**Pilhas em Java**

Notas do Prof. Paulo Barreto

Em Java, as pilhas são implementadas usando a classe java.util.Stack, que é uma subclasse de Vector e segue o princípio LIFO (*Last In, First Out*). No entanto, é importante notar que a documentação do Java recomenda o uso de Deque em vez de Stack para a maioria dos casos, devido a algumas limitações e depreciação de certos métodos em Stack.

Aqui está um exemplo de como usar Stack em Java:

import java.util.Stack;

public class ExemploPilha {

public static void main(String[] args) {

Stack<Integer> pilha = new Stack<>();

// Empilhando elementos

pilha.push(10);

pilha.push(20);

pilha.push(30);

// Acessando o topo da pilha sem removê-lo

System.out.println("Topo da pilha: " + pilha.peek());

// Desempilhando elementos

while (!pilha.isEmpty()) {

System.out.println("Desempilhando: " + pilha.pop());

}

}

}

Neste exemplo, criamos uma instância de Stack chamada pilha que armazena números inteiros. Em seguida, empilhamos três elementos (10, 20 e 30) usando o método push().

Depois, usamos peek() para obter o elemento no topo da pilha sem removê-lo. Finalmente, usamos um loop para desempilhar e imprimir os elementos até que a pilha esteja vazia.

Lembre-se de que Stack é uma classe sincronizada, o que significa que seus métodos são thread-safe. No entanto, se você estiver escrevendo código que não envolve concorrência, pode ser mais eficiente usar Deque em vez de Stack.

Para usar Deque, você pode fazer algo assim:

import java.util.Deque;

import java.util.LinkedList;

public class ExemploDeque {

public static void main(String[] args) {

Deque<Integer> deque = new LinkedList<>();

// Empilhando elementos

deque.push(10);

deque.push(20);

deque.push(30);

// Acessando o topo do deque sem removê-lo

System.out.println("Topo do deque: " + deque.peek());

// Desempilhando elementos

while (!deque.isEmpty()) {

System.out.println("Desempilhando: " + deque.pop());

}

}

}

Alguns exemplos de como usar pilhas em Java em situações comuns. Para isso, usarei a classe Stack da biblioteca Java, que é uma implementação de pilha disponível no pacote java.util.

Certifique-se de importar a classe java.util.Stack para utilizar a pilha em seu código Java.

Exemplo 1: Validando Expressões Matemáticas

import java.util.Stack;

public class ExpressaoMatematica {

public static boolean validaExpressao(String expressao) {

Stack<Character> pilha = new Stack<>();

for (char caractere : expressao.toCharArray()) {

if (caractere == '(') {

pilha.push(caractere);

} else if (caractere == ')') {

if (pilha.isEmpty()) {

return false; // Se houver um ')' sem um '(', a expressão é inválida

}

pilha.pop(); // Remove o '(' correspondente

}

}

return pilha.isEmpty(); // A expressão é válida se a pilha estiver vazia no final

}

public static void main(String[] args) {

String expressao1 = "((3 + 2) \* (5 - 1))";

String expressao2 = "(3 + 2) \* (5 - 1))";

System.out.println("Expressão 1 válida: " + validaExpressao(expressao1));

System.out.println("Expressão 2 válida: " + validaExpressao(expressao2));

}

}

Exemplo 2: Navegação em Árvores (Pré-Ordem)

import java.util.Stack;

class No {

int valor;

No esquerda, direita;

public No(int valor) {

this.valor = valor;

esquerda = direita = null;

}

}

public class NavegacaoArvore {

public static void percorrePreOrdem(No raiz) {

if (raiz == null) {

return;

}

Stack<No> pilha = new Stack<>();

pilha.push(raiz);

while (!pilha.isEmpty()) {

No no = pilha.pop();

System.out.print(no.valor + " ");

if (no.direita != null) {

pilha.push(no.direita);

}

if (no.esquerda != null) {

pilha.push(no.esquerda);

}

}

}

public static void main(String[] args) {

No raiz = new No(1);

raiz.esquerda = new No(2);

raiz.direita = new No(3);

raiz.esquerda.esquerda = new No(4);

raiz.esquerda.direita = new No(5);

System.out.println("Percorrendo árvore em pré-ordem:");

percorrePreOrdem(raiz);

}

}

Esses são apenas dois exemplos simples de como usar pilhas em Java. Pilhas podem ser aplicadas em uma variedade de cenários, como os mencionados anteriormente, e a classe Stack é uma opção conveniente para implementá-las em seus programas Java.

Em Java, você tem várias alternativas para implementar pilhas, além da classe java.util.Stack. Na verdade, é recomendado usar outras estruturas de dados em vez de java.util.Stack, devido a algumas limitações e problemas de desempenho. Aqui estão algumas alternativas que você pode considerar:

Deque (java.util.Deque): Deque é uma interface que representa uma fila dupla (double-ended queue) que pode ser usada tanto como pilha (LIFO) quanto como fila (FIFO). Você pode implementá-la usando LinkedList ou ArrayDeque.

import java.util.Deque;

import java.util.LinkedList;

Deque<Integer> pilha = new LinkedList<>();

Com Deque, você pode usar métodos como push(), pop(), peek(), isEmpty(), entre outros, da mesma forma que faria com uma pilha.

ArrayDeque: ArrayDeque é uma implementação eficiente de Deque baseada em um array dinâmico. É geralmente mais eficiente que Stack e é preferível para a maioria dos casos.

import java.util.ArrayDeque;

ArrayDeque<Integer> pilha = new ArrayDeque<>();

LinkedList: Você pode usar LinkedList diretamente para implementar uma pilha.

import java.util.LinkedList;

LinkedList<Integer> pilha = new LinkedList<>();

LinkedList implementa a interface Deque, portanto, você pode usá-la como uma pilha.

Stack (evitando métodos depreciados): Se você deseja continuar usando Stack, é recomendável evitar os métodos depreciados, como push(), pop(), e peek(), e, em vez disso, usar os equivalentes em Deque. Isso ajudará a evitar problemas de desempenho e depreciação de métodos.

import java.util.Stack;

Stack<Integer> pilha = new Stack<>();

// Evite usar push(), pop() e peek() e use os equivalentes em Deque.

Em resumo, as alternativas em Java para implementar pilhas incluem o uso de Deque, ArrayDeque, LinkedList, ou até mesmo Stack se você evitar os métodos depreciados. A escolha depende das necessidades específicas do seu projeto e das operações que você planeja realizar na pilha.