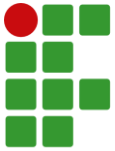


INSTITUTO FEDERAL
Sertão Pernambucano
Campus Salgueiro

Estrutura de Dados e Algoritmos com Java

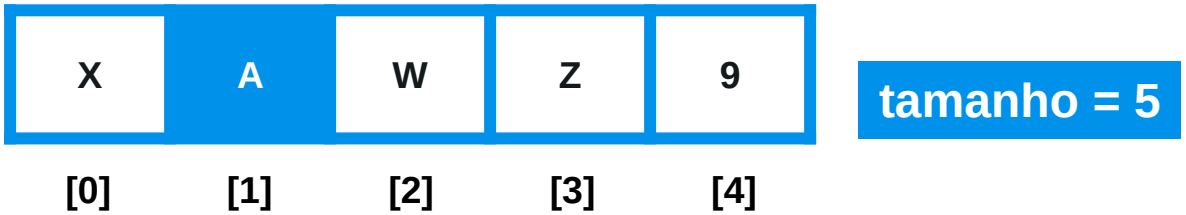
Prof. Heraldo Gonçalves Lima Junior
heraldo.junior@ifsertao-pe.edu.br

1. Listas Lineares Sequenciais

CONTINUAÇÃO...

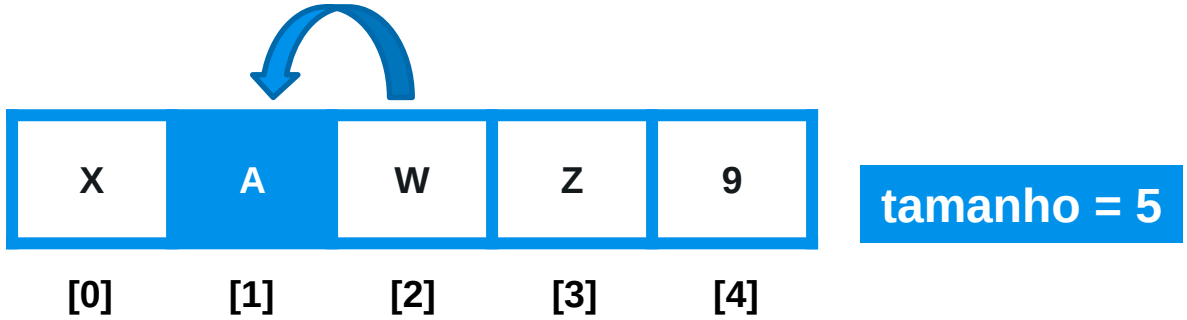
1.5. Remover um elemento de uma dada posição

- Suponha que precisemos excluir o valor **A**, contido no índice [1].
- Lembram da lógica do método **adiciona( )**?
- A lógica agora é contrária, agora empurraremos os elementos na direção do elemento que queremos remover.



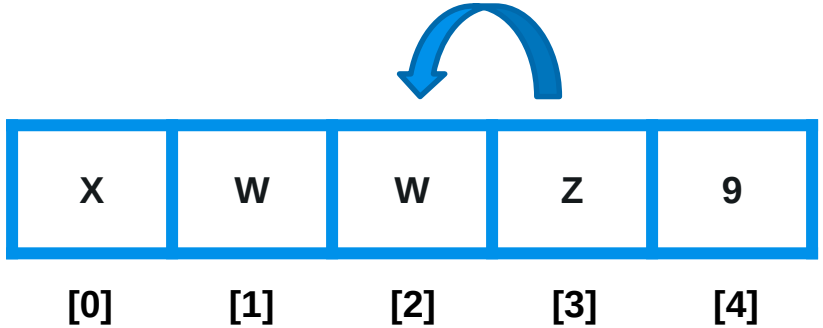
1.5. Remover um elemento de uma dada posição

- A lógica será a seguinte:
- Posição a ser removida: [1] **elementos[1] = elementos[2]**



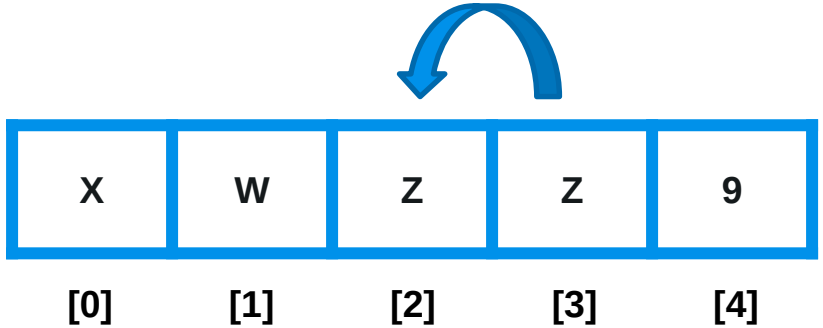
1.5. Remover um elemento de uma dada posição

- A lógica será a seguinte:
- Posição a ser removida: [1] **elementos[2] = elementos[3]**



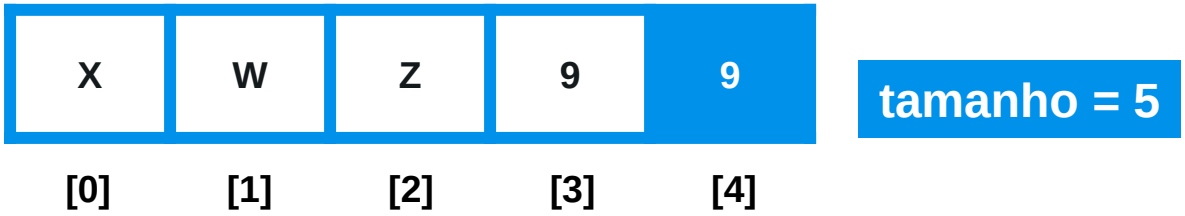
1.5. Remover um elemento de uma dada posição

- A lógica será a seguinte:
- Posição a ser removida: [1] **elementos[3] = elementos[4]**



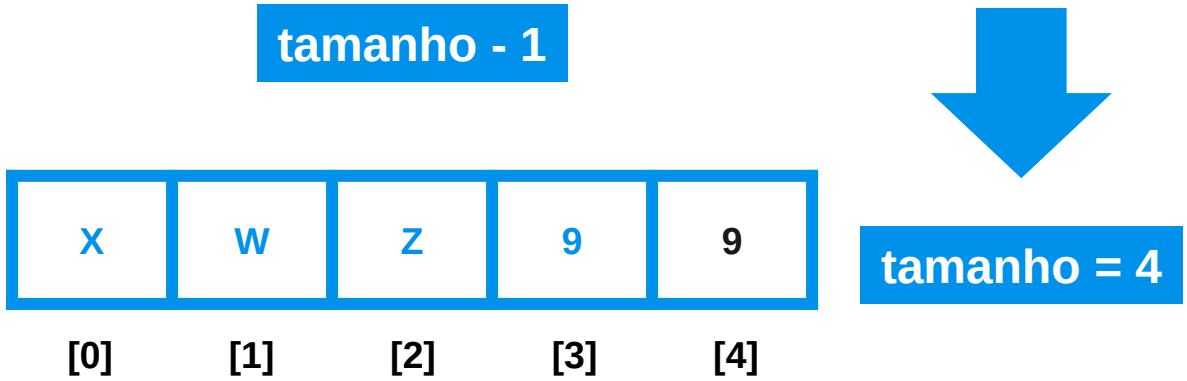
1.5. Remover um elemento de uma dada posição

- Pronto. Elemento removido!
- Mas e agora? O que faremos com o 9 do índice [4]?



1.5. Remover um elemento de uma dada posição

- Reduzimos o atributo tamanho, já que retiramos um elemento.



- O 9 do índice 4 passa a ser ignorado, já que sempre trabalhamos com o atributo tamanho.

1.5. Remover um elemento de uma dada posição

- Então é isso, mas para garantirmos que a posição informada pelo usuário seja válida, precisamos mais uma vez fazer essa verificação.

posicao >= 0 **posicao < tamanho**

1.5. Remover um elemento de uma dada posição

- Nosso código fica assim:

```
public void remove(int posicao){
    if(!((posicao >= 0) && (posicao < this.tamanho))){
        throw new IllegalArgumentException("Posição inválida!");
    }

    for(int i=posicao;i<tamanho-1;i++){
        this.elementos[i] = this.elementos[i+1];
    }

    this.tamanho--;
}
```

1.5. Remover um elemento de uma dada posição

- Podemos fazer outra versão deste código, sendo que o usuário passa o valor do elemento como parâmetro, e não o elemento.
- Podemos utilizar em conjunto o método **verifica( )**.

1.5. Remover um elemento de uma dada posição

- Agora o nosso código, na classe teste, ficou assim:

```
int pos = vetor.verifica("x");
if (pos > -1){
    vetor.remove(pos);
} else {
    System.out.println("O elemento não existe no vetor!");
}
```

1.6. Generalizando o tipo do vetor

- A implementação de vetor feita até agora funciona muito bem para armazenar elementos String. Porém, não serve para armazenar nenhum outro tipo de objeto. Nossa estrutura de dados está muito atrelada ao tipo de dado que ela armazena .
- Se amanhã ou depois precisarmos de uma Lista de idades ou uma Lista de salários teríamos que implementar novamente o Vetor.

1.6. Generalizando o tipo do vetor

- Em vez de colocarmos um array de Elementos na classe Vetor, vamos colocar um **array de Object**. Assim, estamos generalizando a nossa estrutura de dados. Desta forma, poderemos armazenar qualquer tipo de objeto.

1.6. Generalizando o tipo do vetor

- No Java todas as classes herdam, diretamente ou indiretamente, da **classe Object**. Então, **um objeto de qualquer tipo pode ser referenciado com uma variável do tipo Object**. Este conceito de referenciar um mesmo objeto de várias maneira (Elemento ou Object) é chamado de **polimorfismo**.

CALMA, RESPIRA!

Obrigado!

Perguntas?

 heraldo.junior@ifsertao-pe.edu.br

 heraldolimajr.com.br