



**INSTITUTO
FEDERAL**

Sudeste de
Minas Gerais

Campus
Manhuaçu

Estruturas de Dados I

Pilhas

Prof. Leonardo C. R. Soares - leonardo.soares@ifsudestemg.edu.br

Instituto Federal do Sudeste de Minas Gerais

4 de outubro de 2023





Pilhas

Descrição

- Uma pilha é um contêiner de objetos que são inseridos e retirados de acordo com o princípio de que o **último que entra é o primeiro que sai** (do inglês, Last In First Out, LIFO).





Operações

O tipo abstrato de dados (TAD) pilha deve, obrigatoriamente, suportar os métodos:

- `push(o)`: Insere o objeto `o` no topo da pilha.





Operações

O tipo abstrato de dados (TAD) pilha deve, obrigatoriamente, suportar os métodos:

- ▶ `push(o)`: Insere o objeto `o` no topo da pilha.
- ▶ `pop()`: Retira o objeto no topo da pilha e o retorna; se a pilha estiver vazia, ocorre um erro.





Operações

O tipo abstrato de dados (TAD) pilha deve, obrigatoriamente, suportar os métodos:

- ▶ `push(o)`: Insere o objeto `o` no topo da pilha.
- ▶ `pop()`: Retira o objeto no topo da pilha e o retorna; se a pilha estiver vazia, ocorre um erro.

Adicionalmente, podemos definir os seguintes métodos auxiliares:

- ▶ `tamanho()`: Retorna o número de objetos na pilha.





Operações

O tipo abstrato de dados (TAD) pilha deve, obrigatoriamente, suportar os métodos:

- ▶ `push(o)`: Insere o objeto `o` no topo da pilha.
- ▶ `pop()`: Retira o objeto no topo da pilha e o retorna; se a pilha estiver vazia, ocorre um erro.

Adicionalmente, podemos definir os seguintes métodos auxiliares:

- ▶ `tamanho()`: Retorna o número de objetos na pilha.
- ▶ `vazia()`: Retorna um *boolean* indicando se a pilha está vazia.





Formas de implementação

Existem várias opções de estruturas de dados que podem ser usadas para representar pilhas. As duas representações mais utilizadas são:

- ▶ Por meio de arranjos.
- ▶ Por meio de referências.

Independente da forma de implementação, uma pilha é uma lista com restrições quanto às formas de inserção e remoção, o que permite a reusabilidade de código.





Implementação por referência

- Cada célula de uma pilha contém um item da pilha e um apontador para outra célula.





Implementação por referência

- ▶ Cada célula de uma pilha contém um item da pilha e um apontador para outra célula.
- ▶ A estrutura contém um apontador para o topo da pilha (célula cabeça).





Implementação por referência

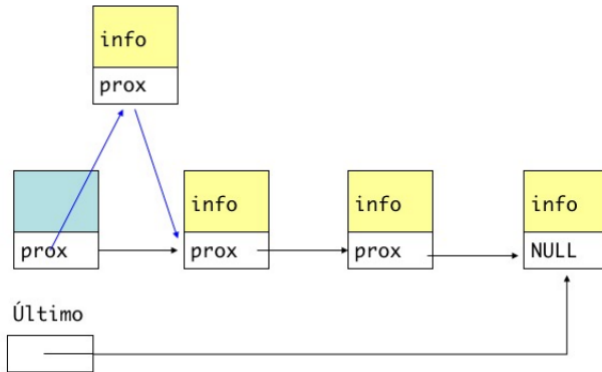
- ▶ Cada célula de uma pilha contém um item da pilha e um apontador para outra célula.
- ▶ A estrutura contém um apontador para o topo da pilha (célula cabeça).
- ▶ Criar um campo tamanho evita a contagem do número de itens na função tamanho.





Implementação por referência - Inserção

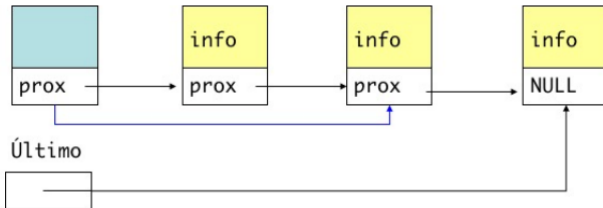
De acordo com a política **LIFO**, há apenas uma opção de posição onde podemos inserir elementos: o topo da pilha (ou seja, primeira posição).





Implementação por referência - Remoção

De acordo com a política **LIFO**, há apenas uma opção de posição onde podemos remover elementos: o topo da pilha (ou seja, primeira posição).





Complexidade

A complexidade de todas as operações é mantida da implementação de Lista:

- ▶ Empilhar: $\theta(1)$
- ▶ Desempilhar: $\theta(1)$





Perguntas?





```
public class Processo {  
    int codigo;  
    String responsavel;  
    String cliente;  
    Processo prox;  
  
    public Processo(int codigo, String responsavel, String cliente) {  
        this.codigo = codigo;  
        this.responsavel = responsavel;  
        this.cliente = cliente;  
        this.prox = null;  
    }  
}
```





```
public class PilhaReferencia {  
    int tam = 0;  
    Processo topo;  
    public boolean isVazia(){  
        return tam==0;  
    }  
    public int getTamanho(){  
        return tam;  
    }  
    public void push(Processo p){  
        if (tam!=0)  
            p.prox = topo;  
        topo = p;  
        ++tam;  
    }  
    public Processo pop() throws Exception{  
        if (tam==0)  
            throw new Exception ("Pilha vazia");  
        Processo p = topo;  
        topo = topo.prox;  
        --tam;  
        return p;  
    }  
}
```





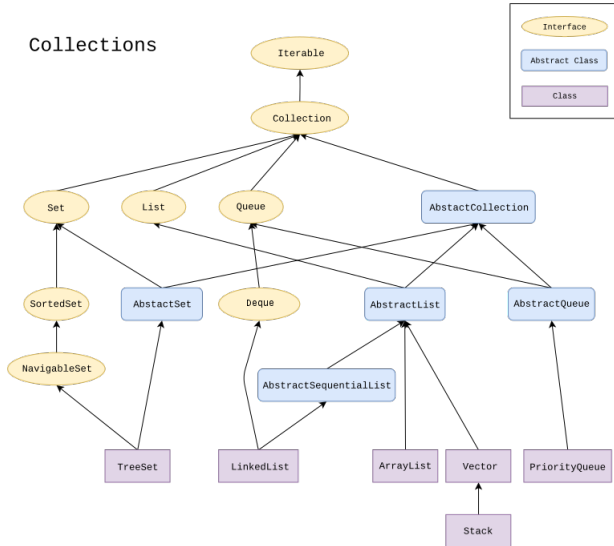
```
public class Main {  
    public static void main(String []args) throws Exception{  
        //PilhaArranjo pilha = new PilhaArranjo();  
        PilhaReferencia pilha = new PilhaReferencia();  
        Processo proc;  
        pilha.push(new Processo(1,"Rosimeire","Acme"));  
        pilha.push(new Processo(2,"Afonso","Samsung"));  
        pilha.push(new Processo(3,"Rosimeire","Lenovo"));  
        pilha.push(new Processo(4,"Ana","Lenovo"));  
        pilha.push(new Processo(5,"Afonso","Acme"));  
        pilha.push(new Processo(6,"Rosimeire","Samsung"));  
  
        System.out.println("Lista de processos a serem executados");  
        while(!pilha.isVazia()){  
            proc = pilha.pop();  
            System.out.printf("Código: %d\t\tResponsável: %s\t\tCliente: %s\n",  
                proc.codigo,proc.responsavel,proc.cliente);  
        }  
    }  
}
```





Utilizando Java Collections Framework

Collections





```
public class Processo {  
    int codigo;  
    String responsavel;  
    String cliente;  
    public Processo(int codigo, String responsavel, String cliente) {  
        this.codigo = codigo;  
        this.responsavel = responsavel;  
        this.cliente = cliente;  
    }  
    public Processo() {  
    }  
}
```





```
import java.util.Stack;
public class Main {
    public static void main(String []args) throws Exception{
        //PilhaArranjo pilha = new PilhaArranjo();
        //PilhaReferencia pilha = new PilhaReferencia();
        Stack<Processo> pilha = new Stack();

        Processo proc;
        pilha.push(new Processo(1, "Rosimeire", "Acme"));
        pilha.push(new Processo(2, "Afonso", "Samsung"));
        pilha.push(new Processo(3, "Rosimeire", "Lenovo"));
        pilha.push(new Processo(4, "Ana", "Lenovo"));
        pilha.push(new Processo(5, "Afonso", "Acme"));
        pilha.push(new Processo(6, "Rosimeire", "Samsung"));

        System.out.println("Lista de processos a serem executados");
        while(!pilha.isEmpty()){
            proc = pilha.pop();
            System.out.printf("Código: %d\t\tResponsável: %s\t\tCliente: %s\n",
                               proc.codigo, proc.responsavel, proc.cliente);
        }
    }
}
```





1. Descreva a saída da seguinte sequência de operações sobre uma pilha:
`push(5), push(3), pop(), push(2), push(8), pop(), pop(), push(9), push(1), pop(), push(7), push(6), pop(), pop(), push(4), pop(), pop()`.
2. Implemente um pequeno **jogo de 21** para dois jogadores. A cada jogada, é sorteada uma carta $\in [1, 2, \dots, 13]$. O jogador pode pegar quantas cartas desejar e sua pontuação é igual a soma dos valores de suas cartas. Caso a pontuação ultrapasse 21, a última carta será descartada, entretanto, isso gerará uma penalidade de 5 pontos. O jogador 2 só joga após o jogador 1 encerrar suas jogadas. Ao final, vence o jogo quem estiver mais próximo de 21 pontos. Exiba as cartas do jogador vencedor.





Exercícios

As sequências de lccanobif são sequências onde cada termo é sempre igual a soma dos dois próximos subsequentes a eles. Exceto pelos dois últimos termos os quais são sempre iguais a 1. Por exemplo, a sequência: 55, 34, 21, 13, 8, 5, 3, 2, 1, 1 - é uma série lccanobif contendo 10 termos.

Escreva um programa que, dado um valor inteiro, imprima a sequência de lccanobif de tamanho correspondente. Seu programa deve utilizar pilha para manipular os dados em memória.





Exercícios

Desenvolva um método para manter duas pilhas dentro de um único vetor linear (um arranjo) de modo que nenhuma das pilhas incorra em estouro até que toda a memória seja usada, e toda uma pilha nunca seja deslocada para outro local dentro do vetor.







Referências

- ▶ CARVALHO, Marco Antonio Moreira de. **Projeto e análise de algoritmos**. 01 mar. 2018, 15 jun. 2018. Notas de Aula. PPGCC. UFOP
- ▶ GOODRICH, Michael T.; TAMASSIA, Roberto. **Estruturas de Dados & Algoritmos em Java**. Bookman Editora, 2013.
- ▶ ZIVIANI, Nivio. **Projeto de Algoritmos com implementações em Java e C++**, 2007.

