.proximo = this.inicio;

    this.inicio = novoNo;
  }
}
```



Operações sobre lista encadeada :: Inserção de itens no final

- O endereço armazenado em **`início`** será alterado caso a lista esteja vazia ou
- O campo **`proximo`** do último item será alterado.

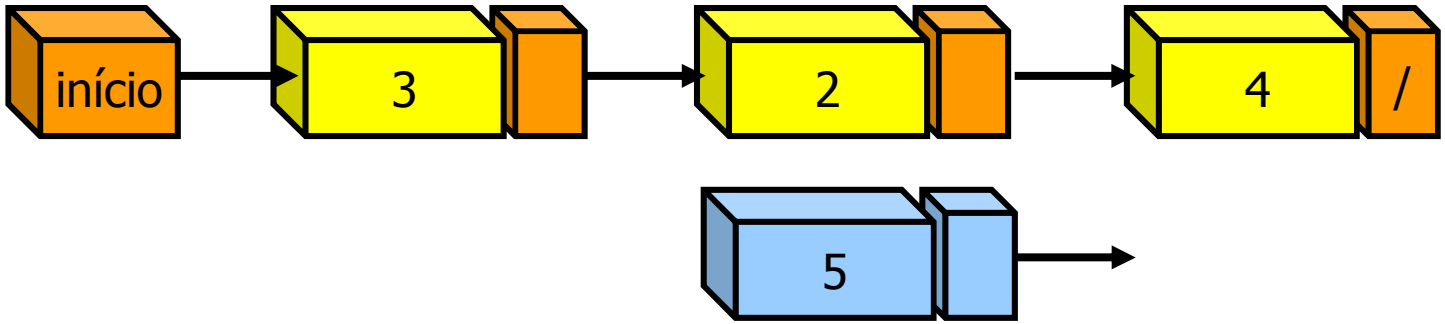


Inserção de itens no final

```
public void inserirNoFim(double valor) {
  if (this.inicio == null) {
    // lista vazia
    this.inicio = new NoLista(valor);
  } else {
    // procura pelo fim da lista
    NoLista atual = this.inicio;
    while (atual.proximo != null)
      atual = atual.proximo;
    // insere o nó no fim da lista
    atual.proximo = new NoLista(valor);
  }
}
```

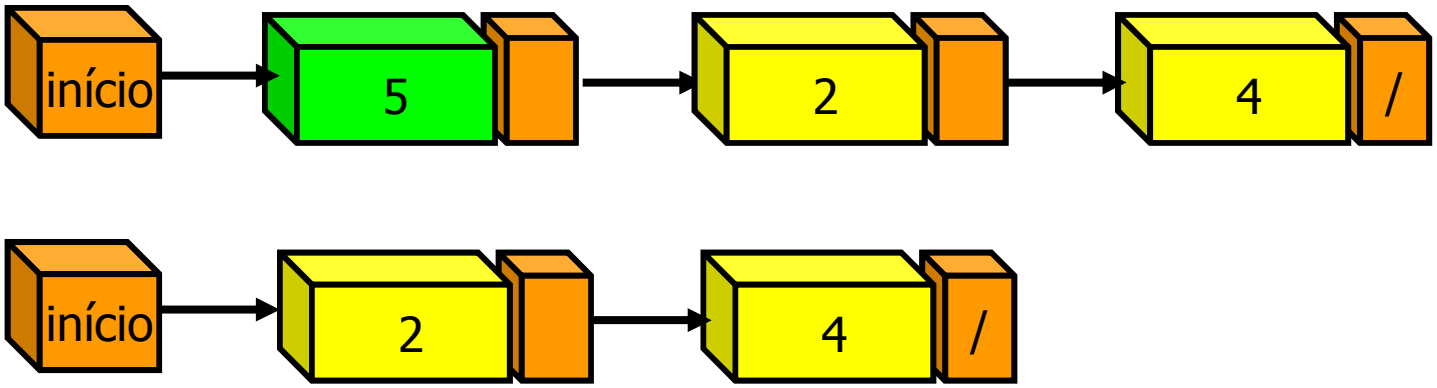
Operações sobre lista encadeada :: Inserção de itens no meio

- Campo **`proximo`** do item a ser inserido recebe o campo **`proximo`** do item posterior
- Campo **`proximo`** do item antecessor recebe o endereço do item a ser inserido



Operações sobre lista encadeada :: Remoção de itens no início

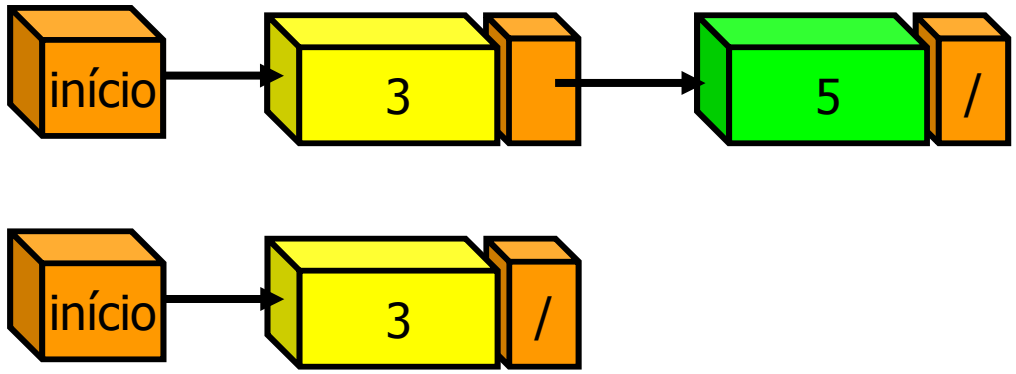
- O endereço armazenado na referência **`início`** deve ser alterado para o endereço do item que segue o primeiro item da lista.

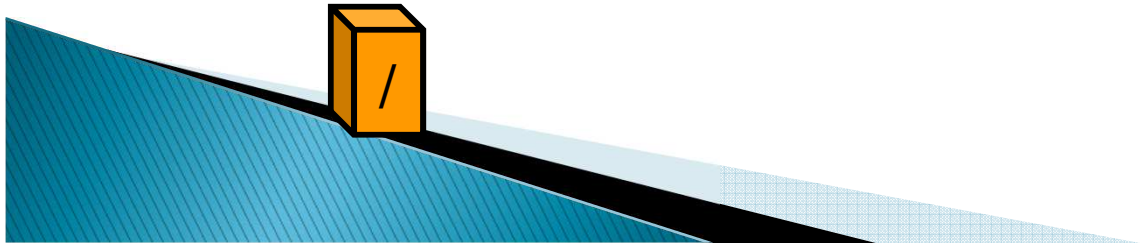


Operações sobre lista encadeada :: Remoção de itens no final

- O campo **`proximo`** do último item será alterado caso a lista contenha mais de um item ou

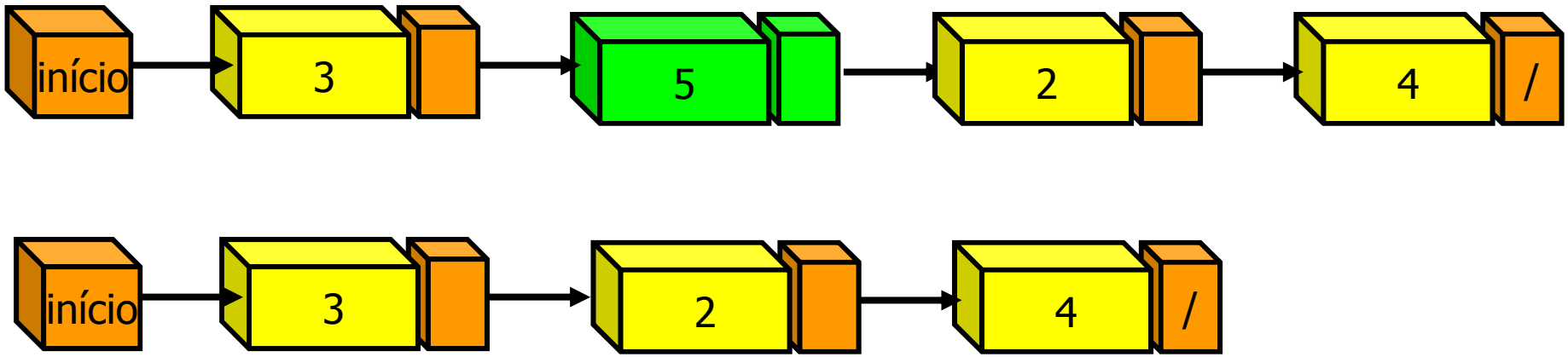
- O endereço armazenado em **`início`** será alterado para null caso tenha somente um elemento.





Operações sobre lista encadeada :: Remoção de itens no meio

- Item antecessor recebe o campo **`proximo`** do item a ser removido

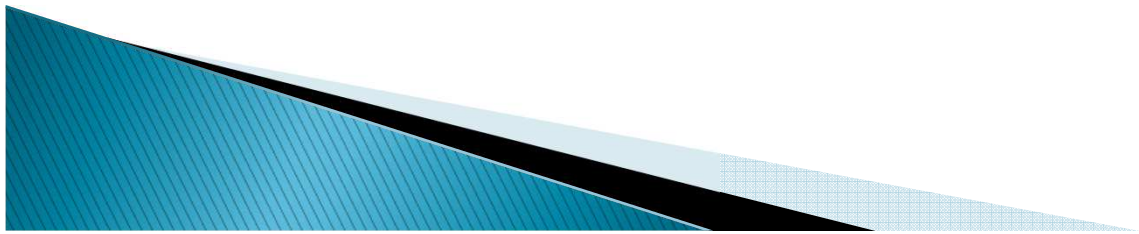


Função retira

```
public void retira (double v) {//Em qualquer posicao

     NoLista ant = null;
     NoLista p = this.inicio;

     while (p != null && p.info != v) {
        ant = p;
        p = p.proximo;
     }
     if (p != null){
       if (ant == null) {
          this.inicio = p.proximo;
       }else {
          ant.proximo = p.proximo;
       }
     }
}
```

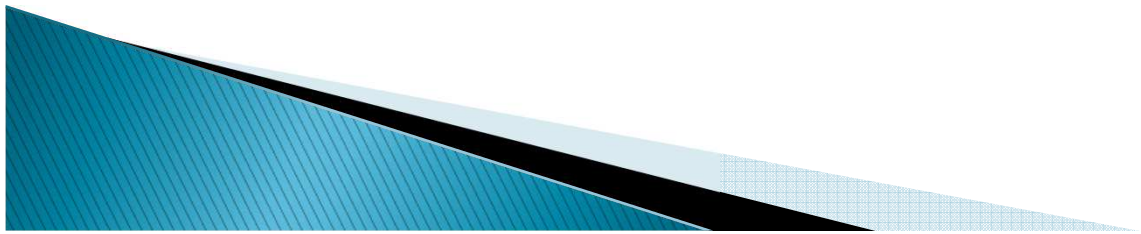
Função de busca

```
public NoLista busca (double v){
    int i=0;
    for (NoLista  p = this.inicio; p!=null; p=p.proximo){
        if(p.info == v){
           System.out.println("\n\nachou “+i+”\n\n");
           return p;
        }
        i++;
    }
    return null;
}
```



Impressão

```
public void imprime (){
  for(NoLista q=this.inicio;q!=null; q=q.proximo)
     System.out.println(q.info);
}
```



Teste

```
public static void main(String[] args){

    Lista l = new Lista();
    l.inserir(20.0);
    l.inserir(44.5);
    l.inserir(33.3);
    l.inserir(20.9);
    l.imprime();
    NoLista n = l.busca(20.9);//Busca
    if (n != null){
        System.out.println("Encontrado:"+n.info);
        l.retira(n.info);
    }
    System.out.println("Configuracao da lista:");
    l.imprime();
    //Libera memoria
    l = null;

}
```

Insere ordenado

```
public void insereOrdenado (double valor)
{
        NoLista novoNo = new NoLista(valor );

        NoLista ant = null;
        NoLista p = this.inicio;

        while (p != null && p.info < valor) {
                ant = p;
                p = p.proximo;
        }
        if (ant == null) {
                novoNo.proximo = this.inicio;
                this.inicio = novoNo;
        } else {
                novoNo.proximo = ant.proximo;
                ant.proximo = novoNo;
        }
}
```

Genéricos

- Adicionado a C# 2.0 e posteriormente a Java 5
- Classes Genéricas, que utilizam o conceito de "parâmetros tipo"<..>
- Lista com Genéricos: cada lista armazena um tipo específico, sem precisar criar código novo para cada tipo

Sem Genéricos:

```
class NoListaI{
  int valor;
  NoLista next;
}
```

```
class NoListaS{
  String nome;
  NoLista next;
}
```

Com Genéricos:

```
class NoLista<E> {
  E elemento;
  NoLista<E> next;
}
```

Listas com Genéricos

```
class Lista<E> {
  NoLista<E> inicio;

  public Lista() {
    this.inicio = null;
  }

  public void inserir(E elemento) {
    if (this.inicio == null) {
      this.inicio = new NoLista<E>(elemento);
    } else {

      NoLista<E> novoNo = new NoLista<E>(elemento);

      novoNo.next = this.inicio;

      this.inicio = novoNo;
    }
  }

}
```