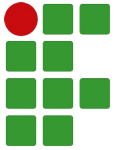


INSTITUTO FEDERAL
Sertão Pernambucano
Campus Salgueiro

Estrutura de Dados e Algoritmos com Java

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

1. Pilhas Dinâmicas

1.1. Características de uma Pilha Dinâmica

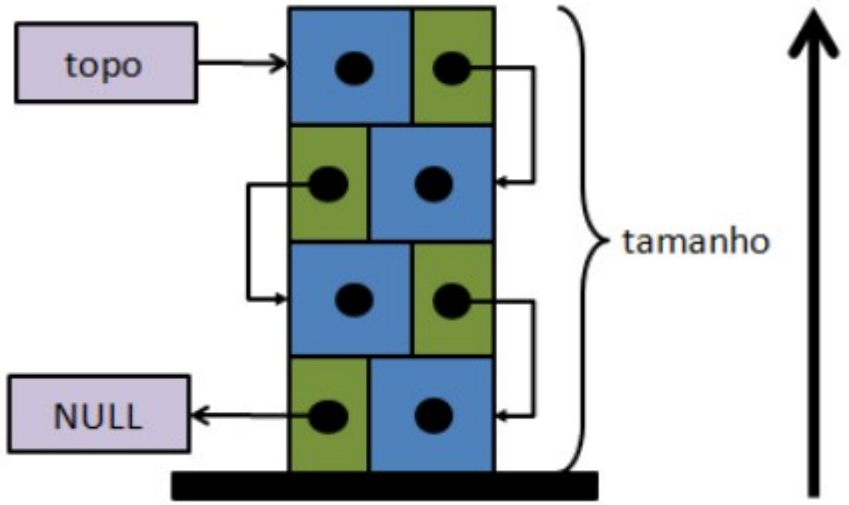
- Assim como as listas, também é possível implementar pilhas utilizando alocação dinâmica de memória. A lógica das operações continua a mesma, sempre no topo.

- **LIFO**

(o último a entrar é o primeiro a sair**, “Last In, First Out”**, em inglês).

1.1. Características de uma Pilha Dinâmica

- Essencialmente a implementação dinâmica de pilhas pode ser feita de diversas maneiras. Neste contexto, **estudaremos a implementação dinâmica usando listas encadeadas**.



1.1. Características de uma Pilha Dinâmica

- Na implementação dinâmica, duas estruturas de dados são necessárias.
- Uma para representar os nós individuais que compõem a pilha (classe **No)** e uma estrutura de cabeçalho que contem a referência para o primeiro elemento da pilha, referência para seu topo e facultativamente o número de elementos existentes na pilha (classe **Pilha**).

1.1. Características de uma Pilha Dinâmica

- Criamos então a classe **No** com seus gets e sets.

```
public class No {
    private Object elemento;
    private No proximo;
}
```

1.2. Criando um nó

- Ainda na classe **No**, criaremos o construtor, que recebe como parâmetros o elemento e a referência pro próximo:

```
public No(Object elemento, No proximo) {
    this.elemento = elemento;
    this.proximo = proximo;
}
```

1.3. Implementando a pilha

- Na nossa classe Pilha, teremos uma referência para o topo da pilha e um atributo para armazenar o tamanho.

```
public class Pilha {
    private No topo;
    private int tamanho;
}
```

1.4. Criando a pilha

- O construtor da nossa pilha apenas define tamanho = 0 e topo = null.

```
public Pilha() {
    this.topo = null;
    this.tamanho = 0;
}
```

1.5. Inserindo um elemento na pilha (Push)

- Para inserirmos um novo elemento, basta **criar um novo nó**, **atualizar a referência de topo** e **incrementar o tamanho**.

```
public void push(Object elemento) {
    No novo = new No(elemento, this.topo);
    this.topo=novo;
    this.tamanho++;
}
```

1.6. Verificando se a pilha está vazia (IsEmpty)

- Para verificarmos se a pilha está vazia, basta verificar o valor do atributo topo. Se for menor que zero, nenhum elemento está contido na pilha.

```
public boolean isEmpty() {
    return this.topo==null;
}
```

1.7. Verificando o tamanho da pilha (Size)

- O tamanho da pilha pode ser verificado através do atributo tamanho.

```
public int size() {
    return this.tamanho;
}
```

1.8. Verificando o elemento do topo (Top)

- O elemento do topo é o único que pode ser manipulado, então, para verificarmos quem é esse elemento, basta acessar a posição topo no nosso vetor.

```
public Object top() {
    return this.topo.getElemento();
}
```

1.9. Removendo o elemento do topo (Pop)

- Para remover, se a pilha não estiver vazia, basta **salvar o valor de topo**, **apontar topo para o seu próximo** e decrementar o atributo **tamanho**.

```
public Object pop() {
    if(!isEmpty()) {
        Object elementoTopo = this.topo.getElemento();
        this.topo = this.topo.getProximo();
        this.tamanho--;
        return elementoTopo;
    }
    return null;
}
```

2. API Java

2.1. A classe Stack

- Na biblioteca do Java existe uma classe que implementa a estrutura de dados pilha, esta classe chama-se **Stack** e implementa, dentre outros métodos, os listados abaixo:
  - **Push( ) – Insere no topo;**
  - **Pop( ) – Remove do topo;**
  - **Peek( ) – Verifica o elemento do topo;**
  - **Search( ) – Verifica a profundidade de um dado elemento;**
  - **Empty( ) – Verifica se a pilha está vazia.**

RES PIRA !

Obrigado!

Perguntas?

 heraldo.junior@ifsertao-pe.edu.br

 heraldolimajr.com.br