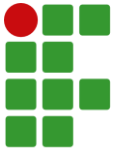


INSTITUTO FEDERAL
Sertão Pernambucano
Campus Salgueiro

Estrutura de Dados e Algoritmos com Java

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

1. Pilhas

1.1. Características de uma Pilha

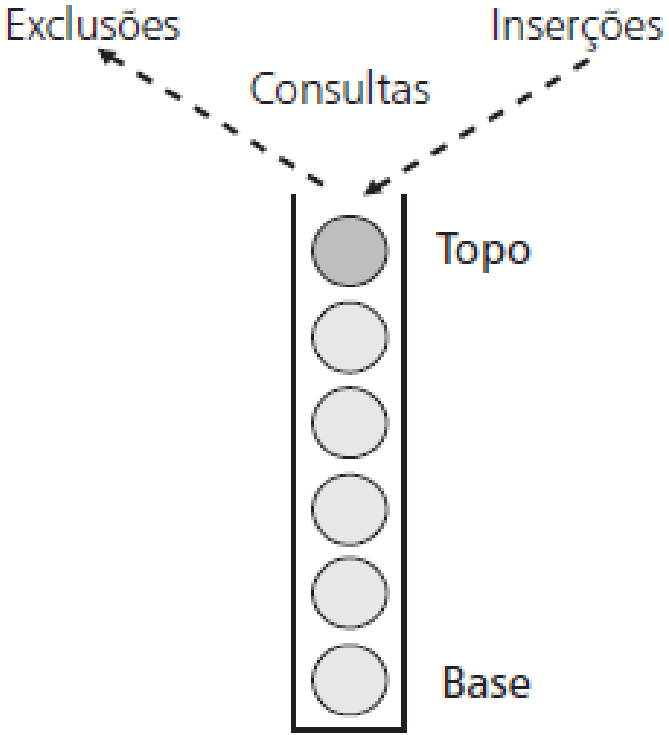
- **Pilhas são listas** nas quais o acesso somente pode ser feito em uma das extremidades, denominada de **topo da pilha**.

- **LIFO**

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

1.1. Características de uma Pilha

- Todas as consultas, alterações, inclusões e remoções de nodos somente podem ser realizadas sobre um nodo, que é aquele que está na extremidade considerada o **topo da pilha.**



1.2. Pilhas x Listas

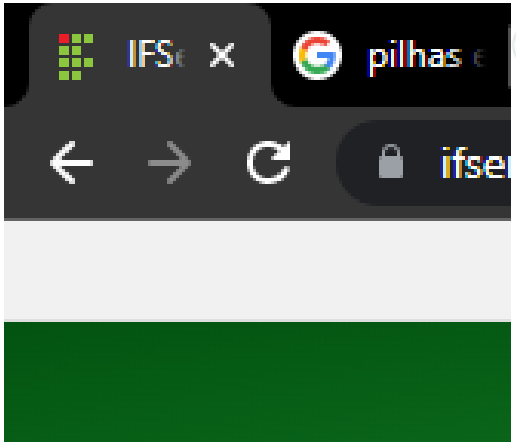
- A diferença entre essas duas estruturas de dados está nas operações.
- **As operações de uma Pilha são mais restritas do que as de uma Lista.**
- **A Pilha possui apenas um subconjunto de operações da Lista.** Então o interessante é que para implementar uma Pilha podemos usar uma Lista.

1.3. Exemplos de uso

- Em uma análise simples, poderia ser utilizada, por exemplo, em um carregamento de um caminhão, pois se o caminhão tiver 4 entregas, a última entrega colocada dentro do caminhão deve ser a primeira a sair, caso contrário, pode dar mais trabalho para descarregar.

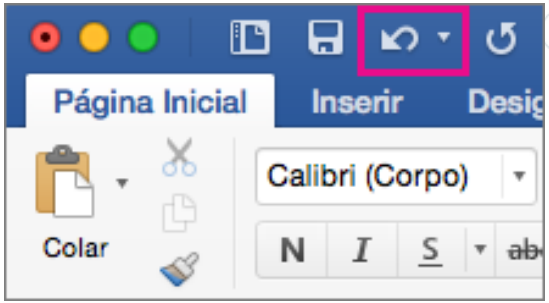
1.3. Exemplos de uso

- Navegadores para Internet também utilizam pilhas. Cada vez que o navegador visita um novo site, o endereço é armazenado na pilha de endereços. Usando a operação de retorno (“back”), o navegador permite que o usuário retorne ao último site visitado, retirando o seu endereço da pilha.



1.3. Exemplos de uso

- Pilhas são estruturas de dados fundamentais, sendo usadas em muitas aplicações em computação. Por exemplo, editores de texto oferecem um mecanismo de reversão de operações (“undo”) que cancela operações recentes e reverte um documento ao estado anterior à operação.



1.4. Operações Básicas

- As operações que podem ser realizadas sobre uma pilha são limitadas pela disciplina de acesso que apresentam. Assim, somente as seguintes operações podem ser executadas:
  - **Inserir (PUSH)**
  - **Remover (POP)**
  - **Verificar se é vazia (IS EMPTY)**
  - **Tamanho (SIZE)**
  - **Verificar qual é o elemento do topo (TOP)**

2. Pilhas Estáticas

2.1. Implementação da Pilha Estática

- Para criar a nossa classe Pilha, utilizaremos um **vetor do tipo Object** onde guardaremos os **elementos da pilha,** um atributo para armazenar a **posição atual do topo**.

```
public class Pilha {
    private Object elementos[];
    private int topo;
```

2.2. Criando uma pilha

- A criação de uma pilha resultará em uma pilha vazia, devolvendo ao usuário as informações necessárias para seu posterior acesso.
- Deverá ser utilizada alguma convenção para indicar que a pilha está vazia. Optamos pela seguinte estratégia: **a indicação de que a pilha está vazia é feita quando o índice de topo = -1,** pois a posição zero do array já armazena informação.

2.2. Criando uma pilha

```
public Pilha(int capacidade) {
    this.topo=-1;
    this.elementos = new Object[capacidade];
}
```

2.3. Inserindo um novo elemento

- Para inserirmos um novo elemento, primeiramente devemos **verificar se a pilha não está cheia**. Feito isso, **basta inserir o novo elemento e incrementar o topo**.

```
public boolean push(Object elemento) {
    if(this.topo < this.elementos.length-1) {
        this.topo++;
        this.elementos[topo] = elemento;
        return true;
    }
    return false;
}
```

2.4. Verificando se a pilha está vazia

- 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 topo<0;
}
```

2.5. Verificando o tamanho da pilha

- O tamanho da pilha pode ser verificado através do atributo **topo,** que traz a posição do último elemento da pilha(o que está no topo).

```
public int size() {
    if(this.isEmpty()) {
        return 0;
    }
    return this.topo+1;
}
```

2.6. Verificando o elemento do topo

- 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() {
    if(this.isEmpty()) {
        return null;
    }
    return this.elementos[this.topo];
}
```

2.7. Removendo um elemento

- Na estrutura Pilha, só podemos remover o elemento do topo.

```
public Object pop() {
    if(this.isEmpty()) {
        return null;
    }else {
        return this.elementos[this.topo--];
    }
}
```

2.8. Conclusão

- Como pode-se imaginar, a implementação estática de pilhas, assim como as listas, impõe várias restrições no que se refere a capacidade de crescimento da estrutura de dados.
- Na próxima aula, veremos como implementar pilhas através de alocação dinâmica.

2.9. Exercícios

- 1. Implemente os códigos vistos em aula e envie através do Classroom.
- 2. Altere o método **pop()** para não retornar nenhuma elemento e só remover o elemento do topo da pilha.

CALMA, RESPIRA!

Obrigado!

Perguntas?

 heraldo.junior@ifsertao-pe.edu.br

 heraldolimajr.com.br