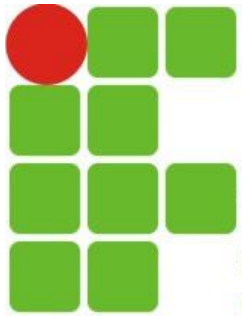


INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SUL DE MINAS GERAIS
Campus Machado

Estrutura de Dados I

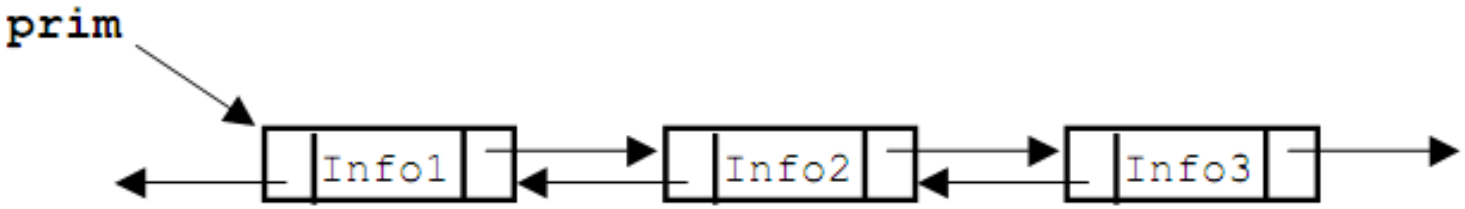
Prof. Matheus Franco

Listas

- A estrutura de lista simplesmente encadeada caracteriza-se por formar um encadeamento simples entre os elementos: cada elemento armazena um ponteiro para o próximo elemento da lista. Desta forma, não temos como percorrer eficientemente os elementos em ordem inversa, isto é, do final para o início da lista.
- O encadeamento simples dificulta a geração de uma lista ordenada e a retirada de um elemento da lista.

Listas duplas

- Para solucionar esses problemas, podemos formar o que chamamos de listas duplamente encadeadas. Nelas, cada elemento tem um ponteiro para o próximo elemento e um ponteiro para o elemento anterior. Desta forma, dado um elemento, podemos acessar ambos os elementos adjacentes: o próximo e o

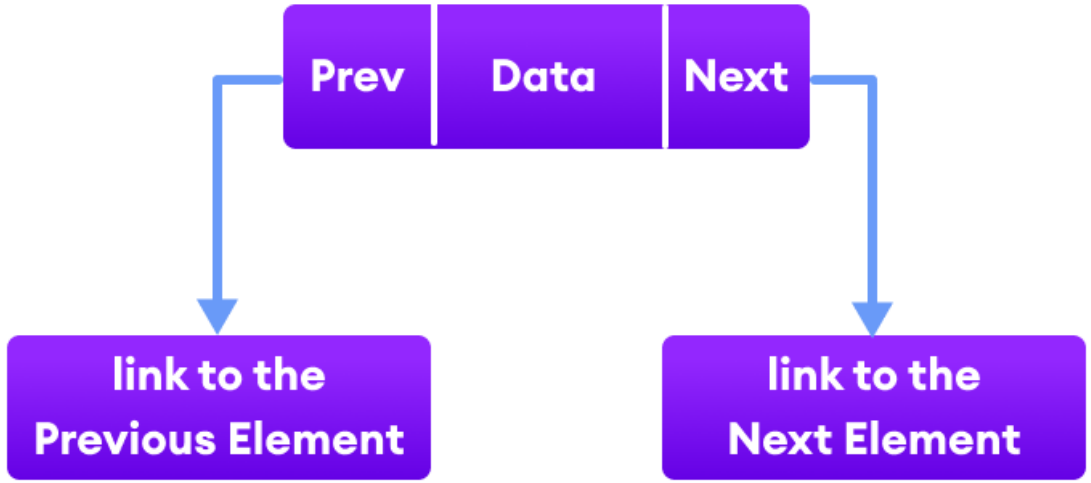


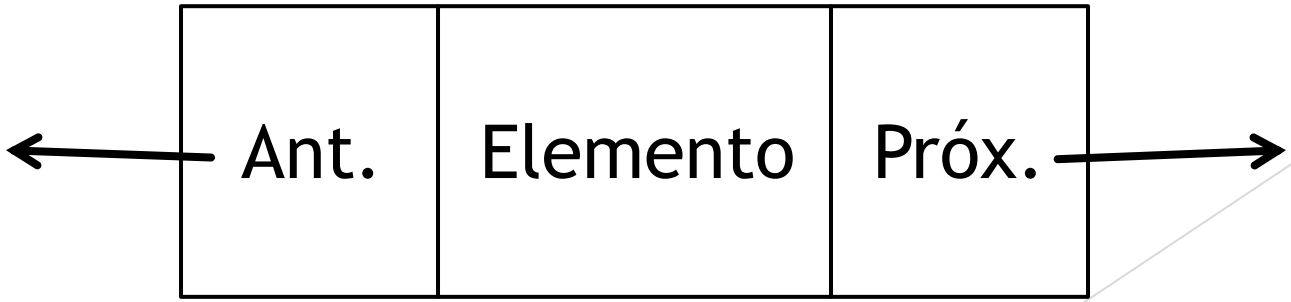
- Vantagens de uma lista encadeada
  - É uma estrutura dinâmica por natureza que aloca a memória quando necessário.
  - Operações de inserção e exclusão pode ser facilmente implementado.
  - Pilhas e filas podem ser facilmente executado.

- Desvantagens
  - Fatia extra de memória para armazenamento dos ponteiros.
  - Nenhum elemento pode ser acessado de forma aleatória; deve-se acessar cada nó sequencialmente.

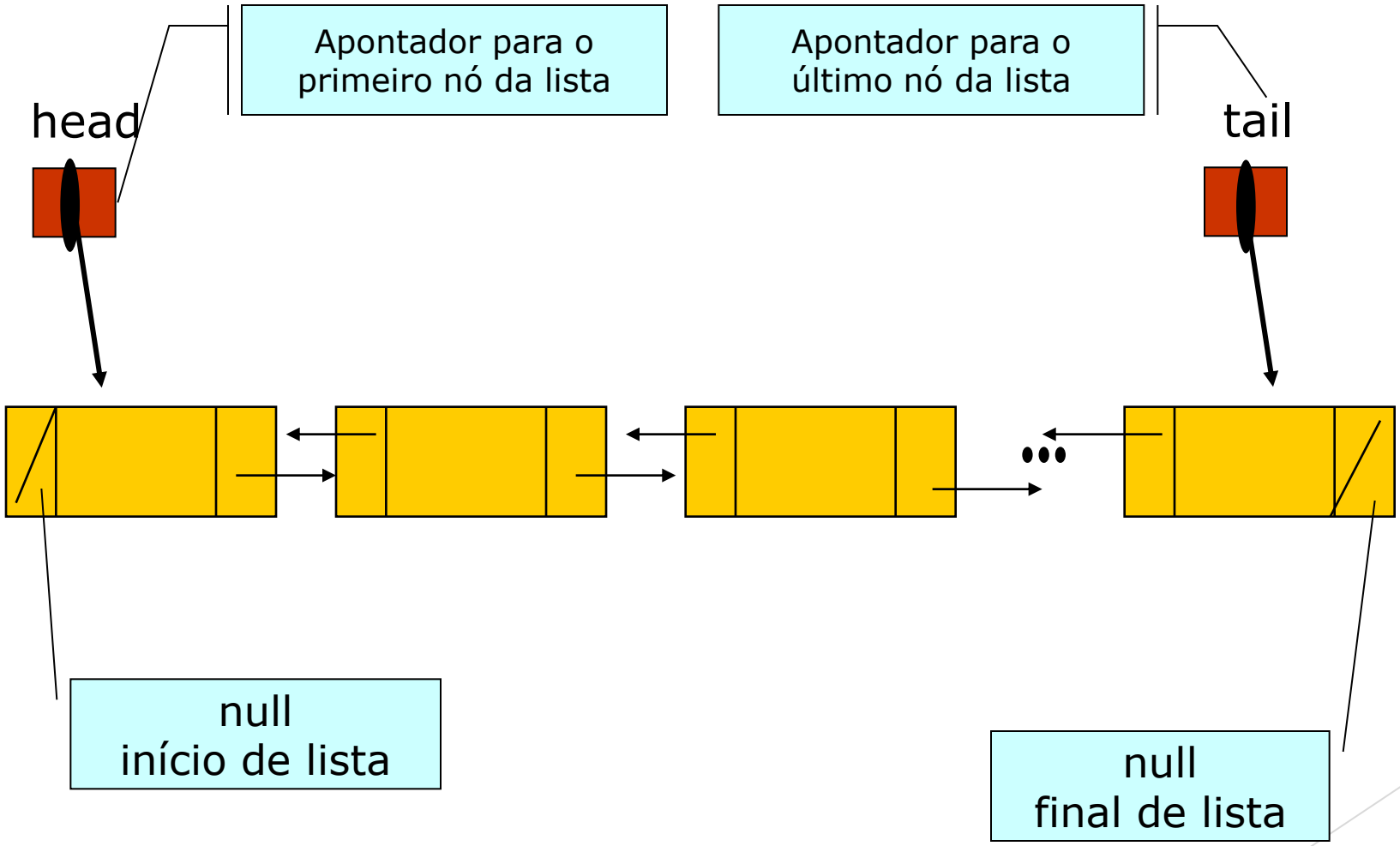
Lista Duplamente Encadeada

- cada nó, além do ponteiro para o **próximo** nodo, também tem uma indicação para o nó **anterior**.



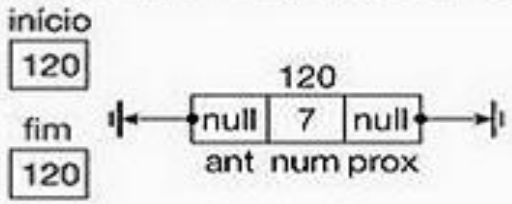


Lista duplamente encadeada com referência ao ultimo elemento da lista

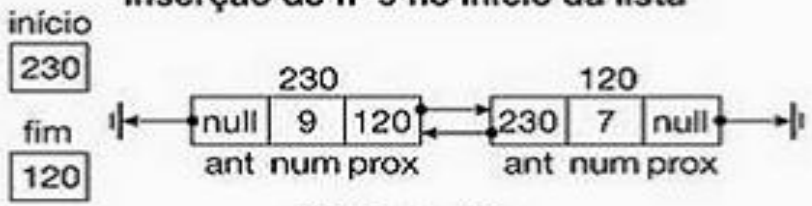


1ª operação
A lista está vazia

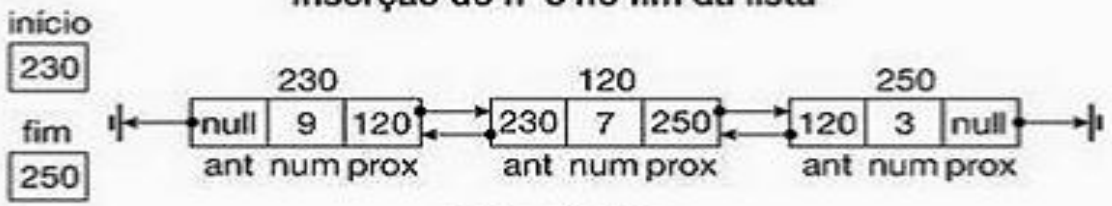
2ª operação
Inserção do nº 7 no início da lista



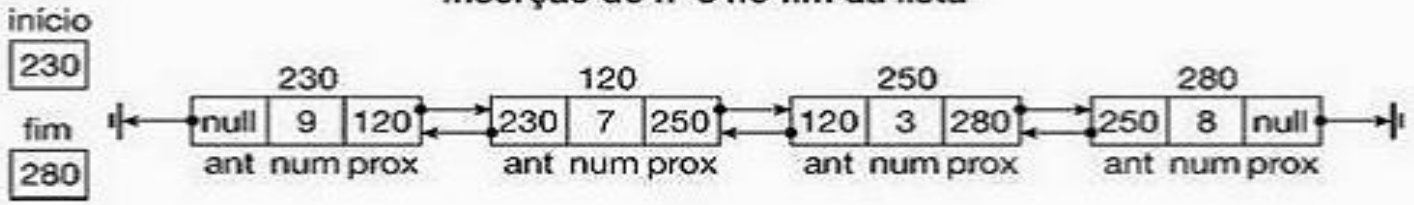
3ª operação
Inserção do nº 9 no início da lista



4ª operação
Inserção do nº 3 no fim da lista

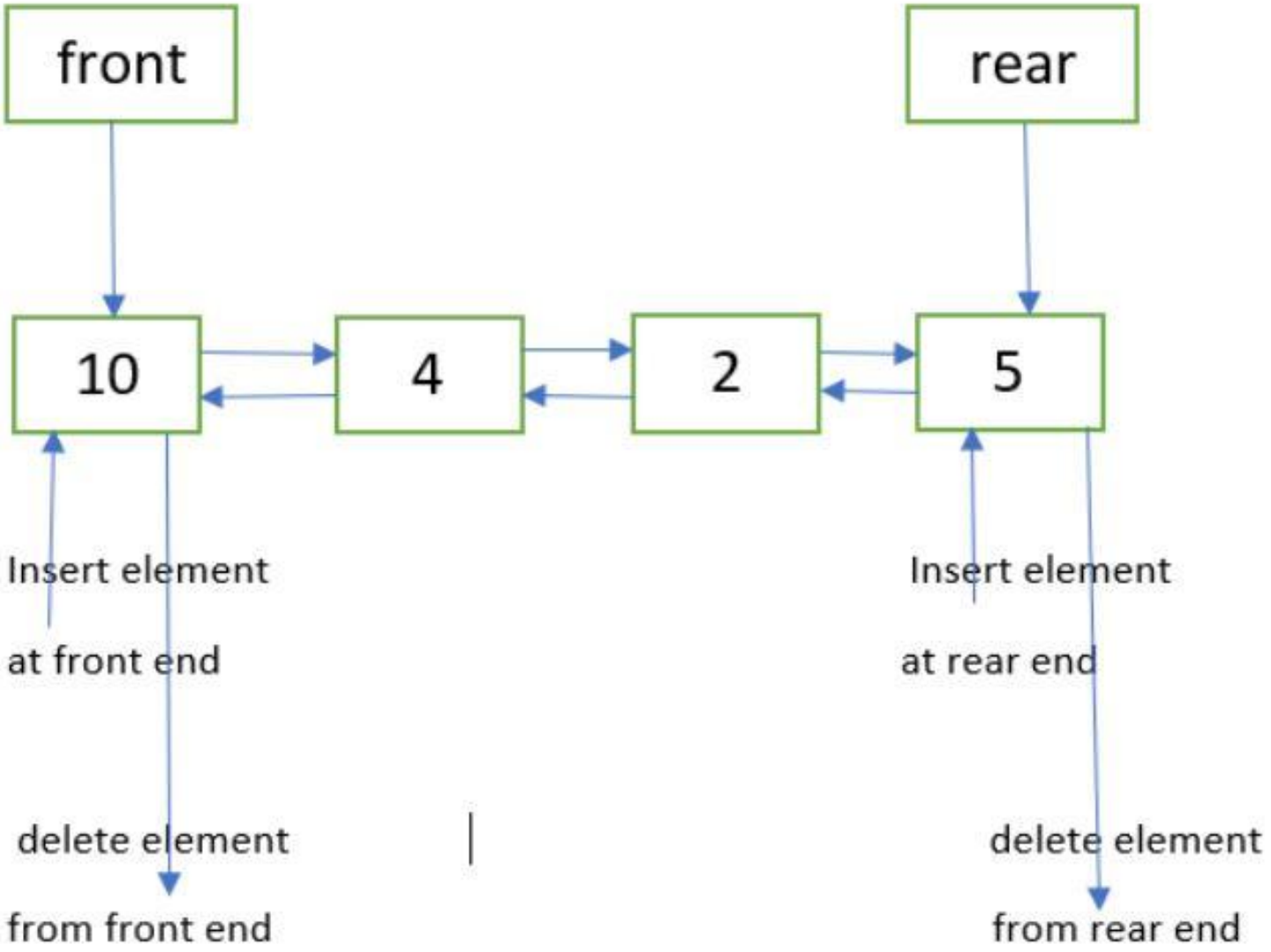


5ª operação
Inserção do nº 8 no fim da lista

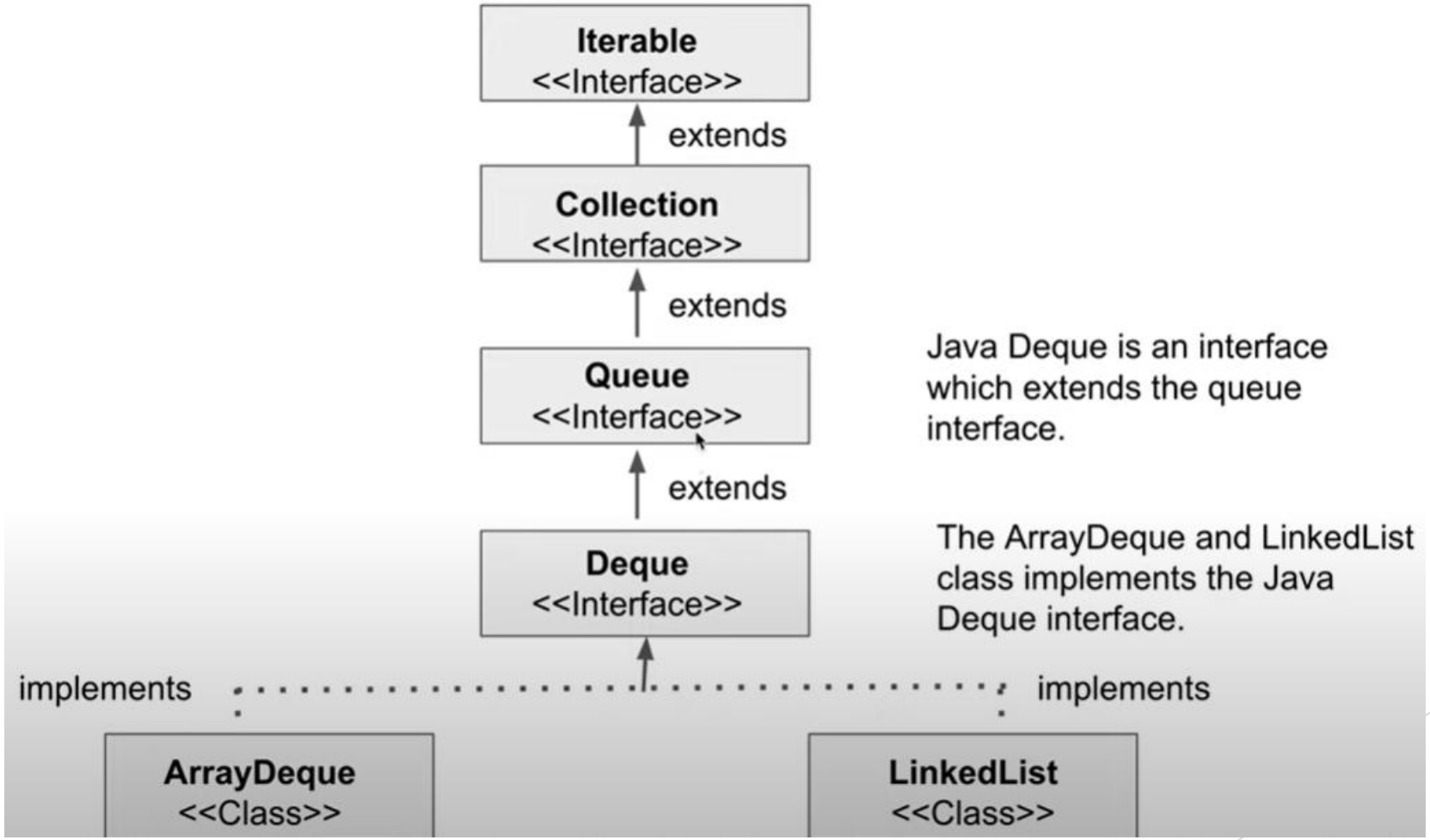


Fila Dupla
DEQUE (double-ended-queue)

- A estrutura de dados que chamamos de fila dupla consiste numa fila na qual é possível inserir novos elementos em ambas as extremidades, no início e no fim.
- Consequentemente, permite-se também retirar elementos de ambos os extremos. É como se, dentro de uma mesma estrutura de fila, tivéssemos duas filas, com os elementos dispostos em ordem inversa uma da outra.
- Pode ser implementado através de vetores ou listas encadeadas.


front
rear
10
4
2
5
Insert element
at front end
Insert element
at rear end
delete element
from front end
delete element
from rear end

Em Java


Iterable
<<Interface>>
extends
Collection
<<Interface>>
extends
Queue
<<Interface>>
extends
Deque
<<Interface>>
Java Deque is an interface which extends the queue interface.
The ArrayDeque and LinkedList class implements the Java Deque interface.
implements
implements
ArrayDeque
<<Class>>
LinkedList
<<Class>>

- A interface Deque em Java representa uma fila de dois extremos, ou seja, permite a inserção e remoção de elementos tanto no início quanto no fim da fila. Isso oferece mais flexibilidade em comparação com uma fila simples (FIFO - First In First Out) ou uma pilha (LIFO - Last In First Out).

```
import java.util.Deque;
import java.util.LinkedList;

public class DequeExample {
    public static void main(String[] args) {
        // Criação de uma Deque usando LinkedList
        Deque<String> deque = new LinkedList<>();

        // Adicionando elementos na deque
        deque.addFirst("Primeiro");
        deque.addLast("Último");

        // Consultando elementos
        System.out.println("Primeiro elemento: " + deque.getFirst());
        System.out.println("Último elemento: " + deque.getLast());

        // Removendo elementos
        deque.removeFirst();
        deque.removeLast();

        // Verificando se a deque está vazia
        System.out.println("Deque está vazia? " + deque.isEmpty());
    }
}
```

Métodos de Inserção

- **void addFirst(E e)**: Insere o elemento especificado no início da deque.
- **void addLast(E e)**: Insere o elemento especificado no fim da deque.
- **boolean offerFirst(E e)**: Tenta inserir o elemento especificado no início da deque e retorna true se for bem-sucedido, false caso contrário.
- **boolean offerLast(E e)**: Tenta inserir o elemento especificado no fim da deque e retorna true se for bem-sucedido, false caso contrário.

Métodos de Remoção

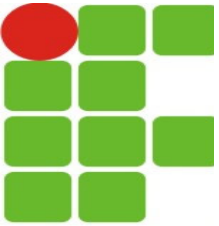
- **E removeFirst()**: Remove e retorna o primeiro elemento da deque. Lança uma exceção se a deque estiver vazia.
- **E removeLast()**: Remove e retorna o último elemento da deque. Lança uma exceção se a deque estiver vazia.
- **E pollFirst()**: Remove e retorna o primeiro elemento da deque, ou retorna null se a deque estiver vazia.
- **E pollLast()**: Remove e retorna o último elemento da deque, ou retorna null se a deque estiver vazia.

Métodos de Consulta

- **E getFirst()**: Retorna o primeiro elemento da deque sem removê-lo. Lança uma exceção se a deque estiver vazia.
- **E getLast()**: Retorna o último elemento da deque sem removê-lo. Lança uma exceção se a deque estiver vazia.
- **E peekFirst()**: Retorna o primeiro elemento da deque, ou retorna null se a deque estiver vazia.
- **E peekLast()**: Retorna o último elemento da deque, ou retorna null se a deque estiver vazia.

Caracteristícas

- Flexibilidade: Permite a inserção e remoção eficiente de elementos em ambas as extremidades da lista.
- Dinamicidade: Diferente de arrays, o tamanho da LinkedList pode crescer e diminuir dinamicamente conforme necessário.
- Complexidade Temporal: As operações de adição e remoção em ambas as extremidades da LinkedList são O(1), o que as torna muito eficientes.

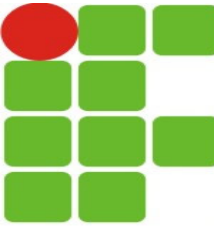


Implementando as Funções

- ***addFisrt()***: Adds an item at the front of Deque.
- ***addLast()***: Adds an item at the rear of Deque.
- ***removeFirst()***: Deletes an item from front of Deque.
- ***removeLast()***: Deletes an item from rear of Deque.

Aplicações

- Pode funcionar como pilha ou fila na mesma estrutura.
- Amplamente utilizado em diferentes algoritmos de escalonamento de processos e controle de processos concorrentes.



Implementação em JS

```
class No {
  constructor(dado) {
    this.dado = dado;
    this.ant = null;
    this.prox = null;
  }
}
```

```
class LinkedList {
  constructor() {
    this.head = null;
    this.tail = null;
    this.length = 0;
  }
}
```

Adicionar no Final

1. Crie o nó;
2. Como o nó vai ser inserido no final, atribua nulo ao próximo;
3. Como esse novo nó deve ser capaz de apontar para seu anterior faça com que o ponteiro do anterior aponte para o antigo final da fila;
4. As demais verificações seguem a mesma lógica da uma fila simples;

Adicionar no Final

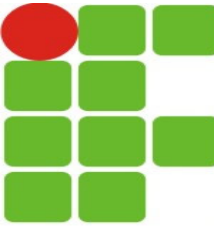
```
addLast(novoDado) {
    const newNo = new No(novoDado);
    if (newNo === null) return false;

    if (this.head === null)
      // Se a lista estiver vazia, o novo nó se torna a cabeça e a cauda
      this.head = newNo;
    else {
      // Caso contrário, adiciona o novo nó à cauda e atualiza a cauda
      newNo.ant = this.tail;
      this.tail.prox = newNo;
    }
    this.tail = newNo;
    this.length++;
    return true;
  }
```

Adicionar no Inicio

```
addFirst(dado) {
    const newNo = new No(dado);
    if (newNo === null) return false;
    if (this.tail === null)
      // Se a lista estiver vazia, o novo nó se torna a cabeça e a cauda
      this.tail = newNo;
    else {
      // Caso contrário, adiciona o novo nó à cabeça e atualiza a cabeça
      newNo.prox = this.head;
      this.head.ant = newNo;
    }
    this.head = newNo;
    this.length++;
    return true;
  }
```

Remoção no inicio

- Utiliza-se uma variável para guarda o dado do nó inicial;
- Desloca-se o ponteiro inicial para seu próximo;
- Faz-se com que o anterior do novo inicio passe a valer nulo;
- Os demais passos seguem a mesma lógica da fila simples.



Remover no início

```
removeFirst() {
  const dadoRemovido = this.head.dado; // Salva o valor do elemento removido
  this.head = this.head.prox;
  if (this.head !== null) {
    this.head.ant = null;
  } else this.tail = null;
  return dadoRemovido; // Retorna o valor do elemento removido
}
```

Funções a criar

- Percurso inverso: Crie uma função que seja capaz de apresentar o percurso inverso em um fila dupla.

- **Atividade**: Crie funções para remover no final da DEQUE;

Atividades

Considerando um deque implementado como lista duplamente encadeada, implemente as seguintes operações:

1. verificar se está vazio;
2. obter o primeiro elemento;
3. obter o último elemento;
4. obter o penúltimo elemento;
5. A quantidade de elementos
6. Verificar se um elemento está na fila
7. Obter o maior elemento da fila