Conteúdo

1. Introdução
2. Listas
3. Pilhas e Filas
4. Árvores
5. Árvores de Pesquisa
   - Árvore Binária e Árvore AVL
   - Árvore N-ária e Árvore B
6. Tabelas de Dispersão (Hashing)
7. Métodos de Acesso a Arquivos
8. Métodos de Ordenação de Dados

Listas Ordenadas

Lista Ordenada

LISTA ORDENADA: lista linear na qual os elementos aparecem em ordem, ou seja, do menor para o maior.

⇨ Inserção de um elemento em uma lista ordenada:

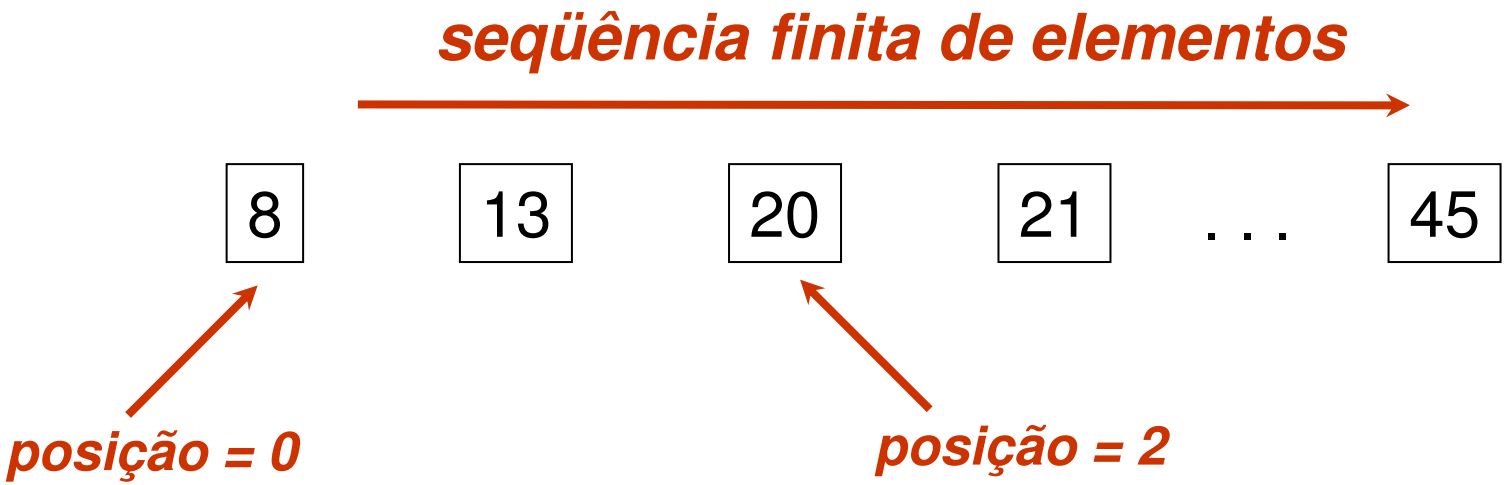
- pré-condição: lista já está ordenada.
- pós-condição: lista continua ordenada.

Exemplos de Listas Ordenadas

- Itens de um índice remissivo (ordenados pelo nome do item)
- Nomes dos alunos em uma lista de chamada (ordenados pelo nome do aluno)
- Nomes dos aprovados em um concurso público (ordenados pelo resultado da prova)

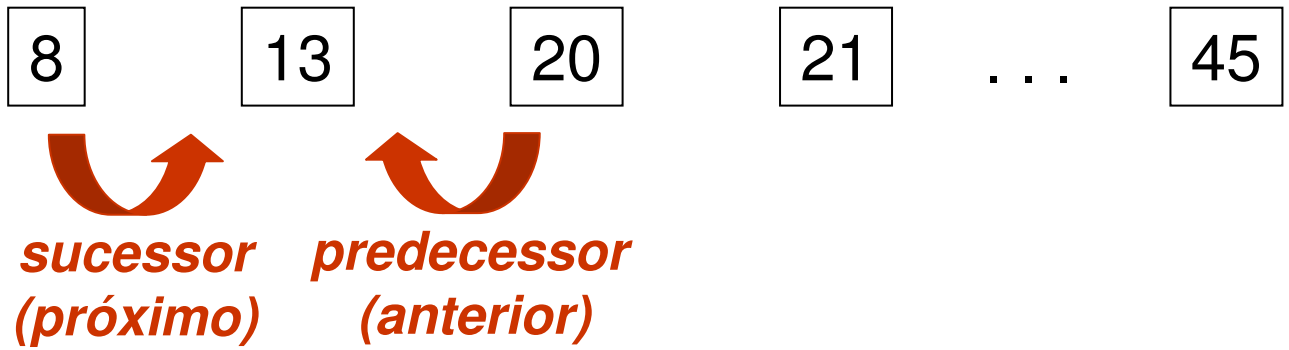
Características de uma Lista Ordenada

- Seqüência Finita de Elementos



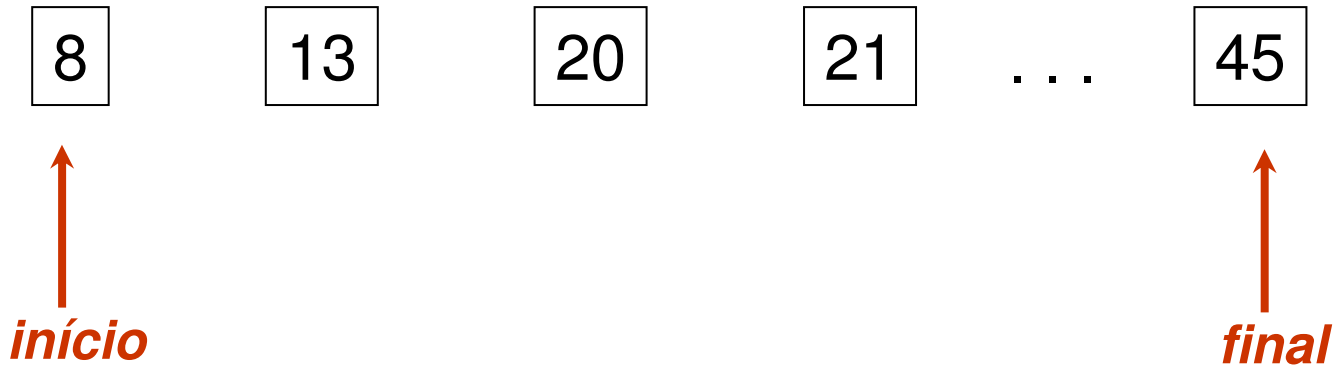
Características de uma Lista Ordenada

- Seqüência: próximo e anterior



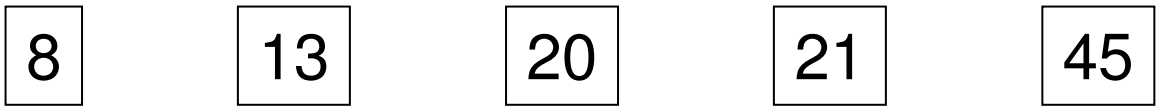
Características de uma Lista Ordenada

- Seqüência: início e fim



Características de uma Lista Ordenada

- Comprimento



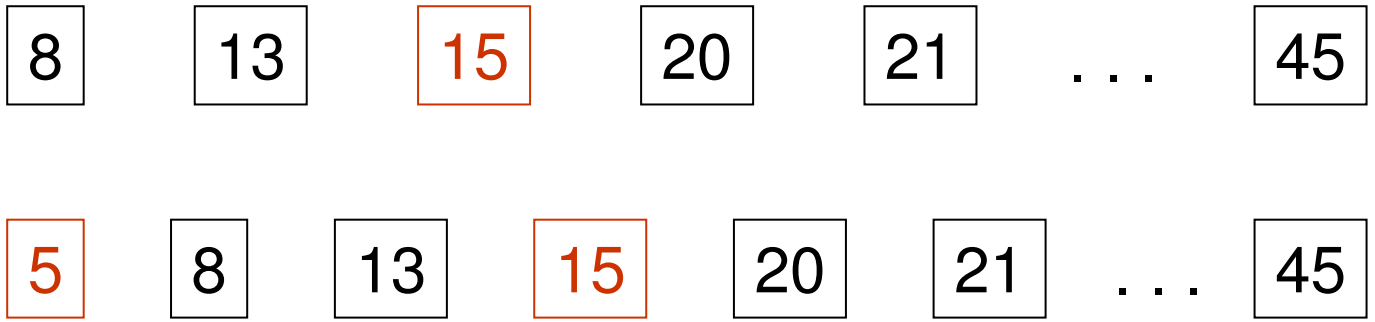
Comprimento = 5

- Lista Vazia

Comprimento = 0

Operações sobre uma Lista Ordenada

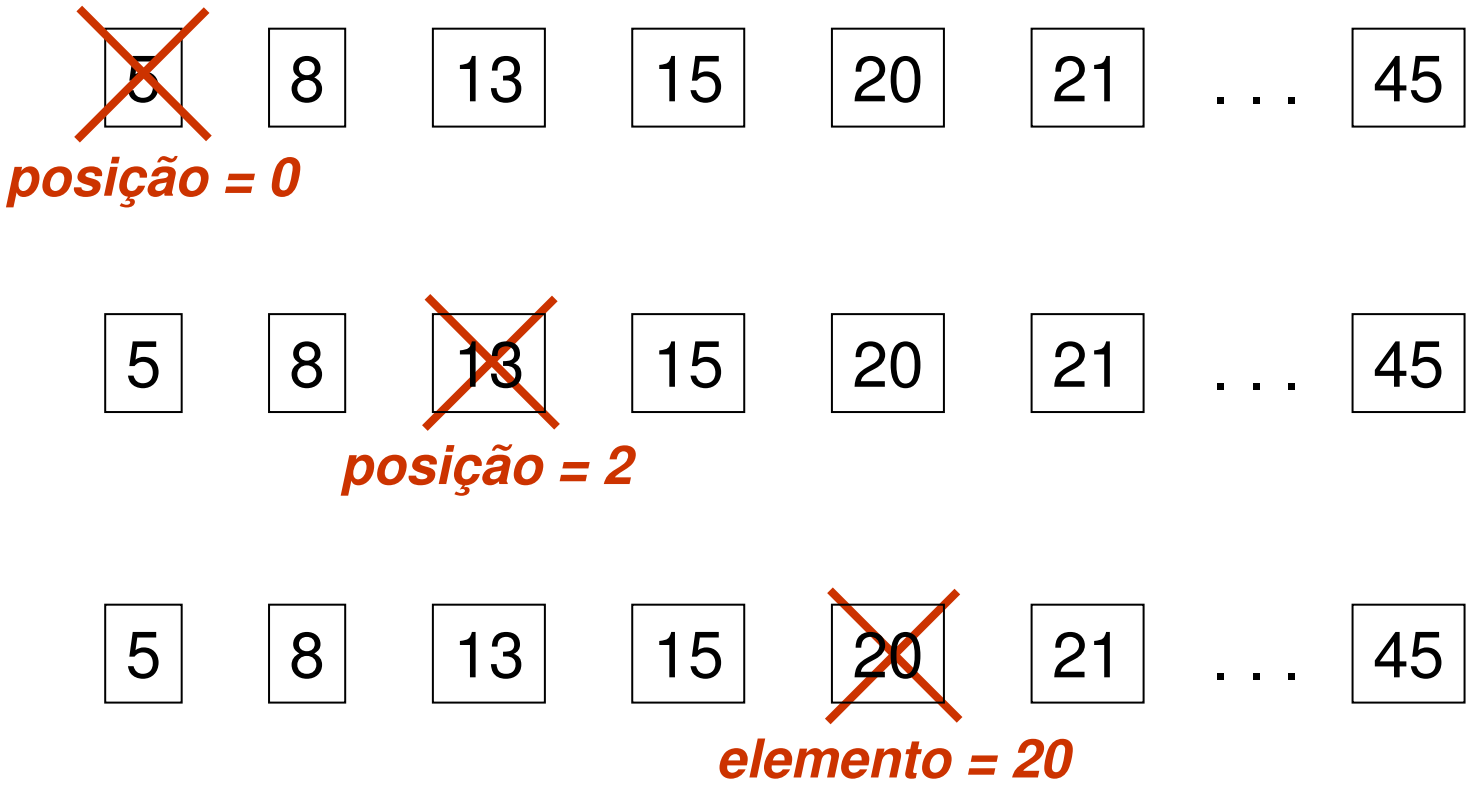
→ **Inserção de elemento**



⇨ **não é possível especificar a posição de inserção**

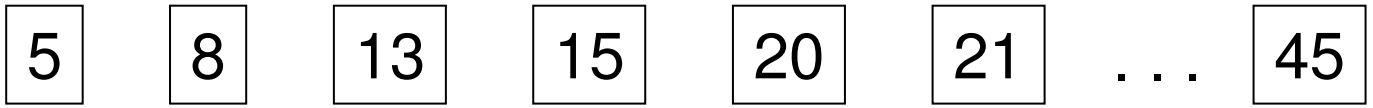
Operações sobre uma Lista Ordenada

→ **Remoção de elemento**



Operações sobre uma Lista Ordenada

→ **Consulta**



- Elemento da posição 1 → 8
- Verificar se o elemento 20 está na lista → true

Lista Ordenada

Uma lista ordenada deve armazenar somente objetos que possam ser comparáveis.

⇨ É necessário que o objeto tenha algum método para comparar se ele é menor, igual ou maior que outro objeto.

Interface Comparable

⇨ Usamos a interface Comparable<T> do Java.

- Todo elemento armazenado na lista ordenada deve implementar esta interface, ou seja, o método compareTo.
- O método compareTo deve retornar
  - inteiro negativo: se o elemento (this) é menor do que o elemento passado como parâmetro;
  - zero: se o elemento (this) é igual ao elemento passado como parâmetro;
  - inteiro positivo: se o elemento (this) é maior do que o elemento passado como parâmetro.

Interface da Lista Ordenada

Quais métodos da interface Lista devem ser mantidos na interface ListaOrdenada?

Interface da Lista Ordenada

```
public interface Lista <E>{

public void insere (E elemento, int posicao);
public void insere (E elemento);
public void insereTodos (Lista<E> l, int posicao);
public boolean contem (E elemento);
public E remove (int posicao);
public int remove (E elemento);
public void removeTodos (Lista<E> l);
public E retorna (int posicao);
public int retornaPosicao (E elemento);
...
```

Interface da Lista Ordenada

```
...
public E substitui (int posicao, E elemento);
public boolean substitui (E elemento1, E elemento2);
public void trocaPosicao (int posicao1, int posicao2);
public Lista<E> retornaSubLista (int inicio, int fim);
public Iterator<E> retornaIterator();
}
```

Interface da Lista Ordenada

```
public interface Lista <E>{

public void insere (E elemento, int posicao);
public void insere (E elemento);
public void insereTodos (Lista<E> l, int posicao);
public boolean contem (E elemento);
public E remove (int posicao);
public int remove (E elemento);
public void removeTodos (Lista<E> l);
public E retorna (int posicao);
public int retornaPosicao (E elemento);
...
```

public void insere (E elemento);

Interface da Lista Ordenada

```
...
public E substitui (int posicao, E elemento);
public boolean substitui (E elemento1, E elemento2);
public void trocaPosicao (int posicao1, int posicao2);
public Lista<E> retornaSubLista (int inicio, int fim);
public Iterator<E> retornaIterator();
}
```

Interface da Lista Ordenada

```
public interface ListaOrdenada <C extends Comparable<C>> {
public void insere (C elemento);
public void insereTodos (ListaOrdenada<C> l);
public C remove (int posicao) throws ExcecaoPosicaoInvalida;
public int remove (C elemento);
public void removeTodos (ListaOrdenada<C> l);
public boolean contem (C elemento);
public C retorna (int posicao) throws ExcecaoPosicaoInvalida;
public int retornaPosicao (C elemento);
public ListaOrdenada<C> retornaSubLista (int inicio, int fim)
                                        throws ExcecaoPosicaoInvalida;
public Iterator<C> retornaIterator();     }
```

Alternativas de Implementação

- Lista Ordenada como Array
- Lista Ordenada como Lista Encadeada

Lista Ordenada como Array

⇨ Os elementos, ordenados, ficam justapostos na memória através da utilização de um vetor unidimensional.

0	1	2	3	4	5	6	7	8	9	...	N
e_1	e_2	e_3	e_4	e_5	e_6	e_7	...	e_m			

Lista Ordenada como Array

Exemplo:

0	1	2	3	4	5	6	7	8	9	...	N
8	13	20	21	30	32	45					

Lista Ordenada como Array

⇨ Classe **ListaOrdenadaArray** implementa ListaOrdenada

Atributos

- **elementos** (array com objetos Comparable ou subclasse)
- **numeroElementos**

Métodos

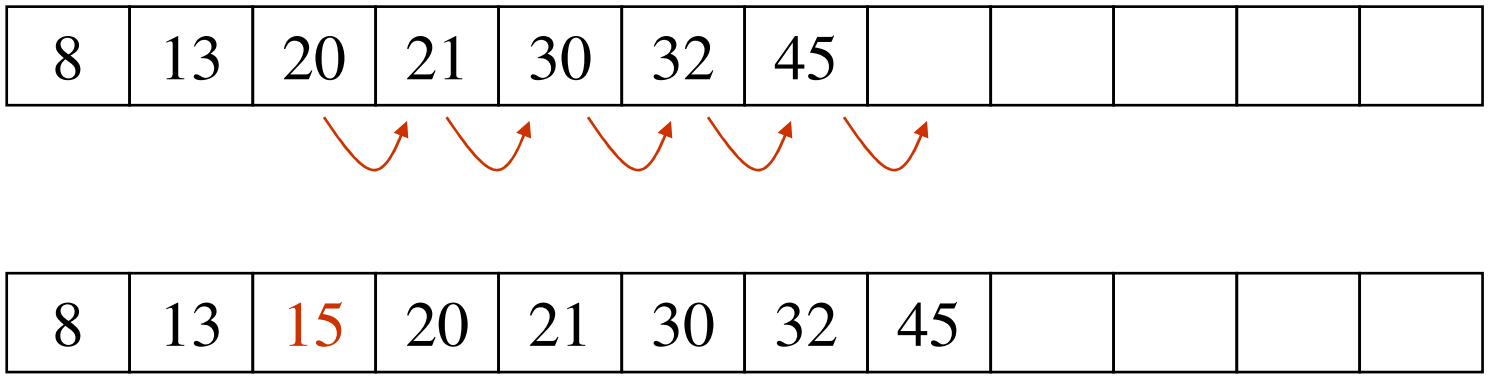
- construtor ()

  \+

- métodos especificados na interface ListaOrdenada

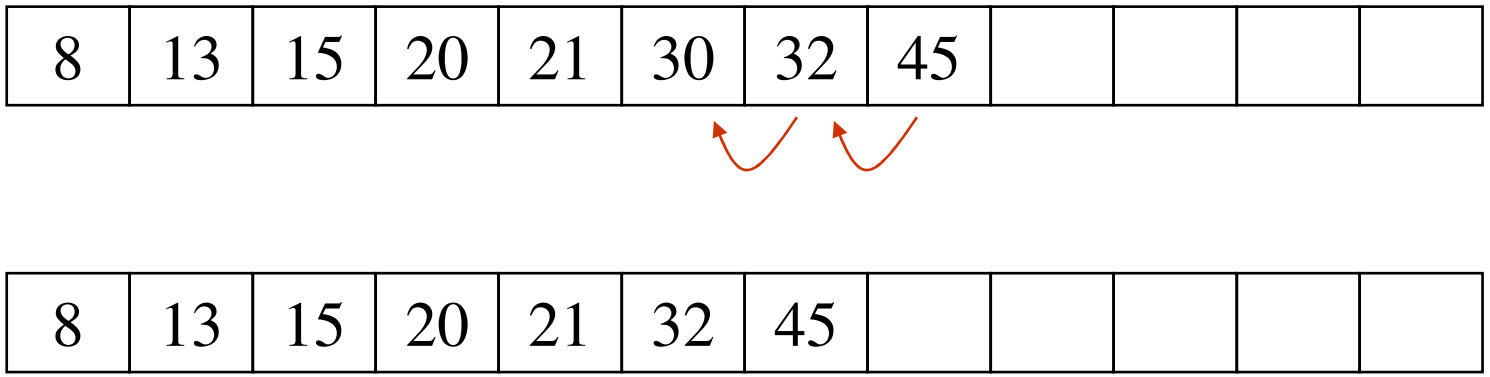
Operações sobre a Lista Ordenada

Exemplo: inserção do elemento 15



Operações sobre a Lista Ordenada

Exemplo: exclusão do elemento 30



Implementação

```
public class ListaOrdenadaArray<C extends Comparable<C>>
                                        implements ListaOrdenada<C>{
    private C[] elementos;
    private int numElementos;

    public ListaOrdenadaArray (int tamanho){
            this.elementos = (C[]) new Comparable[tamanho];
    }

    public ListaOrdenadaArray (){
            this.elementos = (C[]) new Comparable[100];
    }
```

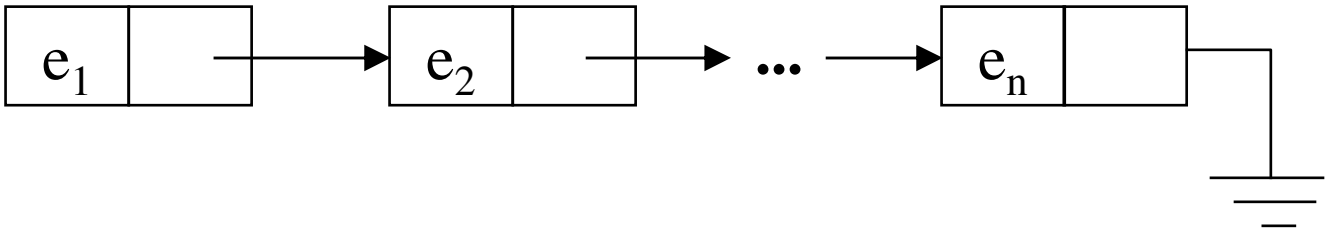
Implementação

```
public void insere (C elemento) {
if (this.numElementos==this.elementos.length){
        C[] novoArray = (C[]) new Comparable[this.elementos.length*2];
        System.arraycopy(this.elementos,0,novoArray,0,this.elementos.length);
        this.elementos = novoArray;
}
int cont = this.numElementos-1;
while (cont >= 0 && elemento.compareTo(this.elementos[cont]) < 0){
        this.elementos[cont+1] = this.elementos[cont];
        cont--;
}
this.elementos[cont+1] = elemento;
this.numElementos++;
}
```

Complexidade: ?

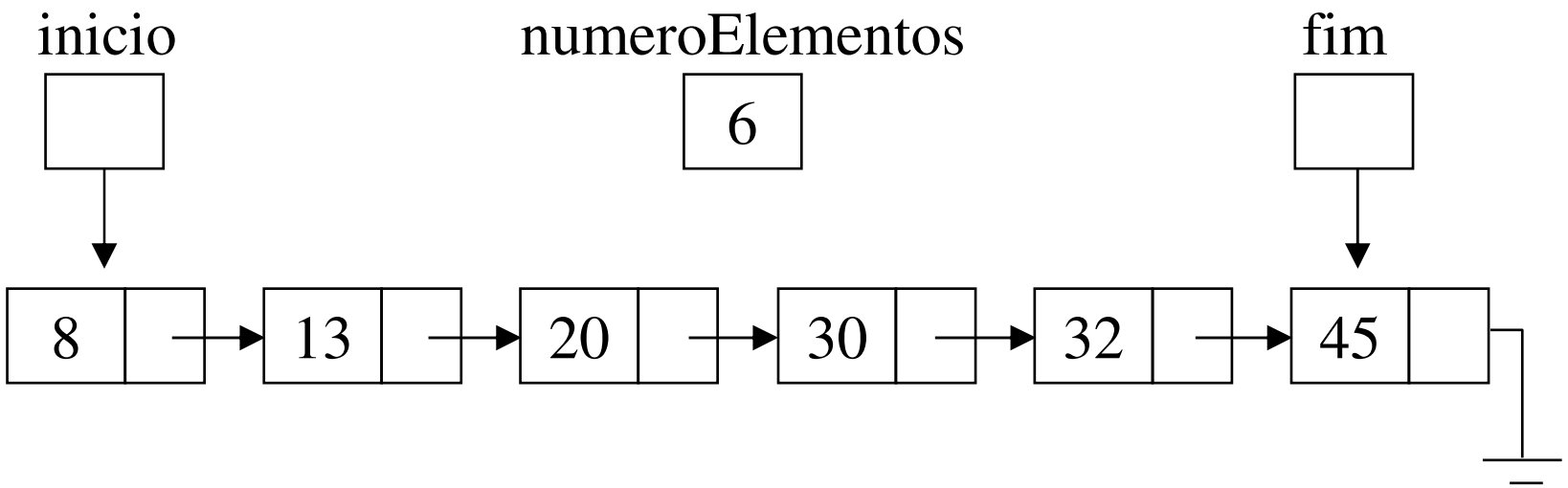
Lista Ordenada Encadeada

⇨ Os elementos, ordenados, estão associados entre si através de elos.



Lista Ordenada Encadeada

Exemplo



Lista Ordenada Encadeada

⇨ Classe **ListaOrdenadaEncadeada** implementa ListaOrdenada

Atributos

- **inicio** (referência a um objeto da classe NodoComparable)
- **fim** (referência a um objeto da classe NodoComparable)
- **numeroElementos**

Métodos

- construtor ()

  +

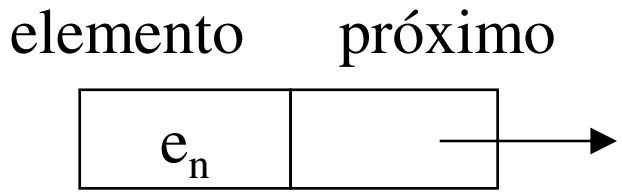
- métodos especificados na interface ListaOrdenada

Lista Ordenada Encadeada

Classe **NodoComparable**

Atributos:

- **elemento** (Comparable ou subclasse)
- **proximo** (referência a um outro objeto da classe NodoComparable)



Lista Ordenada Encadeada

➡ Classe **NodoComparable**

Atributos:

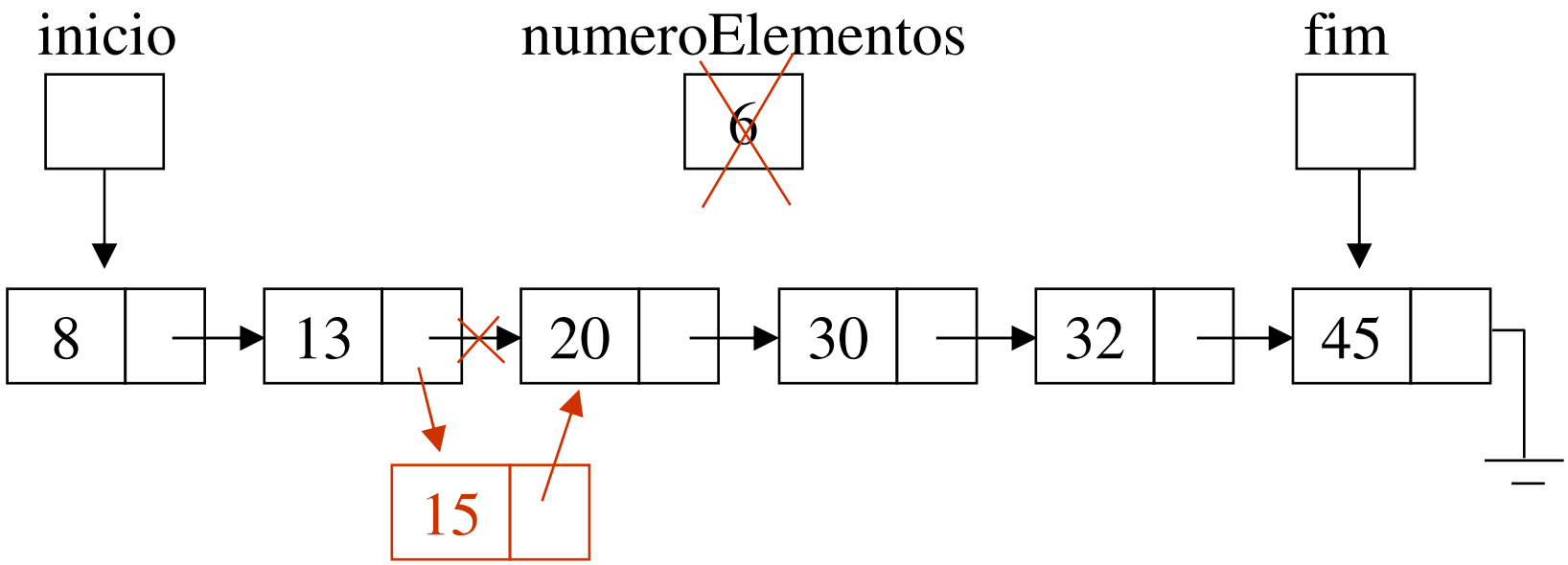
- **elemento** (Comparable ou subclasse)
- **proximo** (referência a objeto da classe NodoComparable)

Métodos:

- construtor (C)
- atribuiProximo (NodoComparable)
- atribuiElemento (C)
- retornaProximo : NodoComparable
- retornaElemento : C

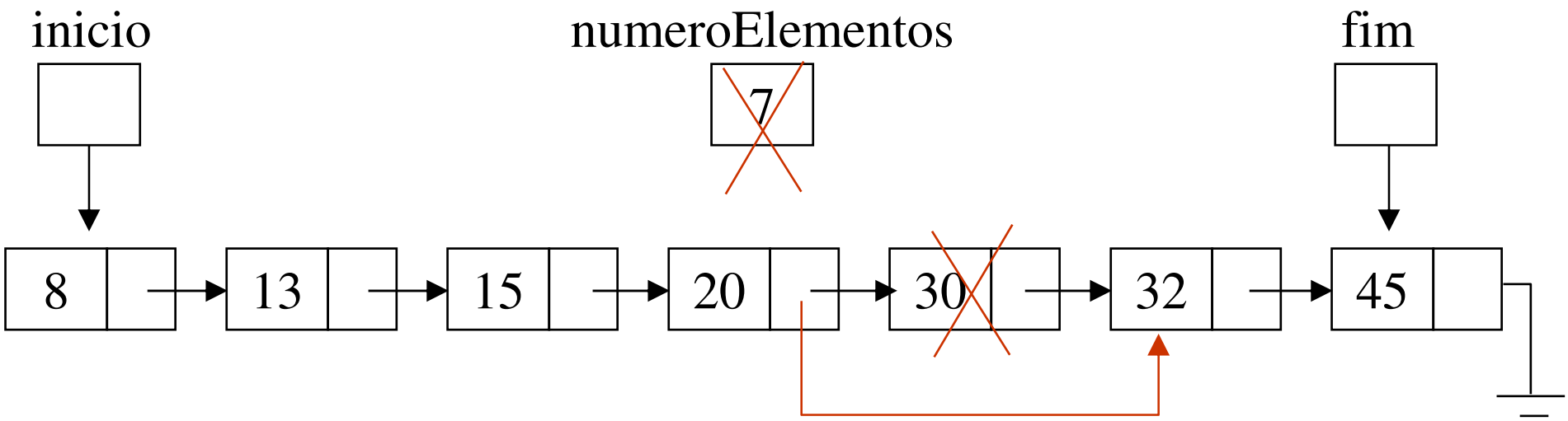
Lista Ordenada Encadeada

Exemplo: inserção do elemento 15



Lista Ordenada Encadeada

Exemplo: exclusão do elemento 30



Implementação

```
public class ListaOrdenadaEncadeada<C extends Comparable<C>>
                                        implements ListaOrdenada<C> {
private NodoComparable<C> inicio;
private NodoComparable<C> fim;
private int numElementos;

public ListaOrdenadaEncadeada() {
        this.inicio = null;
        this.fim = null;
        this.numElementos = 0;
}
```

Implementação

```
public void insere (C elemento) {
NodoComparable<C> nodo = new NodoComparable<C> (elemento);
NodoComparable<C> aponta = this.inicio;
NodoComparable<C> anterior = null;
while (aponta != null && aponta.retornaElemento().compareTo(elemento)<0){
        anterior = aponta;
        aponta = aponta.retornaProximo();
}
…
```

Implementação

```
if (aponta == this.inicio){
        nodo.atribuiProximo(this.inicio);
        this.inicio = nodo;
        if (this.fim == null)
                this.fim = nodo;
}
else
        if (aponta == null){    //inserir no final da lista
                this.fim.atribuiProximo(nodo);
                this.fim = nodo;
        }
        else{                   //inserir no meio da lista
                anterior.atribuiProximo(nodo);
                nodo.atribuiProximo(aponta);
        }
this.numElementos++;            }
```

Complexidade: ?

Lista Ordenada

Exercício:

Faça um teste da lista ordenada array e encadeada utilizando uma classe Pessoa que implementa a interface Comparable<T>. A classe Pessoa deve ter os atributos nome e idade. Considere como ordem a idade, mas se a idade for igual, a ordem é o nome.