# Programação Orientada a Objetos

Prof. Paulo Henrique Pisani

Prof. Saul de Castro Leite

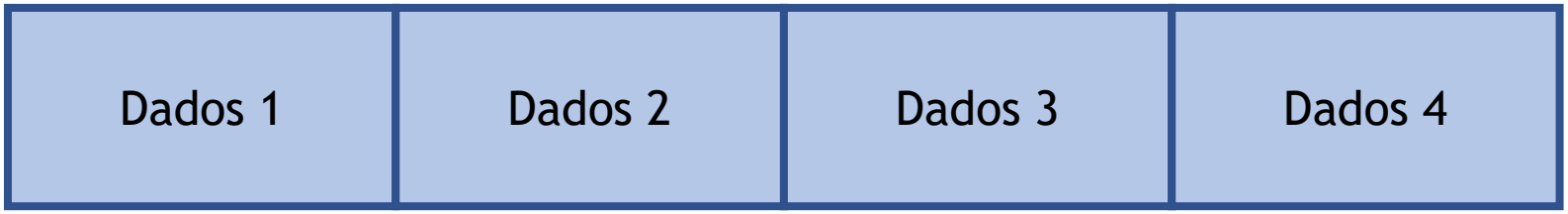
http://professor.ufabc.edu.br/~paulo.pisani/

UFABC

junho/2018

# Listas com arranjos

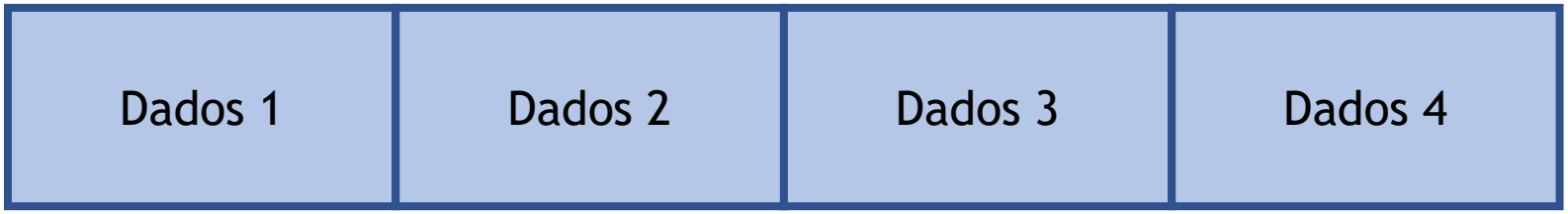
- Itens dispostos em um arranjo sequencial;




UFABC

# Listas com arranjos

- Itens dispostos em um arranjo sequencial;

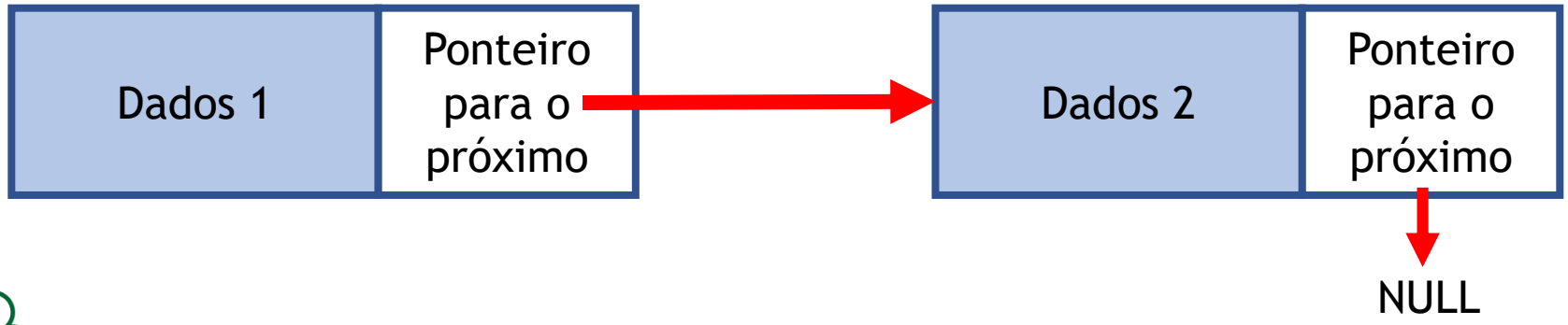



Problemas?

UFABC

# Listas ligadas/encadeadas

- Estrutura de dados que armazena os itens de forma não consecutiva na memória:
  - Usa ponteiros para “ligar” um item no próximo.




UFABC

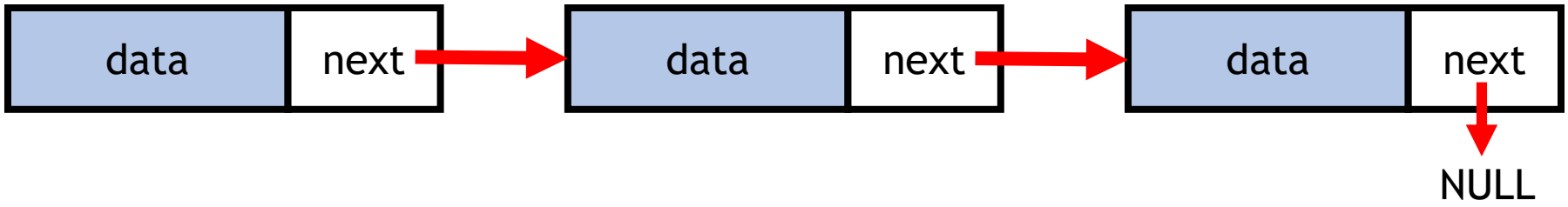
# Listas ligadas/encadeadas

- Vários tipos:
  - Listas simplesmente ligadas (com e sem nó cabeça);
  - Listas duplamente ligadas (com e sem nó cabeça);
  - Listas circulares.

UFABC

# Listas simplesmente ligadas

- Cada item é ligado somente ao próximo item;

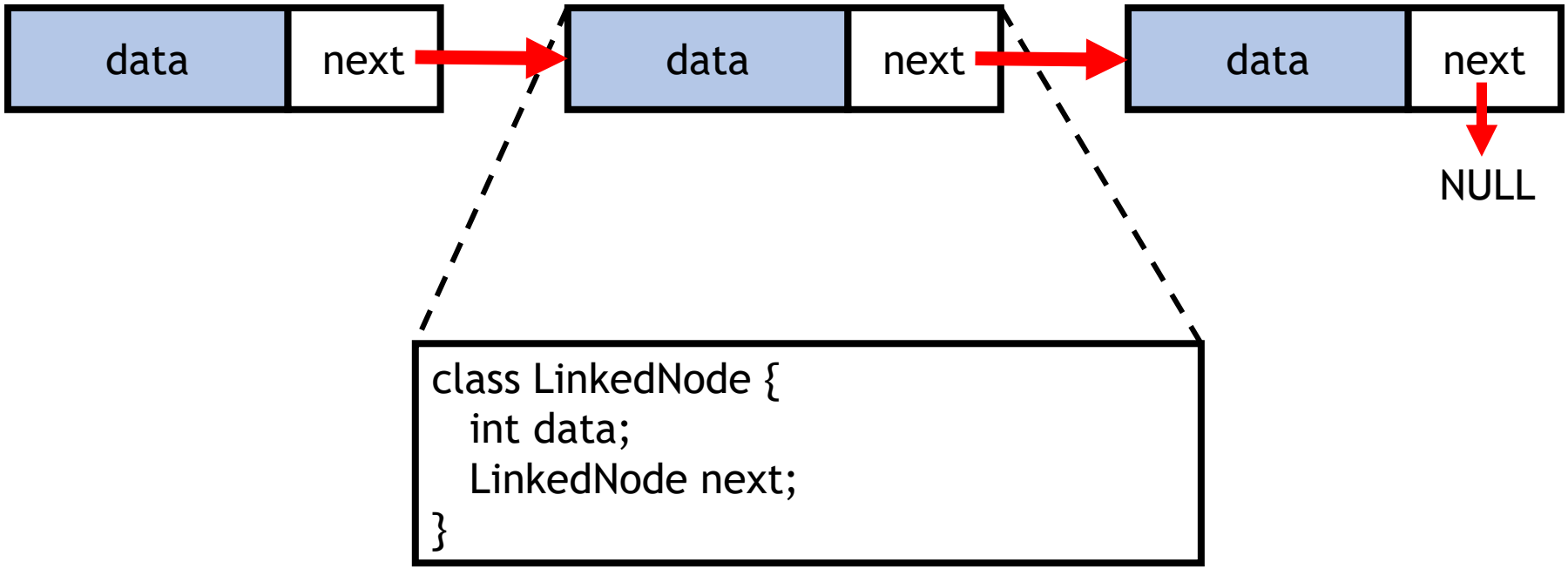



## Como implementar no Java?

UFABC

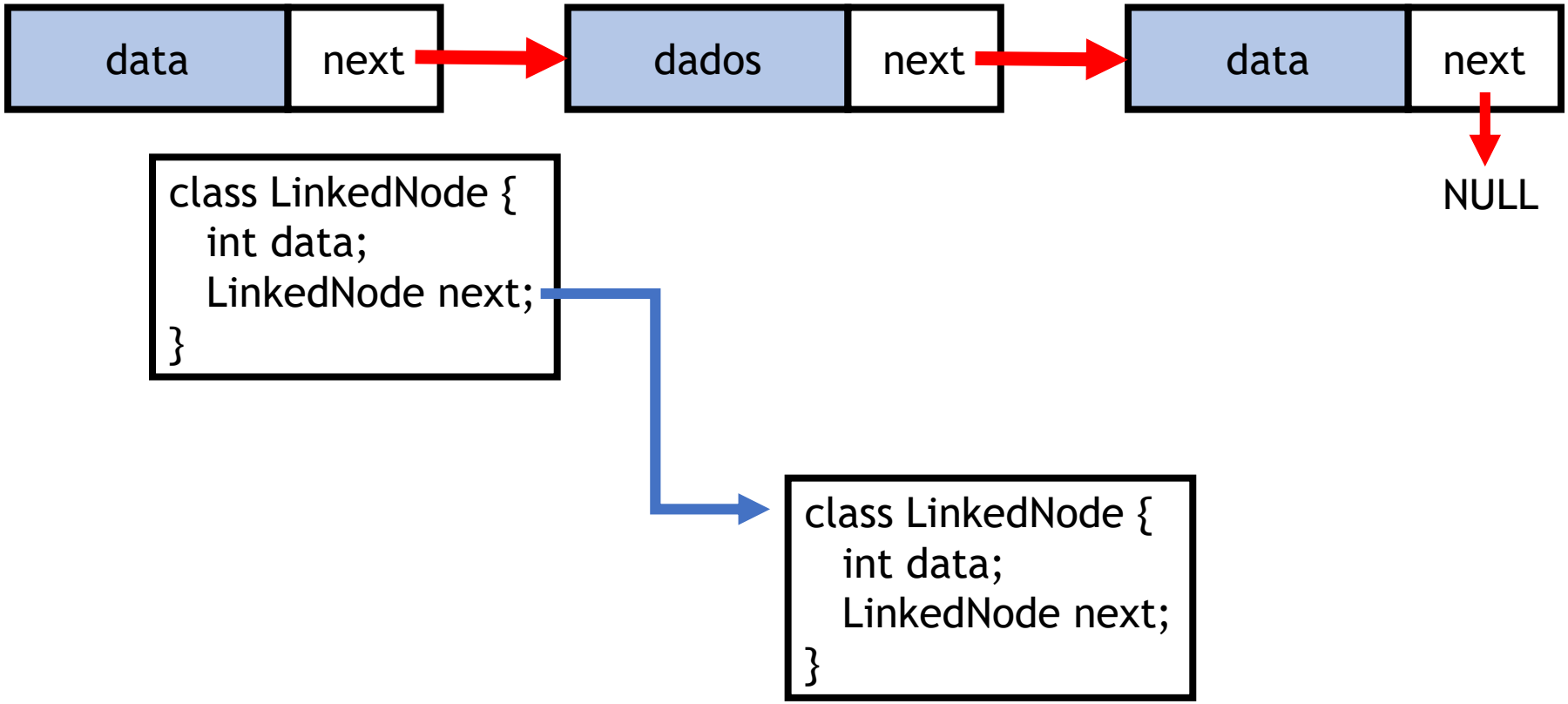
# Listas simplesmente ligadas

- Cada item é ligado somente ao próximo item;




UFABC

# Listas simplesmente ligadas




UFABC

# Listas simplesmente ligadas

- Já criamos uma classe para representar os nós da lista ligada: **LinkedNode**;
- Agora vamos criar uma classe para gerenciar a lista ligada: **ListaLigada**.

UFABC

# Listas simplesmente ligadas

```
public class ListaLigada {
    private LinkedNode first = null;

    public void adicionaItemNoFinal(int item) {
      ...
    }

    public void removeItem(int item) {
      ...
    }

    public void imprimeListaLigada() {
      ...
    }
}
```

UFABC

# Listas simplesmente ligadas

```
public void adicionaItemNoFinal(int item) {
    LinkedNode novoItem = new LinkedNode();
    novoItem.valor = item;
    novoItem.next = null;

    if (this.first == null) {
        this.first = novoItem;
    } else {
        LinkedNode anterior = null;
        LinkedNode atual = first;
        while (atual != null) {
            anterior = atual;
            atual = atual.next;
        }
        anterior.next = novoItem;
    }
}
```

UFABC

# Listas simplesmente ligadas

```
public void removeItem(int item) {
    LinkedNode anterior = null;
    LinkedNode atual = first;
    while (atual != null && atual.valor != item) {
        anterior = atual;
        atual = atual.next;
    }
    if (atual != null) { // Achou o item
        if (anterior == null)
            first = atual.next;
        else
            anterior.next = atual.next;
    }
}
```

UFABC

# Listas simplesmente ligadas

```
public void imprimeListaLigada() {
    LinkedNode atual = first;
    while (atual != null) {
        System.out.print(atual.valor + " ");
        atual = atual.next;
    }
    System.out.print("\n");
}
```

UFABC

# Usando a lista ligada...

```
public class TesteObjetos {

    public static void main(String[] args) {
        ListaLigada listaL = new ListaLigada();
        listaL.adicionaItemNoFinal(5);
        listaL.adicionaItemNoFinal(35);
        listaL.adicionaItemNoFinal(20);
        listaL.adicionaItemNoFinal(70);
        listaL.adicionaItemNoFinal(2);
        listaL.imprimeListaLigada();
        listaL.removeItem(20);
        listaL.imprimeListaLigada();
    }

}
```

UFABC

# Modificadores de acesso...

- Podemos deixar a classe LinkedNode com acesso mais restrito, já que ela é usada apenas por ListaLigada:
  - Quem usa a classe ListaLigada não tem acesso aos nós diretamente;

UFABC

# Modificadores de acesso em classes...

**Classe ListaLigada é pública** (o nome do arquivo segue o nome desta classe)

**Arquivo: ListaLigada.java**

```
public class ListaLigada {
    private LinkedNode first;

    public void adicionaItemNoFinal(int item) {
      ...
    }

    public void removeItem(int item) {
      ...
    }

    public void imprimeListaLigada() {
      ...
    }
}
```

**Classe LinkedNode é pública**

**Arquivo: LinkedNode.java**

```
public class LinkedNode {
   int valor;
   LinkedNode next;
}
```

UFABC

# Modificadores de acesso em classes...

**Classe ListaLigada é pública** (o nome do arquivo segue o nome desta classe)

**Arquivo: ListaLigada.java**

```
public class ListaLigada {
    private LinkedNode first;

    public void adicionaItemNoFinal(int item) {
      ...
    }

    public void removeItem(int item) {
      ...
    }

    public void imprimeListaLigada() {
      ...
    }
}
```

**Classe LinkedNode tem acesso package (sem modificadores de acesso)**

**Arquivo: LinkedNode.java**

```
class LinkedNode {
   int valor;
   LinkedNode next;
}
```

UFABC

# Modificadores de acesso em classes...

**Classe LinkedNode tem acesso package (sem modificadores de acesso)**
**>>> Mas agora está no mesmo arquivo: ListaLigada.java**

**Classe ListaLigada é pública** (o nome do arquivo segue o nome desta classe)

**Arquivo: ListaLigada.java**

```
class LinkedNode {
   int valor;
   LinkedNode next;
}

public class ListaLigada {
    private LinkedNode first;

    public void adicionaItemNoFinal(int item) {
       ...
    }

    public void removeItem(int item) {
       ...
    }

    public void imprimeListaLigada() {
       ...
    }
}
```

UFABC

# Modificadores de acesso em classes...

**LinkedNode agora é uma classe interna a ListaLigada (desta forma, apenas a classe ListaLigada pode usá-la)**

**Classe ListaLigada é pública** (o nome do arquivo segue o nome desta classe)

```
Arquivo: ListaLigada.java
public class ListaLigada {
  private class LinkedNode {
    int valor;
    LinkedNode next;
  }

  private LinkedNode first;

  public void adicionaItemNoFinal(int item) {
    ...
  }

  public void removeItem(int item) {
    ...
  }

  public void imprimeListaLigada() {
    ...
  }
}
```

UFABC

# Modificadores de acesso em classes...

- Os 4 modificadores podem ser aplicados para **atributos** e **métodos** (como já discutimos em aula);
- Para **classes externas**:
  - Apenas package e public são permitidos;
- Para **classes internas (aninhadas)**:
  - Podemos usar o 4 modificadores.

UFABC

# Exercício 1

- Crie um método para **buscar um elemento** na lista ligada. O método deve retornar **true** se o elemente estiver na lista e, caso contrário, deve retornar **false**.

UFABC

# Exercício 2

- Modifique a lista ligada para armazenar objetos da classe Contas (Exercício 3 da outra lista passada hoje) ao invés de considerar apenas números inteiros (int).

UFABC

# Exercício 3

- Crie um método para inserir elementos de forma ordenada na lista ligada;
  - Por exemplo:
    - Lista contém os valores 5, 6, 90, 94, 150
    - Foi solicitada a inserção do elemento 92
    - Lista final será: 5, 6, 90, **92,** 94, 150

UFABC

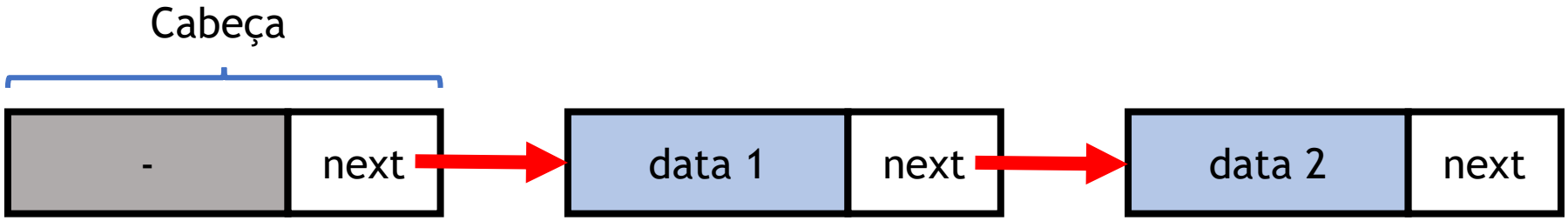
# Exercício 4

- Crie um método para inverter a lista ligada (sem criar outra lista).

UFABC

# Listas simplesmente ligadas **com nó cabeça**

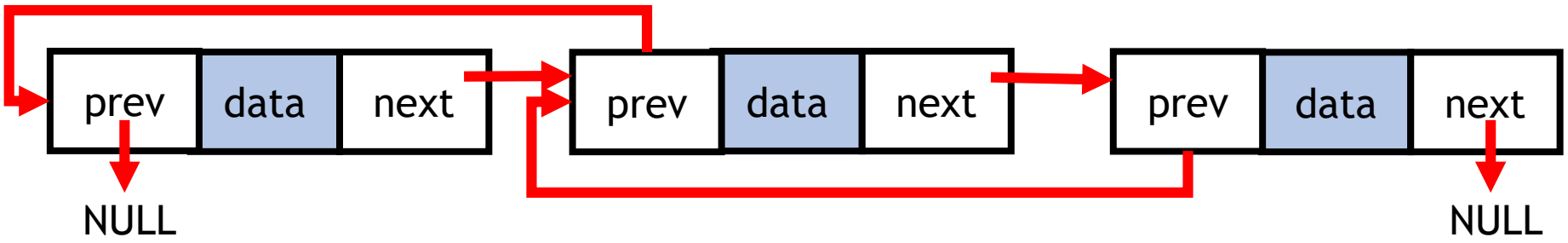
- Cada item é ligado somente ao próximo item;
- O primeiro item não armazena dados da lista (e nunca é excluído);
- **Vantagem**: não é necessário verificar se a lista está vazia (o item cabeça nunca é removido).




UFABC

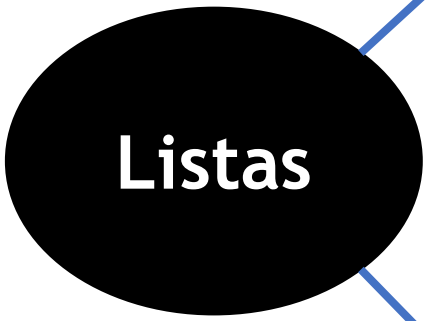
# Listas duplamente ligadas

- Cada item é ligado ao próximo item e também ao anterior;
- **Vantagem**: **a lista pode ser percorrida em ambas as direções.**




UFABC

# Resumo




**Listas com arranjos**

- Simples para usar
- Alocação em bloco contínuo
- Acesso a um item em tempo constante

- Requer saber a quantidade de itens previamente (para alocaçao)
- Inserção/Remoção requer deslocamentos
- Expansão custosa (realocar e copiar)

**Listas ligadas/encadeadas/enlaçadas**

- Não requer conhecer a quantidade de itens previamente
- Inserção e remoção não requer deslocamentos

- Acesso a uma posição necessita percorrer a lista
- Memória extra para os ponteiros

UFABC

# Exercício 5

- Refazer os exercícios anteriores, mas agora para listas duplamente ligadas.

UFABC

# Referências


- Documentação Java: https://docs.oracle.com/javase/8/docs/
- DEITEL, H. M.; DEITEL, P. J. Java: como programar. 6a edição. Porto Alegre, RS: Bookman, 2005.
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Algoritmos: Teoria e Prática. Elsevier, 2012.


UFABC