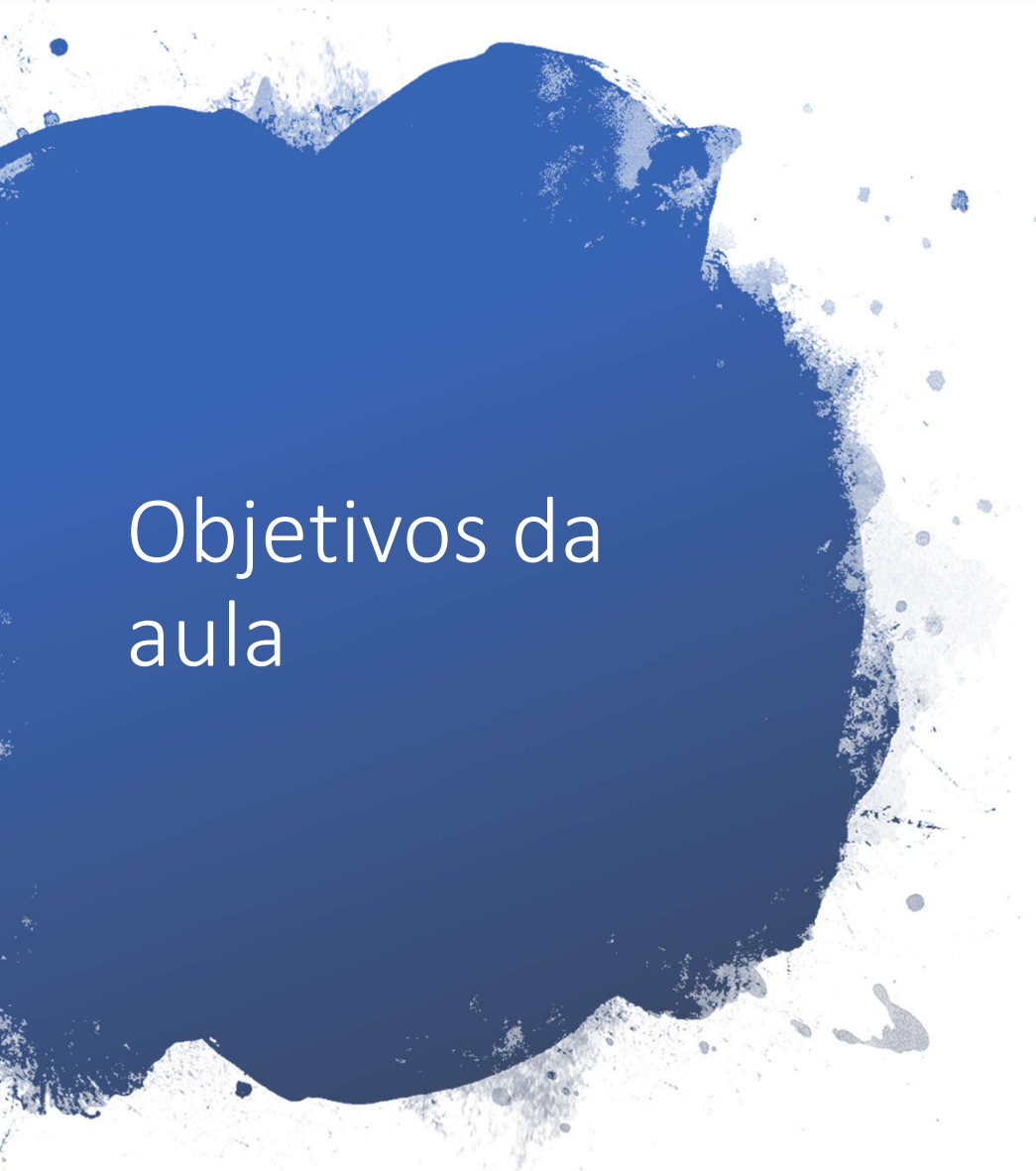


Recursividade e listas encadeadas

Prof. Eduardo Hidenori
Enari

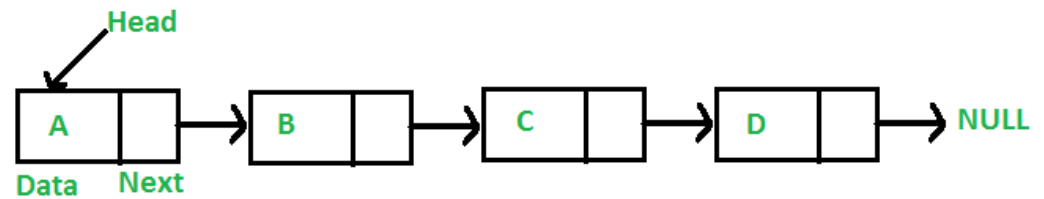


Objetivos da aula

- Discutir o uso da recursividade como estratégia de manipulação de listas encadeadas;
- Preparar os alunos para trabalhar com estruturas de dados mais complexas.

Listas encadeadas:

- Os valores são armazenados nos nós da lista;
- Cada nó possui uma ligação com o próximo nó da lista
- A lista necessita manter apenas a referência para o nó HEAD.



Recursividade:

- Depende diretamente do problema:
 - Divisão em subproblemas
 - Subproblemas tem as mesmas características do problema original
 - Um ou mais subproblemas tem solução conhecida e definem quando não será necessário continuar a divisão

Problema 1

- Construa os métodos que retornem a quantidade de nós da Lista Encadeada.

Solução interativa:

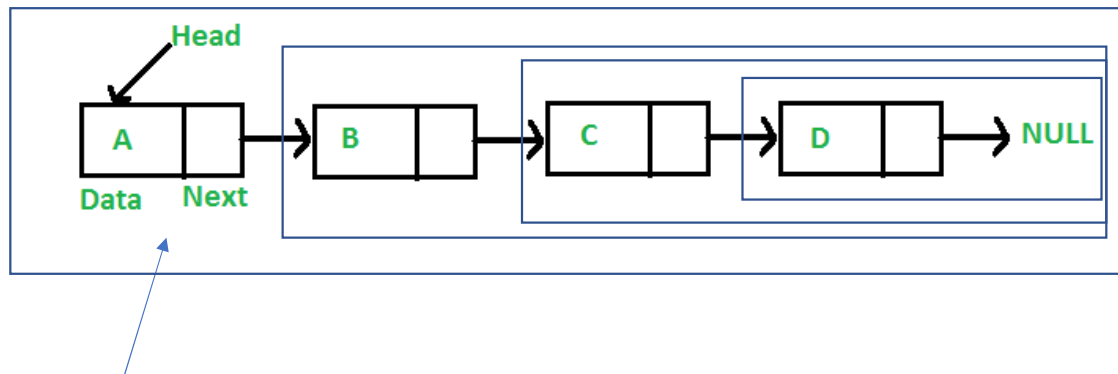
Se houver HEAD válido (diferente de NULL)

- Visitar os nós, começando pelo HEAD da Lista Encadeada;
- A cada nó visitado, uma variável contadora é incrementada de uma unidade.
- Após visitar todos os nós, retornar o valor armazenado na variável contadora

Se não houver HEAD válido,

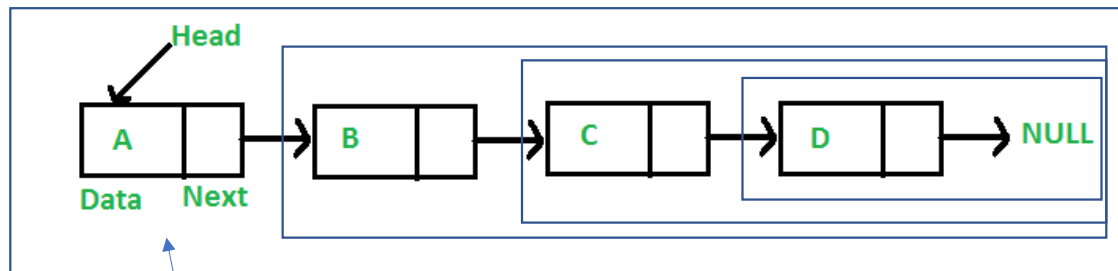
- Retornar 0;

Solução Recursiva – Classe Lista:



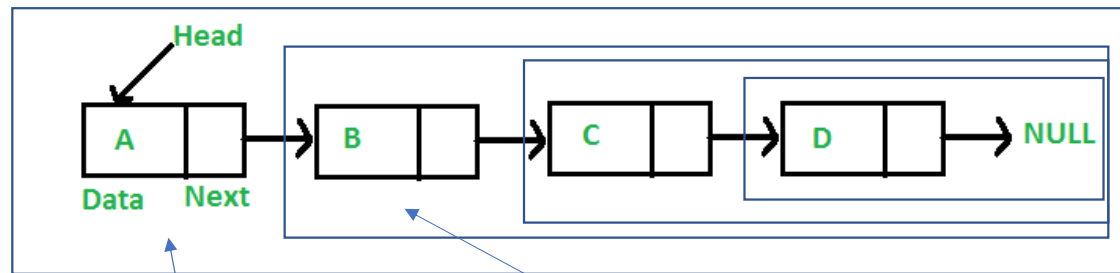
- Se HEAD for válido (HEAD diferente de NULL):
 - Retorna o tamanho da lista a partir de HEAD - head.size()
- Se não houver HEAD válido:
 - Retorna 0;

Solução Recursiva – Classe Lista:



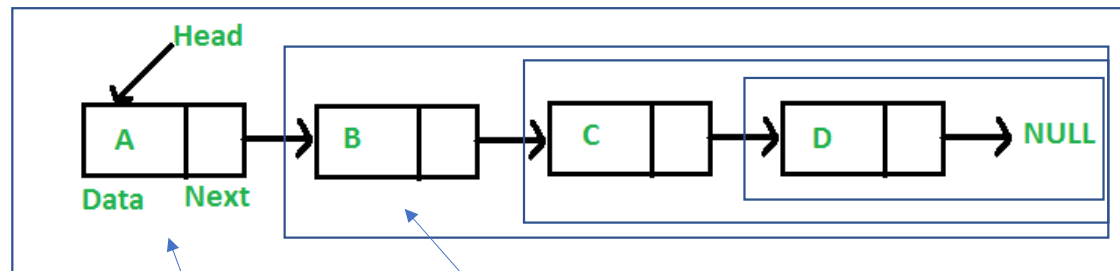
```
public int size(){  
    if(head != null) return head.size();  
    else return 0;  
}
```


Solução Recursiva – Classe Node:



- Método Size do Node
 - Quantidade de nós é 1 + a quantidade de nós da lista que começa no Next do HEAD

Solução Recursiva – Classe Node:



```
public int size(){  
    int s=0;  
    if(next!=null) s=next.size();  
    return 1+s;  
}
```

Método Classe Lista:

```
public int size(){  
    if(head != null) return head.size();  
    else return 0;  
}
```

Método Classe Node:

```
public int size(){  
    int s=0;  
    if(next!=null) s=next.size();  
    return 1+s;  
}
```

Problema 2

- Construa os métodos que retornem a soma dos valores armazenados nos nós da Lista Encadeada.

Solução iterativa:

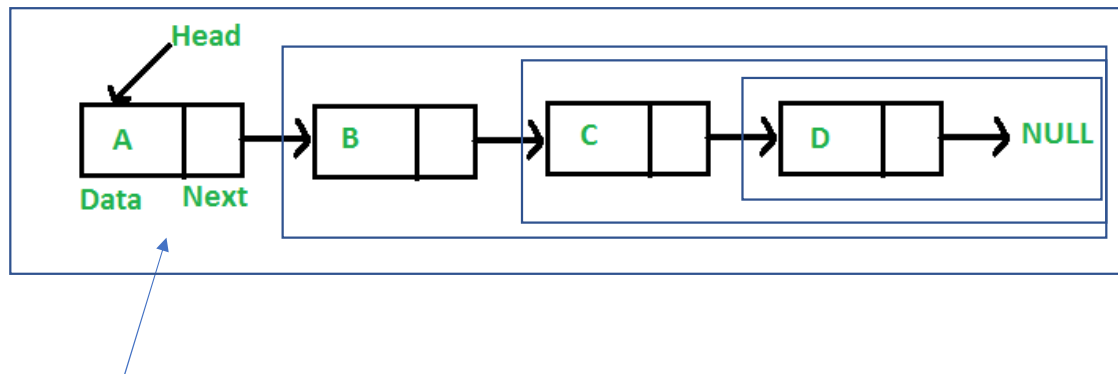
Se houver HEAD válido (diferente de NULL)

- Visitar os nós, começando pelo HEAD da Lista Encadeada;
- A cada nó visitado, uma variável acumuladora é incrementada do valor armazenado no nó.
- Após visitar todos os nós, retornar o valor armazenado na variável acumuladora.

Se não houver HEAD válido,

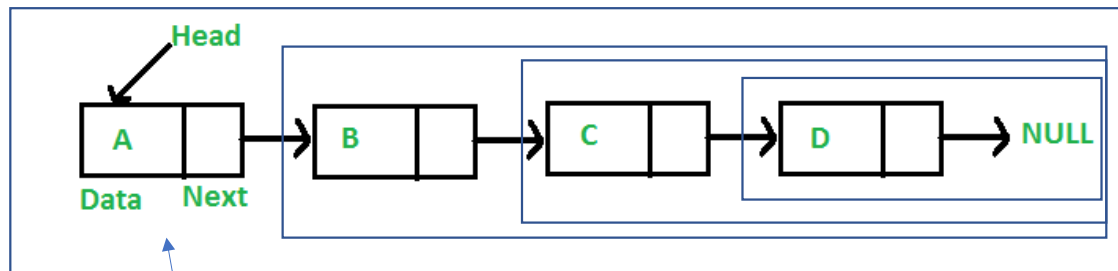
- Retornar 0;

Solução Recursiva – Classe Lista:



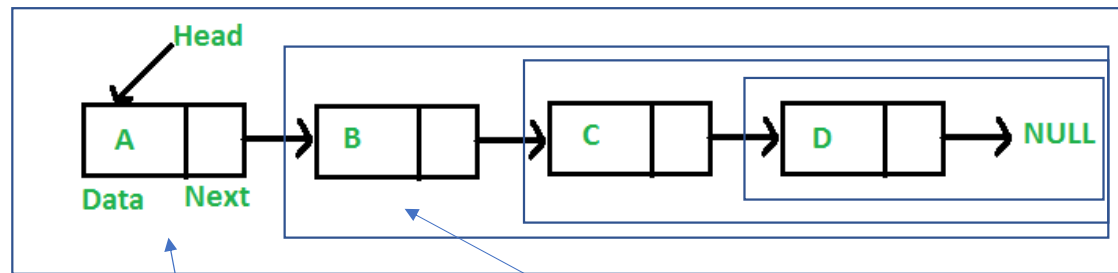
- Se HEAD for válido (HEAD diferente de NULL):
 - Retorna a soma dos valores da lista a partir de HEAD - head.soma()
- Se não houver HEAD válido:
 - Retorna 0;

Solução Recursiva – Classe Lista:



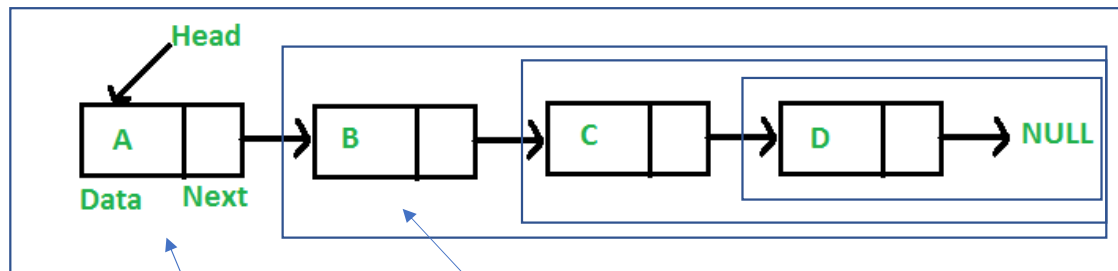
```
public int soma(){  
    if(head != null) return head.soma();  
    else return 0;  
}
```

Solução Recursiva – Classe Node:



- Método Size do Node
 - Quantidade de nós é Data + a soma dos nós da lista que começa no Next do HEAD

Solução Recursiva – Classe Node:



```
public int soma(){  
    int s=0;  
    if(next!=null) s=next.soma();  
    return x+s;  
}
```

Problema 3

- Construa os métodos que retornem o maior dos valores armazenados nos nós da Lista Encadeada.

Solução iterativa:

Se houver
HEAD válido
(diferente de
NULL)

Visitar os nós, começando pelo HEAD da Lista Encadeada;

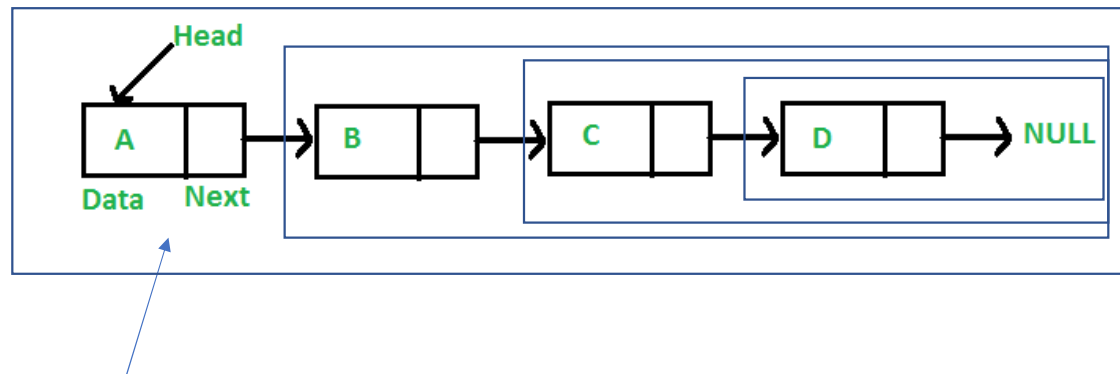
A cada nó visitado, comparar o valor do nó com o valor em uma variável MAIOR. Se o valor do nó for maior que o valor da variável MAIOR, MAIOR deve receber o valor do nó.

Após visitar todos os nós, retornar o valor armazenado na variável MAIOR.

Se não
houver HEAD
válido,

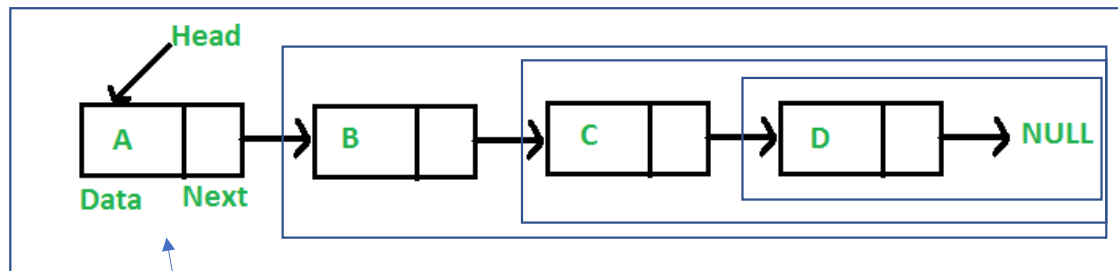
Retornar 0;

Solução Recursiva – Classe Lista:



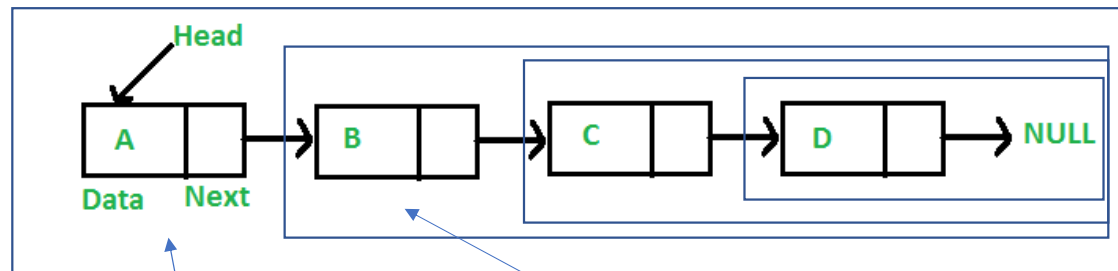
- Se HEAD for válido (HEAD diferente de NULL):
 - Retorna o maior dos valores da lista a partir de HEAD - head.maior()
- Se não houver HEAD válido:
 - Retorna 0;

Solução Recursiva – Classe Lista:



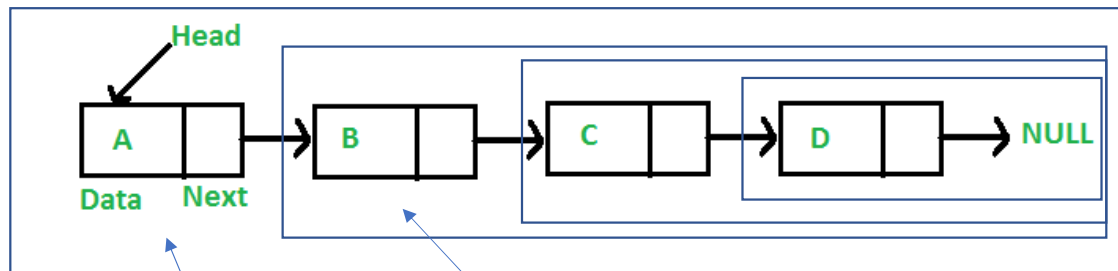
```
public int maior(){  
    if(head != null) return head.maior();  
    else return 0;  
}
```

Solução Recursiva – Classe Node:



- Método Size do Node
 - Se houver Next, guardar o maior valor da lista a partir de next: `next.maior()`
 - Comparar o valor de Data com o valor retornado por `next.maior()` e retornar o que tiver maior valor.

Solução Recursiva – Classe Node:

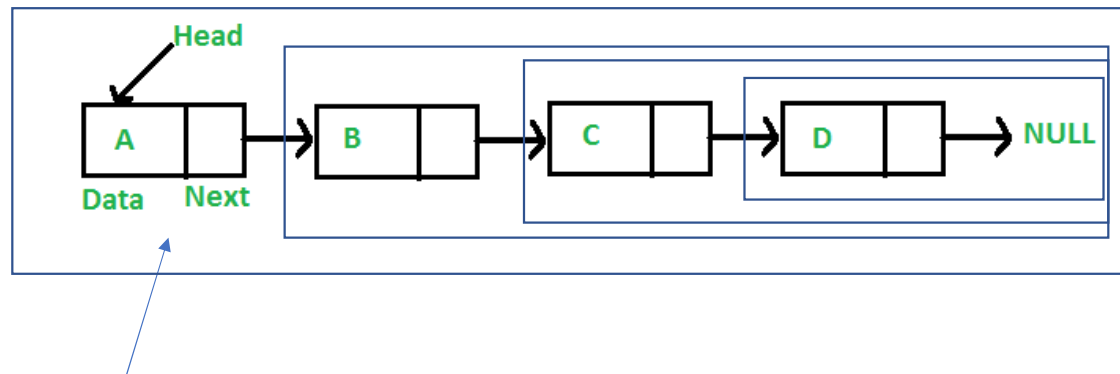


```
public int maior(){  
    int m=-500;  
    if(next!=null) m=next.maior();  
    if(m>x) return m;  
    else return x;  
}
```

Problema 4

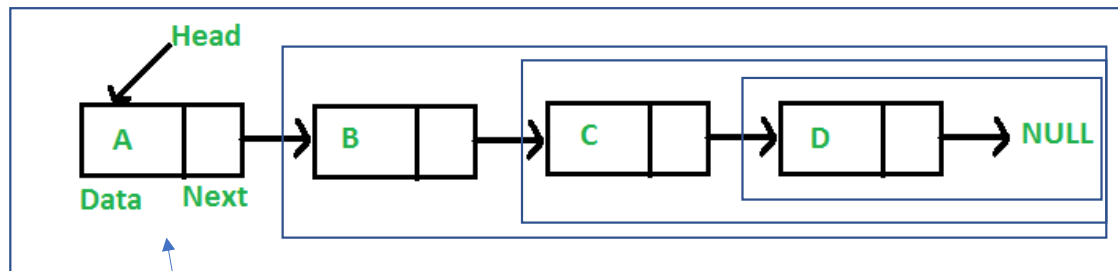
- Construa os métodos que imprima na tela os valores armazenados nos nós da Lista Encadeada.

Solução Recursiva – Classe Lista:



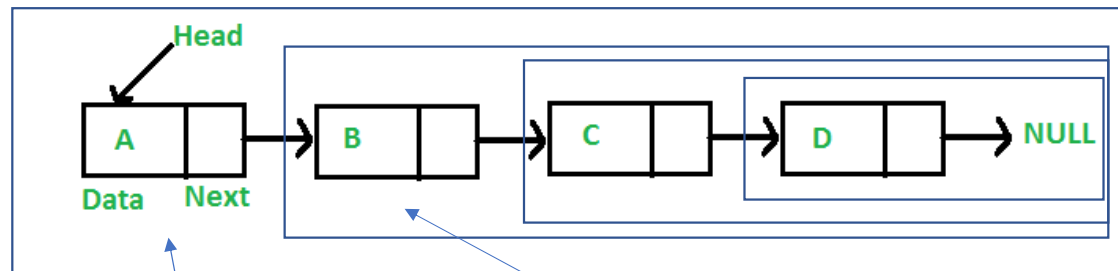
- Se HEAD for válido (HEAD diferente de NULL):
 - Imprime os valores dos nós a partir de HEAD – head.show()
- Se não houver HEAD válido:
 - Não faz nada;

Solução Recursiva – Classe Lista:



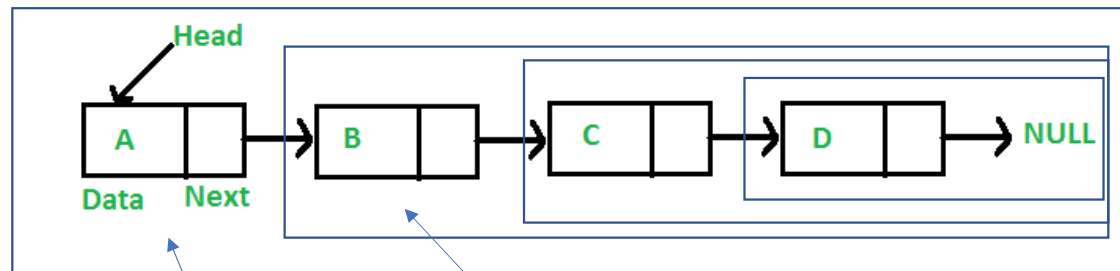
```
public void show(){  
    if(head != null) head.show();  
}
```

Solução Recursiva – Classe Node:



- Método Size do Node
 - Se houver Next, guardar o maior valor da lista a partir de next: `next.maior()`
 - Comparar o valor de Data com o valor retornado por `next.maior()` e retornar o que tiver maior valor.

Solução Recursiva – Classe Node:



```
public void show(){  
    System.out.println(x);  
    if(next != null) next.show();  
}
```

Problema 5

- Construa os métodos que, dado um valor inteiro como parâmetro, retornem a quantidade de valores armazenados nos nós da Lista Encadeada que sejam iguais ao valor passado.

Problema 6

- Construa os métodos que imprimam na tela os valores da lista começando pelo último nó e terminando no valor armazenado em HEAD.

Problema 7

- Construa os métodos que imprimam na tela os valores armazenados nos nós da lista que ocupem posições pares, considerando que o nó HEAD ocupa a primeira posição (1).