# Implementação da Lista Ligada [Lista Encadeada]

## Estrutura de Dados e Armazenamento

**Crie um projeto no IntelliJ chamado lista-ligada**

1) Implementar a **classe Node**, contendo:

Atributos (encapsulados):

info – tipo int

next – tipo Node

Construtor que recebe o valor de info do nó:

- atribui o valor recebido como argumento ao atributo info
- atribui null para next

Setter e Getters

2) Implementar a classe ListaLigada, contendo:

Atributo (encapsulado):

head – tipo Node

Construtor (não recebe argumentos):

Cria um nó, com valor de info igual a 0 ou null (dependendo se info do Node for int ou Integer) e atribui esse nó para head (nó cabeça da lista)

Getter do head

```
ListaLigada()
  head← new Node(0);
```

Exemplo: Supondo que criamos um objeto lista da classe ListaLigada

```
ListaLigada lista = new ListaLigada();
```

head

| 0 | **null** |
|---|---|
| info | next |

**Métodos:**

a) void insereNode (int valor)

- cria um objeto **novo** da classe Node com info igual ao valor
- insere o novo nó na lista encadeada

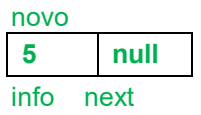
atribui para next de **novo** o conteúdo de head.next

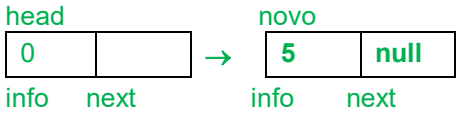
atribui para next de head o **novo**

```
public void insereNode(int valor)
  Node novo ← new Node(valor);
  novo.next←head.next;          // novo.setNext(head.getNext())
  head.next← novo;              // head.setNext(novo)
```

Exemplo: Supondo que queremos inserir o valor 5 na lista.

```
lista.insereNode(5);
```

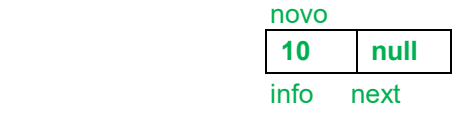


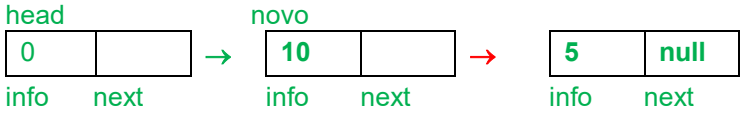

Exemplo: Supondo que agora queremos inserir o valor 10 na lista.

```
lista.insereNode(10);
```

Antes da inserção:




Após inserção:




b) void exibe ( )

- percorre a lista encadeada, exibindo seus valores (os infos dos nós)
  - atribui para a variável atual (do tipo Node) o conteúdo de head.next
  - enquanto atual for diferente de null,
    - exibe o valor do info do nó atual,
    - atribui para atual o conteúdo de atual.next

```
public void exibe()
  Node atual←head.next;
  enquanto atual ≠ null faça
   início
     exibe(atual.info);
     atual←atual.next;
   fim
```

c) Node buscaNode(int valor)

- percorre a lista encadeada, verificando se existe um nó com o valor passado como argumento
- Se existir, devolve o endereço desse nó, senão devolve null

```
public Node buscaNode(int valor)
  Node atual←head.next;
  enquanto atual ≠ null faça
   início
     se atual.info = valor
     então retorna atual;
     senão atual←atual.next;
   fim
  retorna null;
```

d) boolean removeNode(int valor)

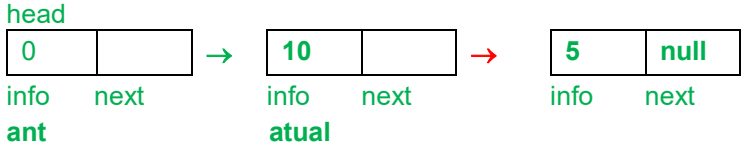
- percorre a lista encadeada, verificando se existe um nó com o valor passado como parâmetro
- Se existir, remove esse nó da lista e devolve true
- Se não existir, devolve false

```
public boolean removeNode(int valor)
  Node ant ← head;
  Node atual ← head.next;
  enquanto atual ≠ null faça
    início
      se atual.info = valor
      então início
              ant.next←atual.next;      // ant.setNext(atual.getNext());
              retorna true;
            fim
      senão início
              ant← atual;
              atual←atual.next;
            fim
    fim
  retorna false;
```

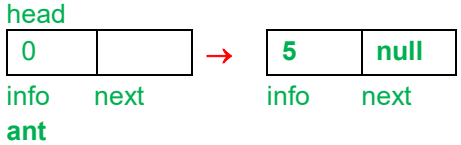
Exemplo: Supondo que queremos remover o valor 10 da lista.

```
lista.removeNode(10);
```

Antes da remoção:




Após remoção:



e) int getTamanho( )

- percorre a lista encadeada, calcula e devolve o tamanho da lista

```
public int getTamanho()
  Node atual←head.next;
  int tam ← 0;
  enquanto atual ≠ null faça
    início
      tam←tam + 1;
      atual←atual.next;
    fim
  retorna tam;
```

3) Na classe Main, no método main:

Testar a classe ListaLigada, criando um objeto ListaLigada, e inserindo vários valores.

- Exiba os valores para verificar se está inserindo corretamente.
- Depois teste os métodos buscaNode e removeNode, sempre exibindo os valores da lista, para verificar se está executando corretamente.
- Teste também o método getTamanho.