N_EST DAD_A4 – Texto de Apoio

Índice

TAD PILHA

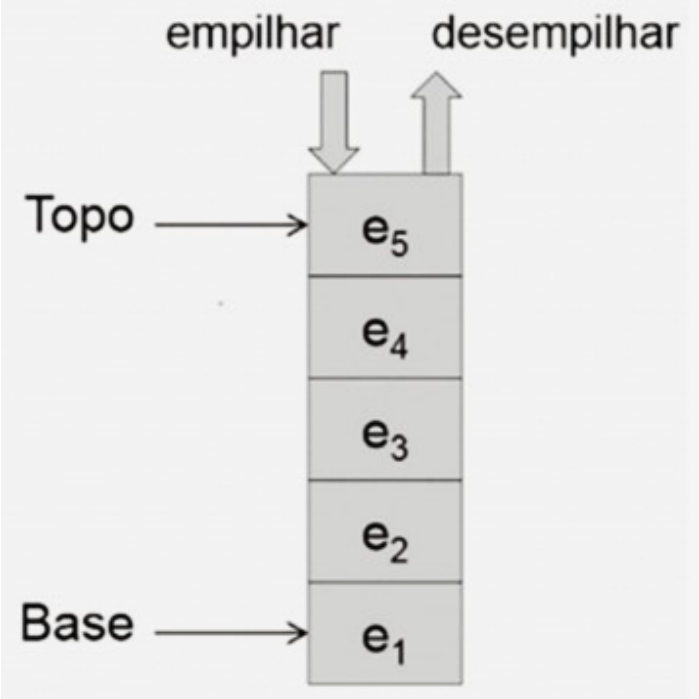
Conceitos

A pilha é uma Lista linear **com disciplina de acesso**: todas as inserções e remoções de elementos só podem ser feitas em apenas uma extremidade chamada **TOPO**. Os elementos são colocados uns sobre os outros e, assim, o elemento inserido menos recentemente fica no fundo da pilha e o mais recente está no topo. A esta regra, atribui-se o nome **LIFO** (Last in First Out).

No mundo real, podemos empilhar diversos tipos de objetos, como livros, CDs, roupas, cartas, moedas etc. Independentemente do objeto que está sendo empilhado, a disciplina de acesso é a mesma:

- Comumente, empilhamos o objeto no topo da pilha.
- Para retirar um objeto de algum ponto da pilha, precisamos retirar todos os objetos que estão acima do objeto a ser retirado.
- Sempre podemos ver, em detalhes, o conteúdo do último objeto empilhado observando o topo da pilha.

As únicas operações que podemos realizar em uma pilha são: a operação de inserção (ou empilhamento), que é chamada **push**, a remoção (desempilhamento), que é denominada **pop** e, ainda, é possível consultar o último objeto inserido pela operação **top**.



Fonte: CURY et al. (2018, p. 96).

SIMULANDO UMA PILHA

Simulemos algumas operações em uma pilha de números inteiros. Acompanhe a sequência abaixo:

Operação	Retorno	Status da Pilha
push (5)		5 **Topo**
push (3)		3 **Topo** 5
push (7)		7 **Topo** 3 5
pop ()		3 **Topo** 5
push (9)		9 **Topo** 3 5
pop ()		3 **Topo** 5

top ()	**3**	
pop ()		
pop ()		
pop ()	**Erro**	
isEmpty ()	**True**	

Repare que a operação pop() apenas remove (desempilha) um elemento da pilha. Em nossa abordagem, para ter acesso ao elemento do topo, devemos usar a operação top().

IMPLEMENTAÇÃO EM VETORES

Faremos a implementação da Pilha em um vetor. Implementaremos, inicialmente, uma pilha de dados do tipo String, da mesma forma que fizemos com a Lista Linear. Para essa implementação, utilizaremos a classe Vetor, a mesma utilizada na Aula 2 para Listas Lineares e que tem as seguintes operações:

```
public class Vetor {

	private String[ ] A;		// armazena os elementos do vetor
	private int capacity;		// capacidade do vetor
	private int size;		// elementos no vetor

	public Vetor(int capacity) {
		A = new int[capacity];
		this.size = 0;
		this.capacity = capacity;
	}

	public boolean isEmpty() {…}
	public int size() {…}
	public String get(int i) throws Exception {…}
	public void set(int i, String n) throws Exception {…}
	public void add(int i, String n) throws Exception {…}
	public void remove(int i) throws Exception {…}
	public int search(String n) {…}

}
```

Na verdade, não precisaremos de todos os métodos, porém, como a classe já está implementada, manteremos todos.

A classe Pilha estenderá a classe Vetor e, portanto, terá acesso aos métodos públicos dessa classe. É muito importante que você retome os conceitos de herança que estudou no semestre passado!

PASSO 1 – IMPLEMENTAR A CLASSE PILHA, DE FORMA A ESTENDER A CLASSE VETOR

Vejamos como fica a implementação dos métodos, uma vez que a classe vetor já está pronta. No áudio a seguir, você acompanhará uma explicação detalhada.


```java
public class Pilha extends Vetor {

    int top;      // indica em qual posição está o topo da pilha

    public Pilha(int capacity) {
    // Cria uma pilha com uma capacidade inicial
        super(capacity);
        top = -1;
    }

    public int size() {
        // Devolve o número de elementos da pilha
        return super.size();
    }

    public void push(String n) throws Exception {
        // Empilha, caso a pilha não esteja cheia, o elemento n
        top++;
        add(top, n);
    }
```

```
    public void pop() throws Exception {
        // Desempilha, caso a pilha não esteja vazia, o elemento do topo
        remove(top);
        top--;
    }

    public String top() throws Exception {
        // Devolve (não desempilha)o elemento do topo se a pilha não estiver vazia
        return get(top);
    }
}
```

PASSO 2 – IMPLEMENTAR A CLASSE DE TESTE

Vamos fazer uma pilha de Strings que representem nome de filmes.

```java
public class ProjPilha {

    public static void main(String[] args) {
        Pilha filmes = new Pilha(10);
        try {
            filmes.push("A vida é bela");
            filmes.push("Homem aranha");
            filmes.push("Harry Potter");
            System.out.println("Filme do topo: " + filmes.top());
            filmes.pop();
            filmes.push("Nasce uma estrela");
            System.out.println("Filme do topo: " + filmes.top());
            filmes.push("Bohemian Rhapsody");
            System.out.println("Filmes na pilha: " + filmes.size());
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

CLASSE STACK DO JAVA COLLECTIONS

O framework Java Collections implementa a estrutura de pilha por meio da classe Stack.

Observe que, conforme acabamos de implementar, a classe Stack também estende a classe Vector. Só é importante perceber que, diferentemente da nossa implementação, o método pop da classe Stack do Java Collections devolve, também, o elemento desempilhado.

Um objeto do tipo Stack<T> representa uma sequência de objetos (qualquer objeto) do tipo T armazenados em um vetor que aumentará seu tamanho, quando necessário, na inserção de novos elementos. O método de acesso aos dados é também LIFO (Last In First Out). Assim:

```
Stack<String> lista = new Stack<>();
```

Cria um `Stack` do tipo `String`. Se existisse um objeto chamado `Aluno` e se desejasse criar um `Stack` de objetos do tipo `Aluno`, a instrução ficaria:

```
Stack<Aluno> lista = new Stack<>();
```

Os principais métodos para manipulação de `Stacks` são:

- `nomeStack.empty()` – retorna `true` se a pilha estiver vazia ou `false`, caso contrário;
- `nomeStack.peek()` – retorna o objeto posicionado no topo da pilha. Ocorrerá um erro se a pilha estiver vazia;
- `nomeStack.pop()` – remove o objeto posicionado no topo da pilha, retornando-o. Ocorrerá um erro se a pilha estiver vazia;
- `nomeStack.push(objeto)` – empilha o objeto na pilha;
- `nomeStack.search(objeto)` – retorna a distância da primeira ocorrência do objeto informado como parâmetro em relação ao topo da pilha ou retorna -1 se o objeto não existir na pilha.

Exemplo:

```
public class Proj_Ex1 {

    public static void main(String[] args) {
        Stack<String> nomes = new Stack<>();
        nomes.push("Solange");
        nomes.push("Pedro");
        nomes.push("Álvaro");

        boolean teste = nomes.empty();
        if (teste) {
            System.out.println("A pilha está vazia!");
        } else {
            System.out.println("A pilha não está vazia!");
        }

        String topo = nomes.peek();
        System.out.println("O nome que esté no topo é " + topo);

        nomes.pop();
        topo = nomes.peek();
        System.out.println("O nome que esté no topo é " + topo);

        nomes.push("Cleide");
        nomes.push("Tania");
        int pos = nomes.search("Pedro");
        System.out.println("Pedro está  na " + pos + "a. posição a partir do topo");
        pos = nomes.search("Álvaro");
        if (pos == -1) {
            System.out.println("Álvaro não existe na pilha!");
        }
    }
}
```

REFERÊNCIAS

CURY, T. E. et al. *Estrutura de Dados*. Porto Alegre: SAGAH, 2018.