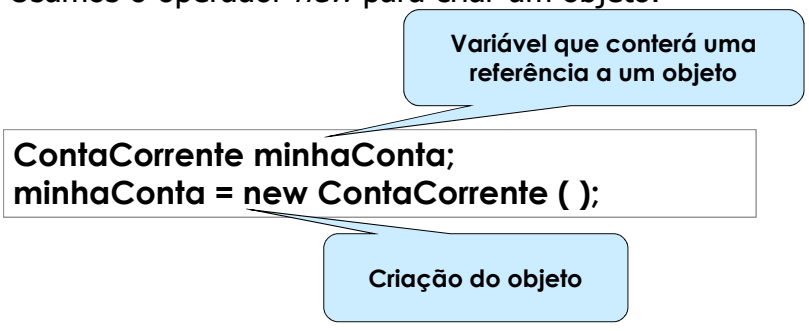
Listas
Simplesmente Encadeadas

Professores de Programação II
Gustavo Lermen
Mateus Raeder
Patricia Jaques

UNISINOS
UNIVERSIDADE DO VALE DO RIO DOS SINOS

Criando um objeto

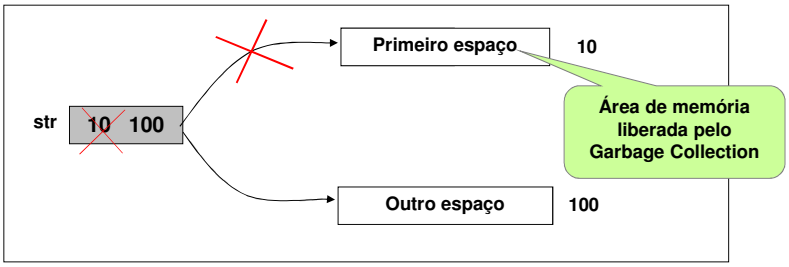
- Objeto é uma instância de uma classe;
- Usamos o operador *new* para criar um objeto.



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

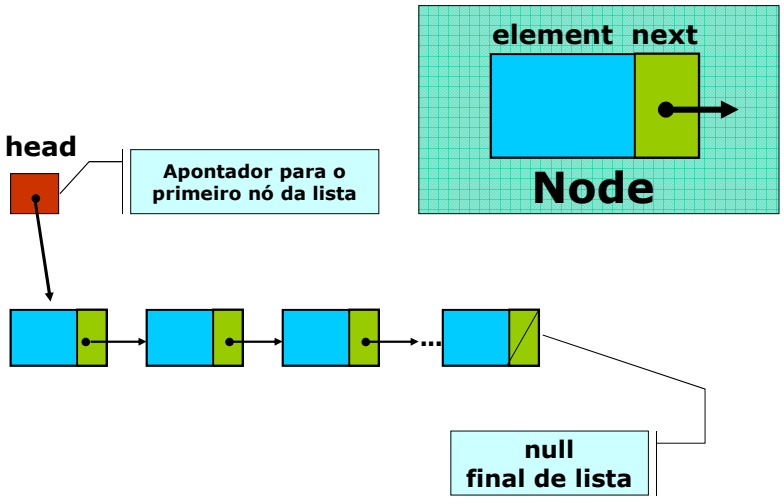
- O tipo de armazenamento de uma lista linear pode ser classificado de acordo com a posição relativa na memória de dois nós consecutivos na lista (sempre contígua ou não).
- Lista linear com alocação estática de memória
  - Também chamadas de **Listas Sequenciais**
  - Nós em posições contíguas de memória
  - Geralmente representado por arrays
  - Útil para implementar filas e pilhas (variáveis para controlar fim e início)
- Lista linear com alocação dinâmica
  - Também chamadas de **Listas Encadeadas**
  - 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
  - último elemento contém a referência *null* (para indicar que não referencia nenhum outro).

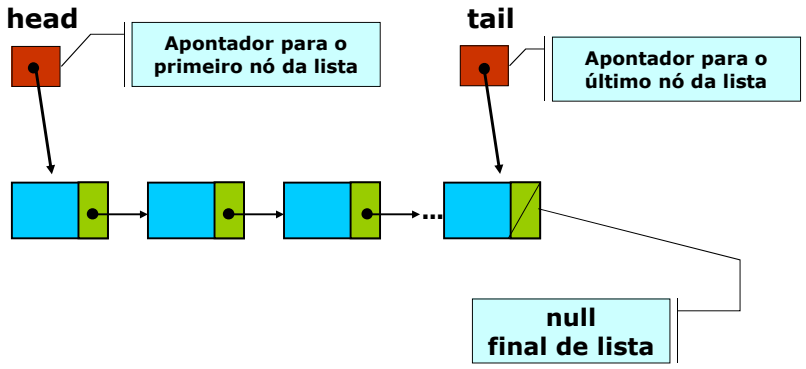
Ineficiente: se queremos inserir um elemento no final da lista, temos que localizar o último elemento: para tanto é necessário percorrer todos os elementos da lista.

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 Genérica

```
public class Node<E> {
  // Instance variables:
  private E element;
  private Node<E> next;
  /** Creates a node with null references to its element and next node. */
  public Node() {
    this(null, null);
  }
  /** Creates a node with the given element and next node. */
  public Node(E e) {
                this(e, null);
  }
  /** Creates a node with the given element and next node. */
  public Node(E e, Node<E> n) {
    element = e;
    next = n;
  }
  // Accessor methods:
  public E getElement() {
    return element;
  }
  public Node<E> getNext() {
    return next;
  }
  // Modifier methods:
  public void setElement(E newElem) {
    element = newElem;
  }
  public void setNext(Node<E> newNext) {
    next = newNext;
  }
}
```

Operações sobre lista

- **public boolean isEmpty()**
  - verifica se a lista está vazia
- **public E getFirst()**
  - Retorna o primeiro elemento da lista, sem removê-lo
- **public E getLast()**
  - Retorna o último elemento da lista, sem removê-lo
- **public void addFirst(E e)**
  - insere element na frente da lista
- **public void addLast(E e)**
  - insere element no final da lista
- **public E removeFirst()**
  - remove e retorna o primeiro elemento da lista
- **public E removeLast()**
  - remove e retorna o último elemento da lista
- **public void print()**
  - exibe o conteúdo da lista

Classe SLinkedList Genérica

```
/** Lista simplesmente encadeada **/
public class SLinkedList<E> {
        protected Node<E> head; //nodo cabeça da lista
        protected Node<E> tail; //nodo cauda da lista
        protected long size; //número de nodos da lista

        /** Construtor default que cria uma lista vazia **/
        public SLinkedList(){
                head = null;
                tail = null;
                size = 0;
        }
}
```

isEmpty(), getFirst() e getLast()

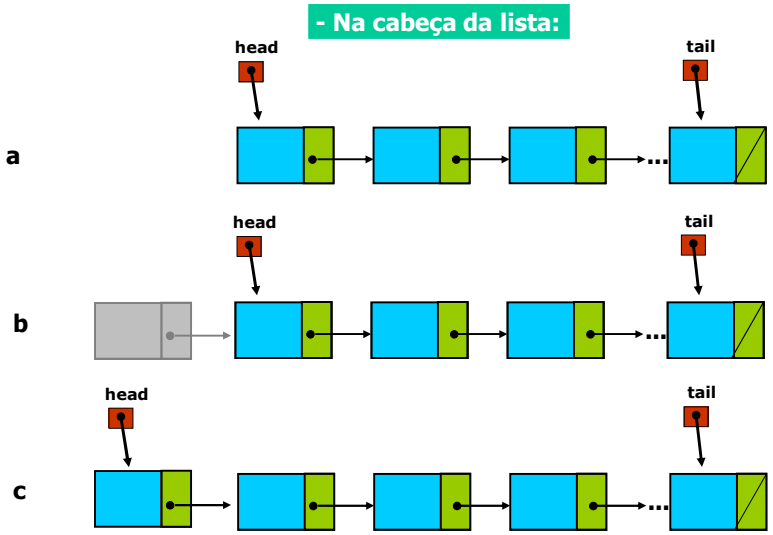
```
public boolean isEmpty(){
        return head == null;
}

public Node<E> getFirst() throws UnderflowException {
        if (isEmpty()) throw new UnderflowException();
        return head;
}

public Node<E> getLast() throws UnderflowException {
        if (isEmpty()) throw new UnderflowException();
        return tail;
}
```


Inserção em lista simplesmente encadeada

- Na cabeça da lista:



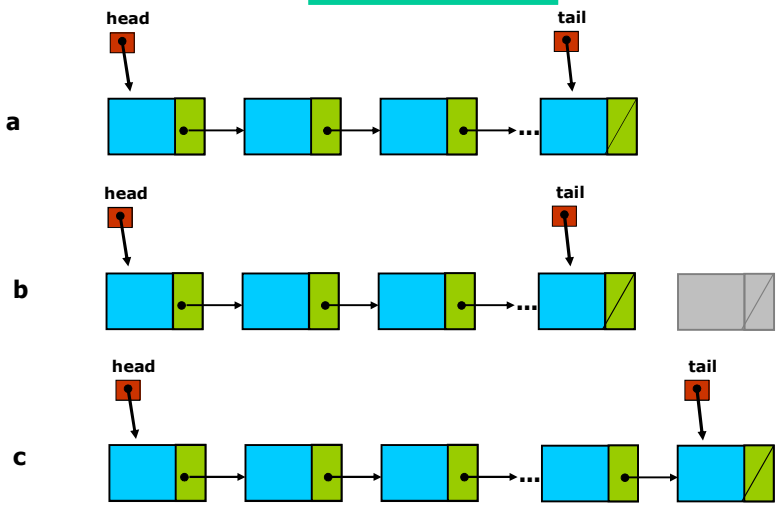
Inserção em lista simplesmente encadeada

- Na cabeça da lista:

```
public void addFirst(Node<E> novoNodo){
        novoNodo.setNext(head);
        head = novoNodo;
        size++;
        if(size == 1)
                tail = head;
}
```


Inserção em lista simplesmente encadeada

- Na cauda da lista:



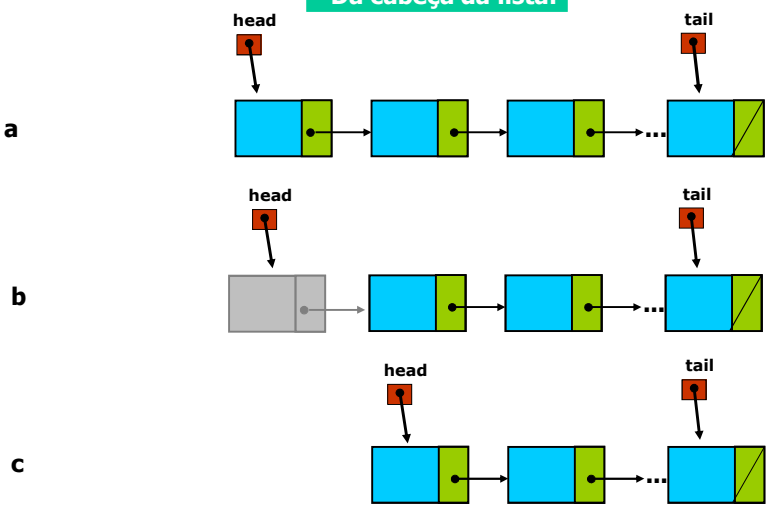
Inserção em lista simplesmente encadeada

- Na cauda da lista:

```
public void addLast(Node<E> novoNodo){
        if(isEmpty())
                addFirst(novoNodo);
        else{
                novoNodo.setNext(null);
                tail.setNext(novoNodo);
                tail = novoNodo;
                size++;
        }
}
```


Remoção em lista simplesmente encadeada

- Da cabeça da lista:



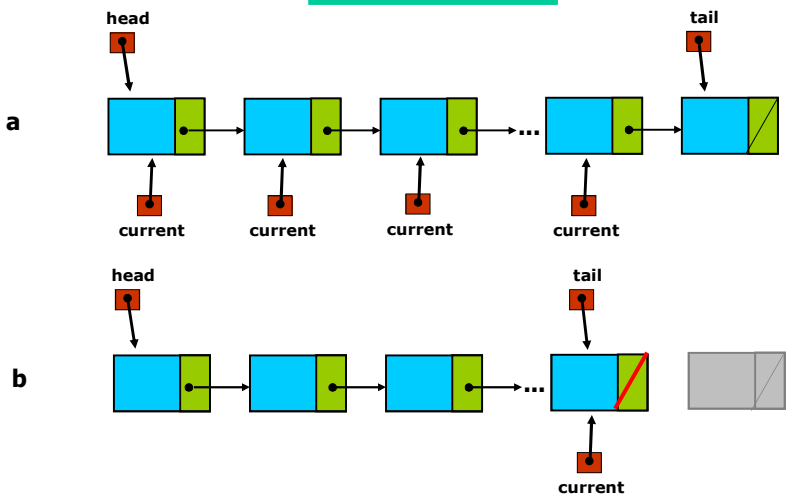
Remoção em lista simplesmente encadeada

- Da cabeça da lista:

```
public Node<E> removeFirst() throws UnderflowException {
        if(isEmpty()) throw new UnderflowException();
        Node<E> removedItem = head;
        if (head == tail) {
                head = tail = null;
        }
        else {
                head = head.getNext();
        }
        size--;
        return removedItem;
}
```


Remoção em lista simplesmente encadeada

- Da cauda da lista:



Remoção em lista simplesmente encadeada

- Da cauda da lista:

```
public Node<E> removeLast() throws UnderflowException {
        if (isEmpty()) throw new UnderflowException();
        Node<E> removedItem = tail;
        if (head == tail) {
                head = tail = null;
        }
        else {
                Node<E> current = head;
                while (current.getNext() != tail) {
                        current = current.getNext();
                }
                tail = current;
                current.setNext(null);
        }
        size--;
        return removedItem;
}
```


Classe UnderflowException

```
public class UnderflowException extends Exception {
        public String toString() {
                return "UNDERFLOW!";
        }
}
```

Exercício

Como imprimir a lista simplesmente encadeada?

a) Crie um método print(), que percorre a lista e imprime os elementos.

Exercício - Resposta

```
public void print() {
        if (isEmpty()) {
                System.out.println("Empty list");
        }
        else {
                System.out.print("The list is: ");
                Node<E> current = head;
                while (current != null) {
                        System.out.print(current.getElement().toString() + " ");
                        current = current.getNext();
                }
                System.out.println("\n");
        }
}
```

Testando a Lista Simplesmente Encadeada

```
public static void main(String args[]){
        SLinkedList<String> lista = new SLinkedList<String>();
        try{
                lista.addFirst(new Node<String>("A"));
                lista.addFirst(new Node<String>("B"));
                lista.addFirst(new Node<String>("C"));
                lista.addLast(new Node<String>("D"));
                lista.addLast(new Node<String>("E"));
                lista.addLast(new Node<String>("F"));
                lista.removeFirst();
                lista.removeLast();

        }catch(UnderflowException e){
                System.out.println("ERRO: lista vazia, impossìvel remover!");
                e.printStackTrace();
        }
        lista.print();
    }
```


Exercício

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

a) Crie um método chamado addAfter(Node<E> n, 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(Node<E> n, int pos), que insere o nodo n antes do nodo de número pos (considerando que o primeiro nodo é o nodo na posição 0).