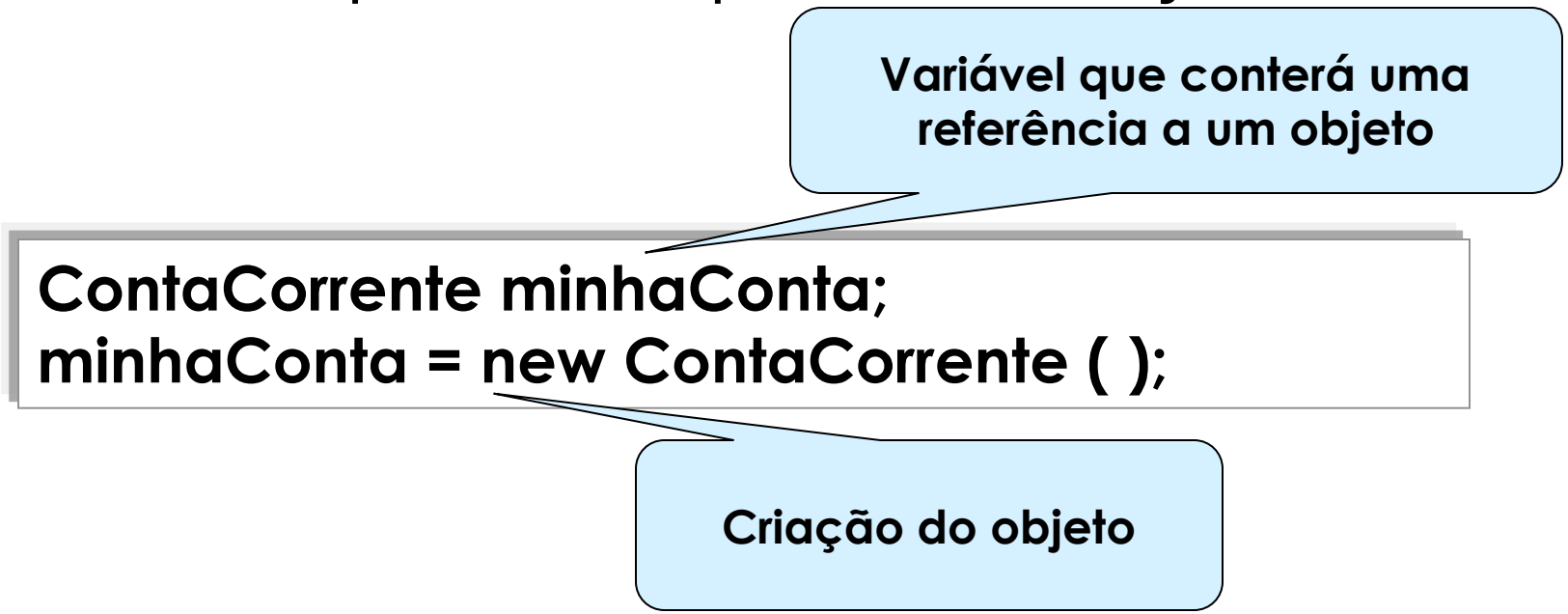
# Listas
# Simplesmente Encadeadas

# Criando um objeto

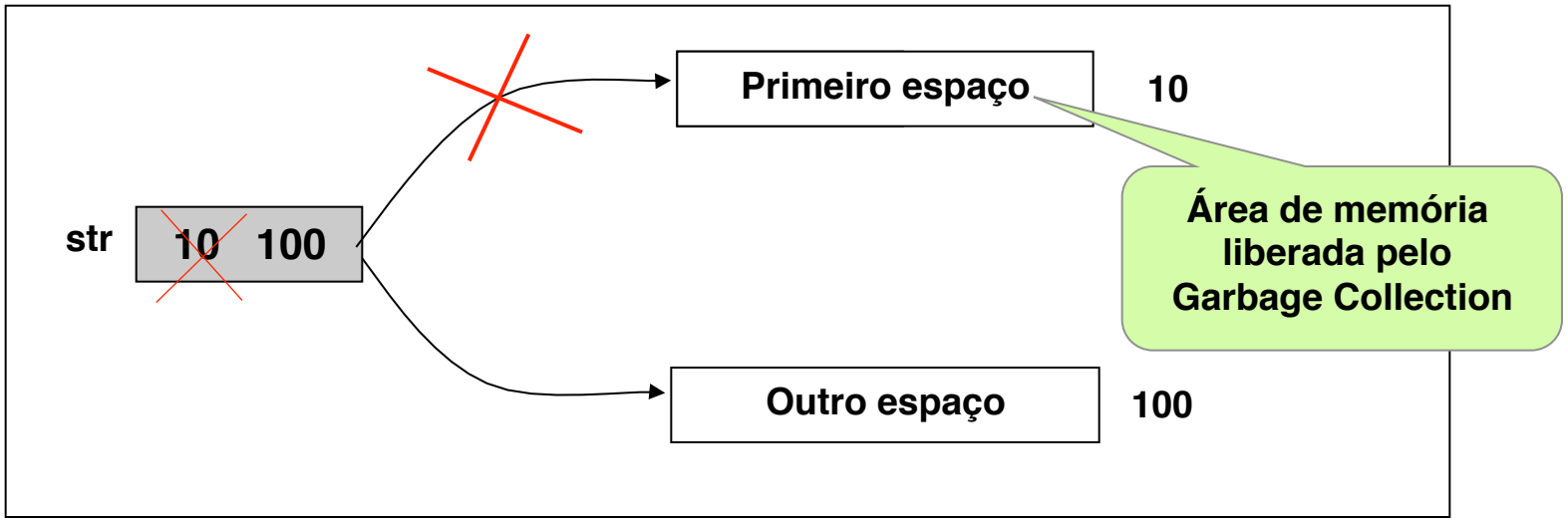
- Objeto é uma instância de uma classe;
- Usamos o operador *new* para criar um objeto.




**ContaCorrente minhaConta = new ContaCorrente ( );**

# Garbage Collection

```
String str = “Primeiro espaço”;
System.out.println (“Usando memória original: “+str);
str = “Outro espaço”;
System.out.println (“Usando outro espaço de memória: “+str);
```




**System.gc();** → **Não obriga a limpar, mas “pede” para que o Garbage Collection limpe se possível**

# Listas: Tipo de Armazenamento

- O tipo de armazenamento de uma lista linear pode ser classificado de acordo com a posição relativa (sempre contígua ou não) na memória de dois nós consecutivos na lista.

- Lista linear com alocação seqüencial de memória
  - Nós em posições contíguas de memória
  - Geralmente representado por arrays
  - Útil para implementar filas e pilhas (variáveis para controlar fim e início)

# Listas: Tipo de Armazenamento

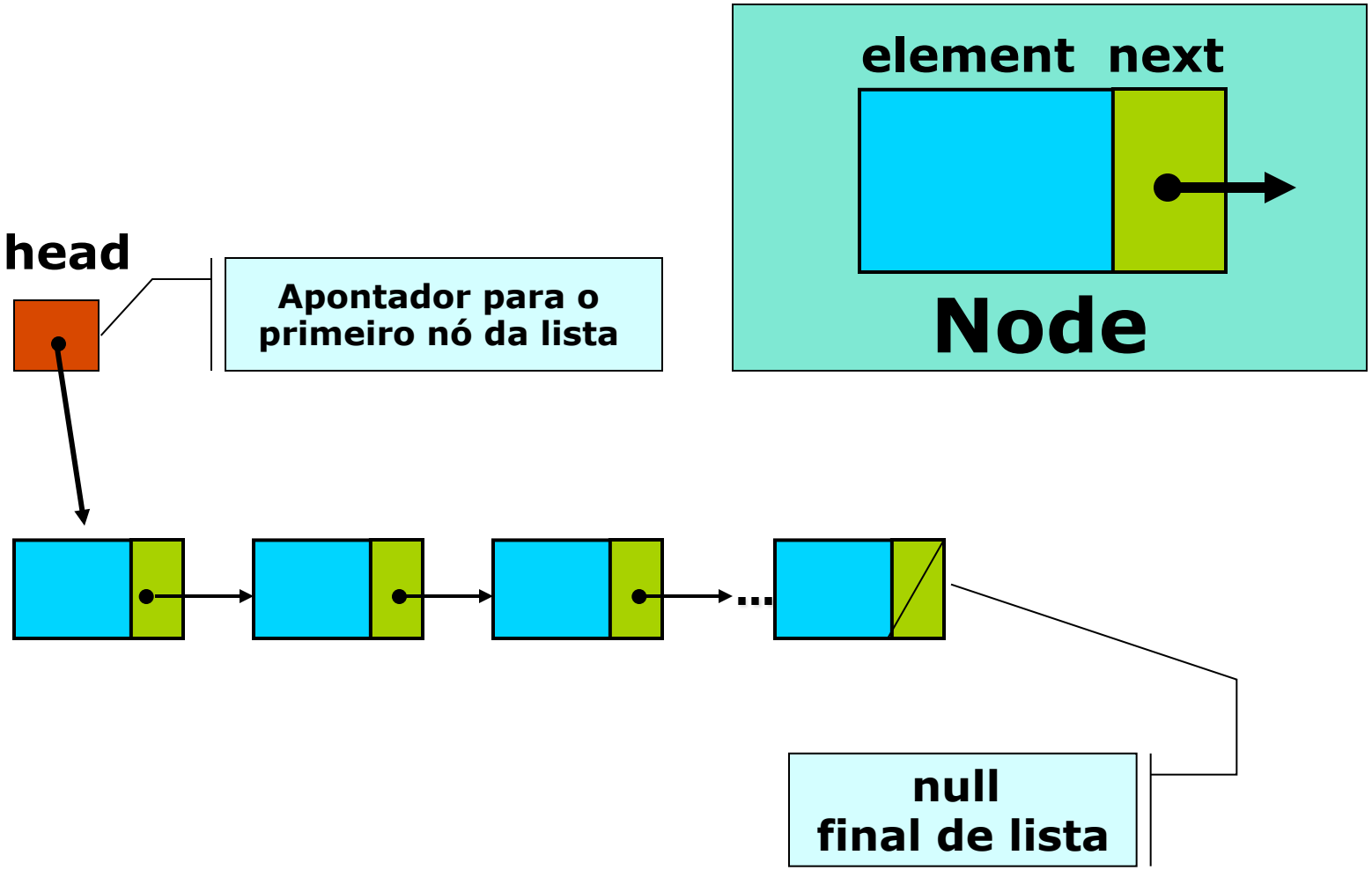
- Lista linear com alocação encadeada
  - Posições de memória são alocadas a medida que são necessárias
  - Nós encontram-se aleatoriamente dispostos na memória e são interligados por ponteiros, que indicam a próxima posição da tabela
    - Nós precisam de um campo a mais: campo que indica o endereço do próximo nó.

# Listas Simplesmente Encadeadas

- Uma lista simplesmente encadeada é uma sequência de objetos alocados dinamicamente, onde cada objeto faz referência ao seu sucessor na lista
- Lista encadeada básica:
  - possui variável *head* que referencia para o primeiro elemento da lista
  - cada Objeto refere a seu sucessor
  - ultimo elemento contém a referência *null* (para indicar que não referencia nenhum outro).

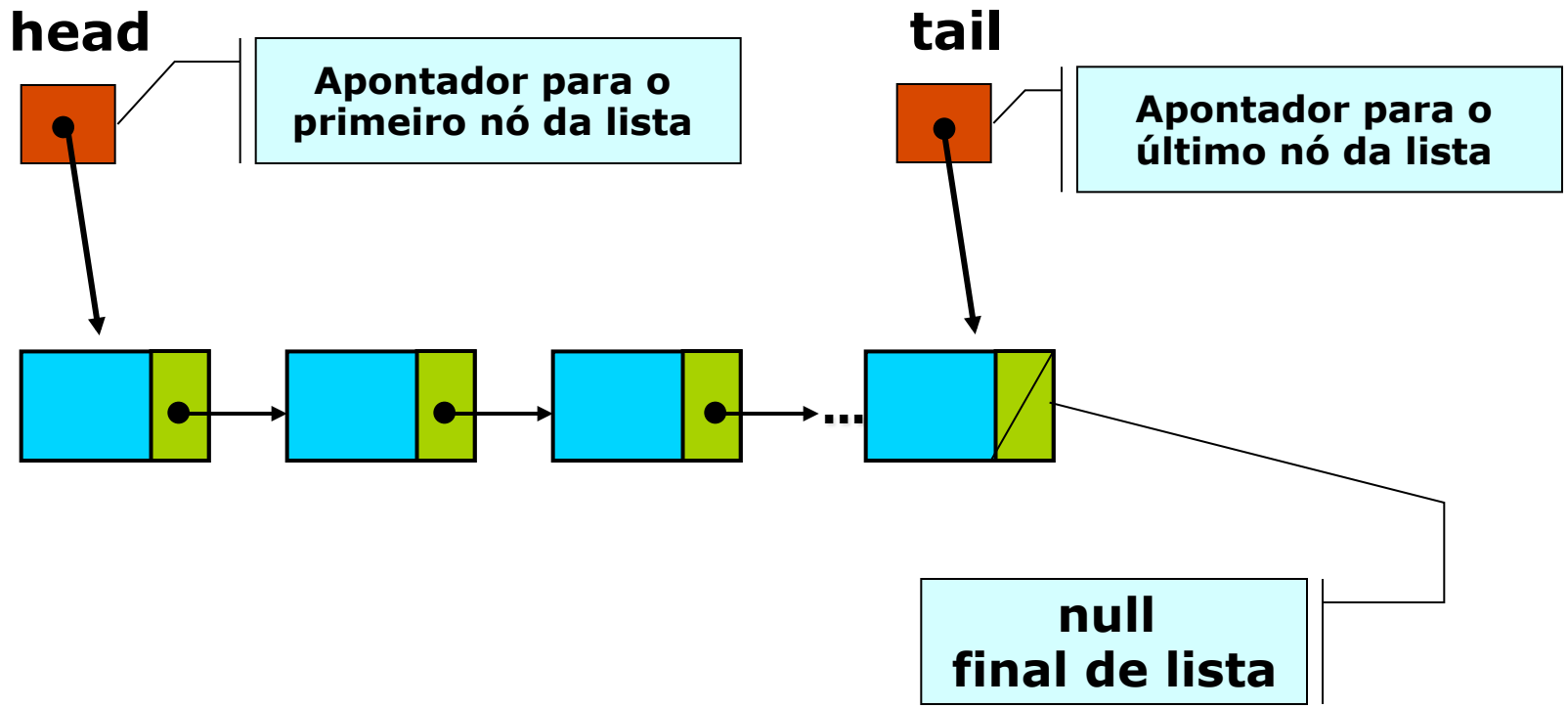
**Ineficiente: se queremos inserir um elemento no final da lista, temos que localizar o último elemento: para tanto é necessário percorrer todos os elementos da lista.**

# Lista Encadeada Básica



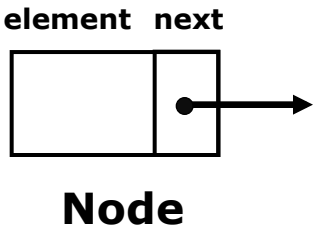
# Lista encadeada com referência ao último elemento da lista

- Como tornar mais eficiente:
  - utilizar uma segunda variável, chamada *tail,* que referencia o último elemento da lista.
  - eficiência obtida a custa do espaço adicional



# Classe Node

```
public class Node<E> {
        protected E element;
        protected Node<E> next;

        public Node(E e) {
                element = e;
                next = null;
        }

        public E getElement() {
                return element;
        }

        public Node<E> getNext() {
                return next;
        }

        public void setElement(E e) {
                element = e;
        }

        public void setNext(Node<E> n) {
                next = n;
        }
}
```



# Operações sobre lista

- **public boolean isEmpty()**
  - verifica se a lista está vazia
- **public E get(int n)**
  - retorna o elemento contido na posição n, sem removê-lo
- **public int numElements()**
  - retorna o número de elementos armazenados na lista
- **public void insertFirst( E element )**
  - insere element na frente da lista
- **public void insertLast( E element )**
  - insere element no final da lista
- **public void insert( E element, int pos )**
  - insere element na posição especificada por pos
- **public E removeFirst()**
  - remove e retorna o primeiro elemento da lista
- **public E removeLast()**
  - remove e retorna o último elemento da lista
- **public E remove(int pos)**
  - remove e retorna o elemento da posição pos da lista
- **public int search (E element)**
  - retorna a posição na lista do elemento element

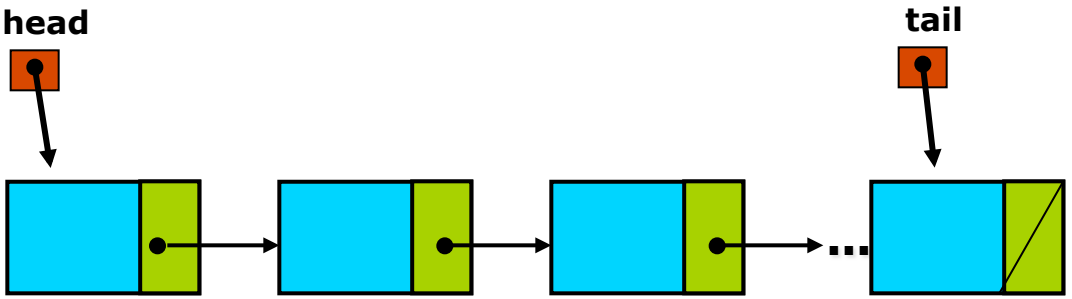
# Class SLinkedList

```
public class SinglyLinkedList<E> implements List<E> {
        private Node<E> head;
        private Node<E> tail;
        private int numElements;

        public SinglyLinkedList() {
            head = tail = null;
            numElements = 0;
        }

        public int numElements() {

        }

        public boolean isEmpty() {

        }

        public boolean isFull() {

        }
```

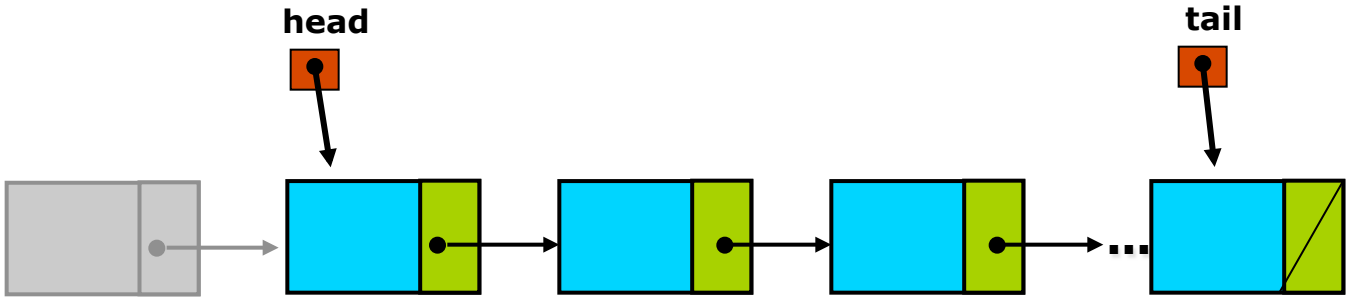
# Inserção em lista simplesmente encadeada

**- Na cabeça da lista:**

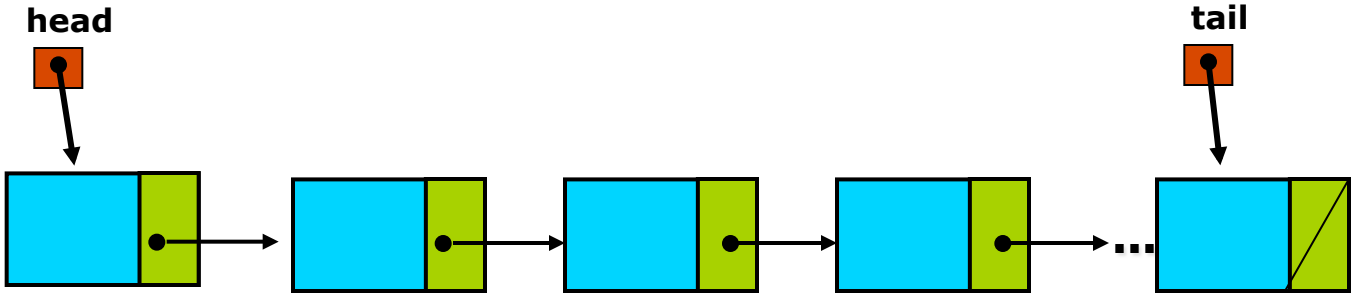
a




b




c



# Inserção em lista simplesmente encadeada

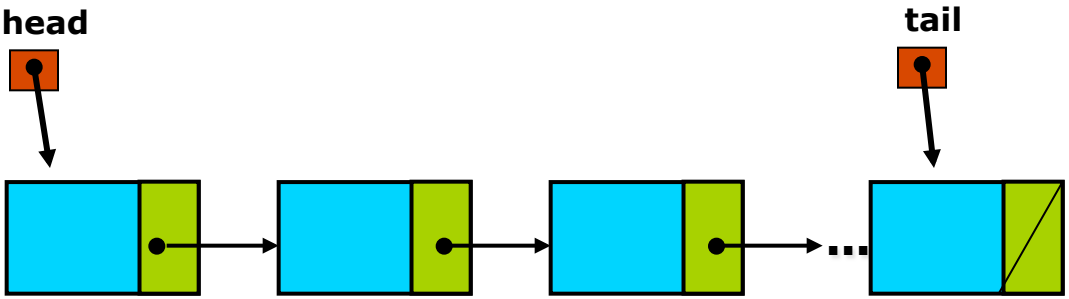
**- Na cabeça da lista:**

```
public void insertFirst(E element) {




}
```

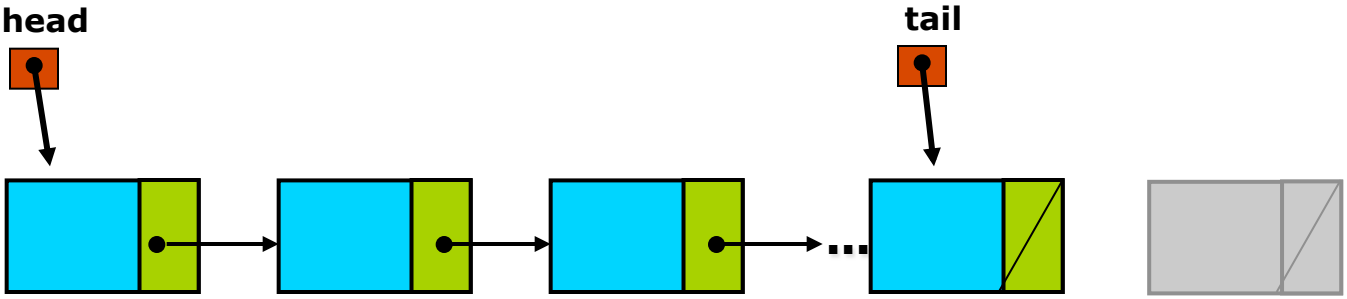
# Inserção em lista simplesmente encadeada

**- Na cauda da lista:**

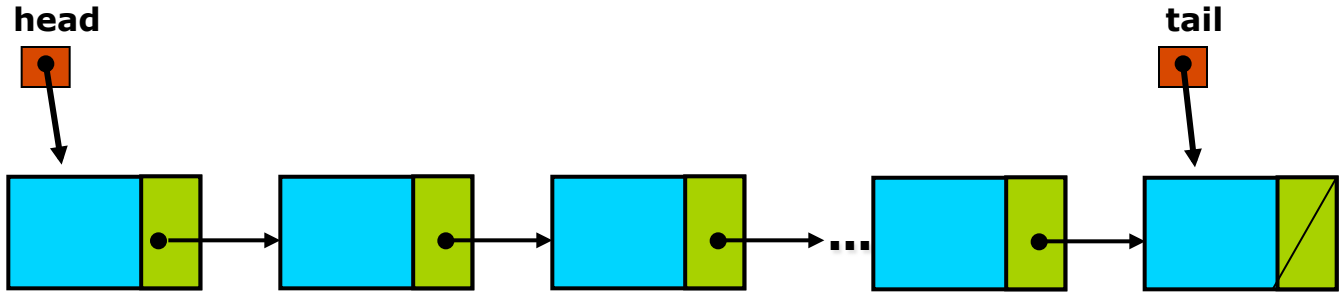
a




b




c



# Inserção em lista simplesmente encadeada

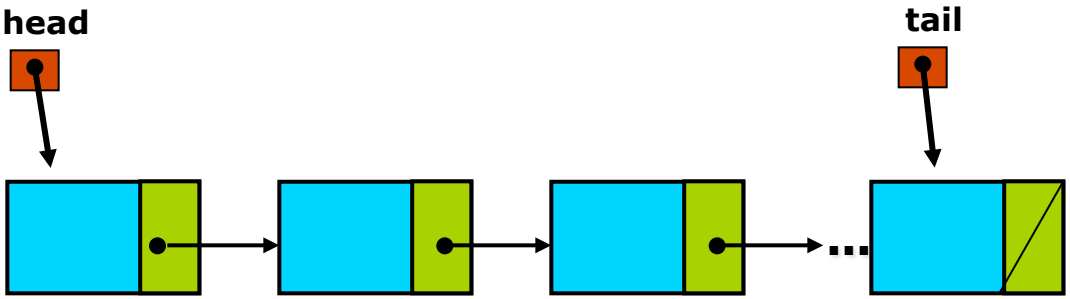
**- Na cauda da lista:**

```
public void insertLast(E element) {




}
```

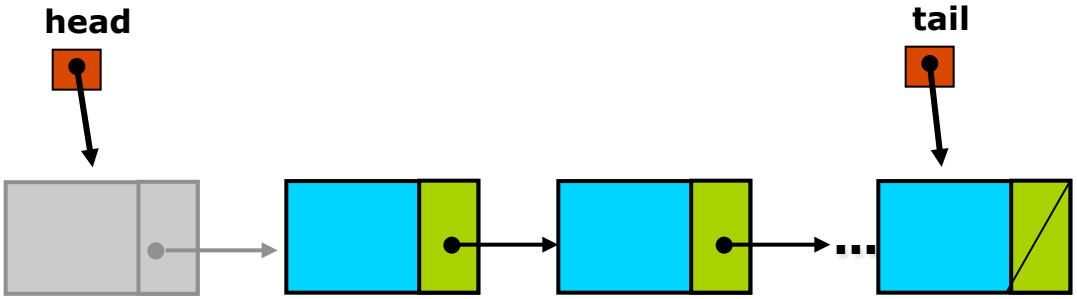
# Remoção em lista simplesmente encadeada

**- Da cabeça da lista:**

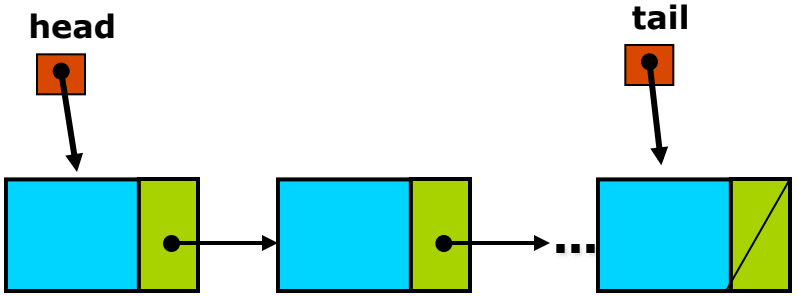
**a**




**b**




**c**



# Remoção em lista simplesmente encadeada

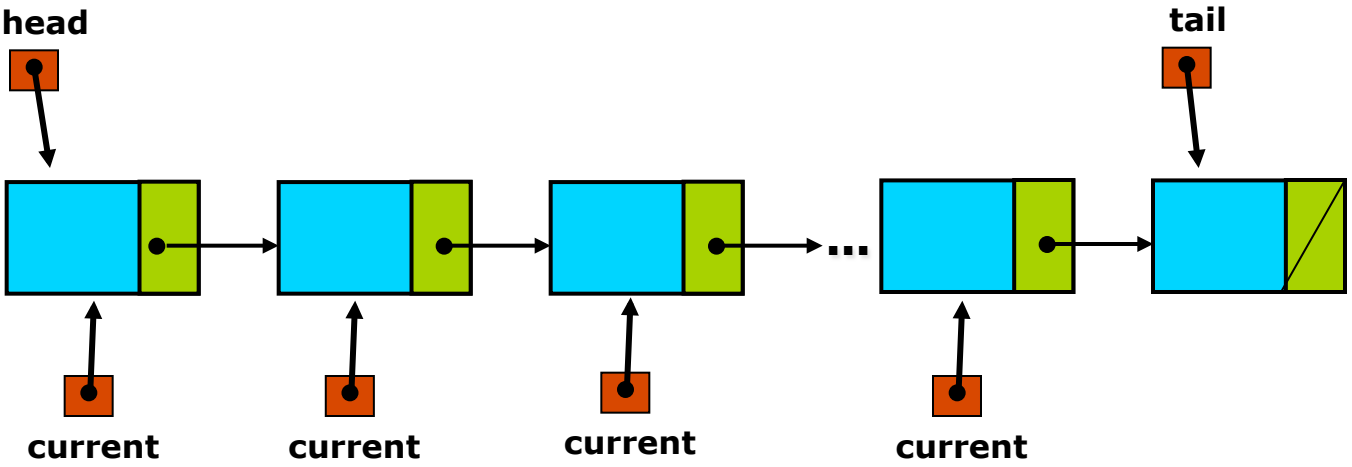
**- Da cabeça da lista:**

```
public E removeFirst() {



}
```

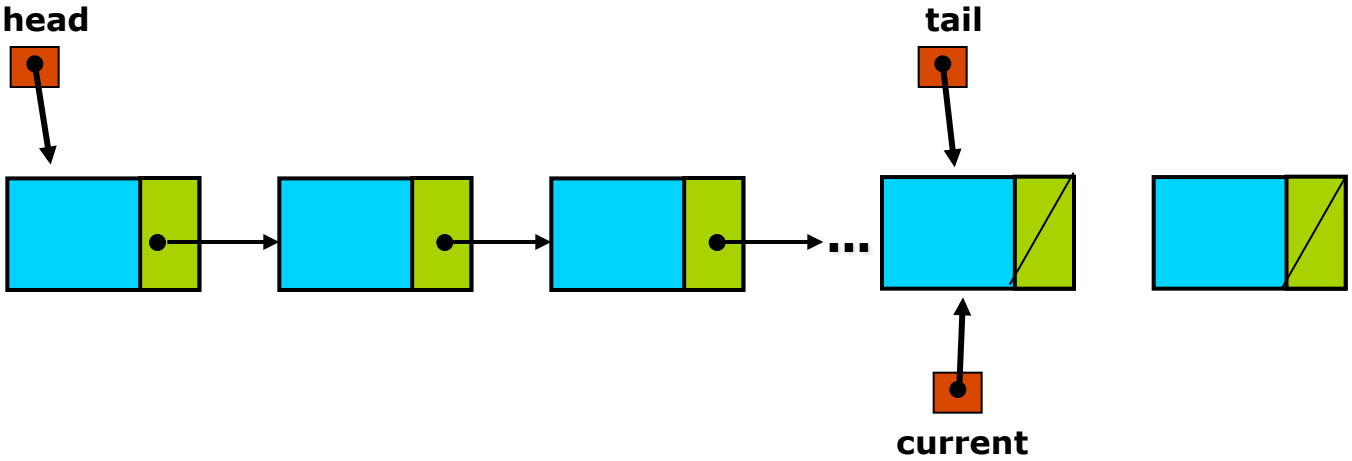
# Remoção em lista simplesmente encadeada

**- Da cauda da lista:**

a




b



# Remoção em lista simplesmente encadeada

**- Da cauda da lista:**

```
public E removeLast() {



}
```

# Inserção em uma posição qualquer

```
public void insert(E element, int pos) {




}
```

# Remoção em uma posição qualquer

```
public E remove(int pos) {




}
```

# Exercício

- Complete a classe SinglyLinkedList, de acordo com o que foi visto em aula.
- Além dos métodos da interface, crie também o método toString(), que retorna uma String com os elementos da lista dispostos lado a lado, separados por um hífen.

```
public String toString() {


}
```

- Crie também um método chamado search, que recebe um elemento por parâmetro e retorna a posição da lista em que este elemento se encontra, considerando que o primeiro elemento está na posição zero

```
public int search(E element) {


}
```

# Testando a Lista Simplesmente Encadeada

```
public static void main(String[] args) {
        SinglyLinkedList<Integer> lista = new SinglyLinkedList<Integer>();
        try{
                lista.insertFirst(1);
                lista.insertFirst(2);
                lista.insertFirst(3);
                lista.insertFirst(4);
                lista.insertFirst(5);
                lista.insertFirst(6);
                lista.insertLast(7);

                System.out.println(lista);

                while (!lista.isEmpty()) {
                        System.out.println(lista.removeFirst());
                }

         }
                catch (UnderflowException e) {
                        System.out.println(e);
        }
}
```

# Exercícios extras

Como inserir um elemento no meio de uma lista simplesmente encadeada?

a) Crie um método chamado addAfter(Element e, int pos), que insere o nodo n depois do nodo de número pos (considerando que o primeiro nodo é o nodo na posição 0).

b) Crie um método chamado addBefore(Element e, int pos), que insere o nodo n antes do nodo de número pos (considerando que o primeiro nodo é o nodo na posição 0).