



Estrutura de Dados

Prof. Dr. Bruno Aguilar da Cunha
bruno.cunha@facens.br

FUNÇÕES

Em Java, os módulos, sejam eles procedimentos ou funções, são representados pelos métodos. Para escrever um método em Java, representando procedimentos, utiliza-se a seguinte sintaxe:

```
static void nome_Método (<argumentos>)  
{  
    <instruções>;  
}
```

MODIFICADORES DE ACESSO

static: é um modificador que indica que o método será alocado em memória sem que haja necessidade de ser instanciado. Não necessita objeto; pode ser invocado com base no nome da classe.

public: pode ser invocado livremente e indica um método que é visível para qualquer um que “enxergue” a classe.

MODIFICADORES DE ACESSO

protected: pode ser utilizado apenas no mesmo pacote e em subclasses;

private: pode ser invocado apenas na classe;

final: não pode ser sobrescrito e equivale à declaração de constante.

FUNÇÕES

void: indica que não será retornado nenhum valor do programa que realizou a chamada;

<nome_Método>: é um identificador válido da linguagem, obedece às mesmas regras que os identificadores de classe, objeto e variável;

<argumentos>: indica a lista de argumentos que serão passados como parâmetros para o método. A sintaxe dos argumentos é a mesma da declaração de variáveis: **tipo_dado identificador**, e os vários parâmetros são separados por vírgulas.

EXEMPLO FUNÇÃO – SEM PARÂMETROS

```
static void modAdicao( ){  
    double v1;  
    double v2;  
    double res;  
    v1 = Double.parseDouble(JOptionPane.showInputDialog(  
        "Digite o primeiro valor"));  
    v2 = Double.parseDouble(JOptionPane.showInputDialog(  
        "Digite o segundo valor"));  
    res = v1 + v2;  
    JOptionPane.showMessageDialog(  
        null, "Soma = " + res);  
}
```

EXEMPLO FUNÇÃO – COM PARÂMETROS

```
static void modAdicao(double v1, v2) {  
    double res;  
    res = v1 + v2;  
    JOptionPane.showMessageDialog(  
        null, "Soma = " + res);  
}
```

CHAMANDO E USANDO FUNÇÕES

```
import javax.swing.JOptionPane;
public class Menu {
    public static void main (String args[]){
        int opcao;
        opcao = Integer.parseInt(JOptionPane.showInputDialog(
            "Escolha a sua opção:\n" +
            "1 - Adição\n" +
            "2 - Subtração\n" +
            "3 - Multiplicação\n" +
            "4 - Divisão"));
        switch (opcao){
            case 1 : modAdicao(); break;
            case 2 : modSubtr(); break;
            case 3 : modMultipl(); break;
            case 4 : modDiv();break;
            default : JOptionPane.showMessageDialog(
                null, "Fim do Programa");
        }
    }
}
```


FUNÇÕES - EXERCÍCIOS


- 1) Complete o exemplo anterior criando e utilizando as funções `modSubtr()`, `modMultipl()`, `modDiv()`. As funções foram criadas sem parâmetros. Faça testes e compreenda o funcionamento.
- 2) Altere o programa e inclua o uso de parâmetros em todas as funções. Faça testes.

PILHAS



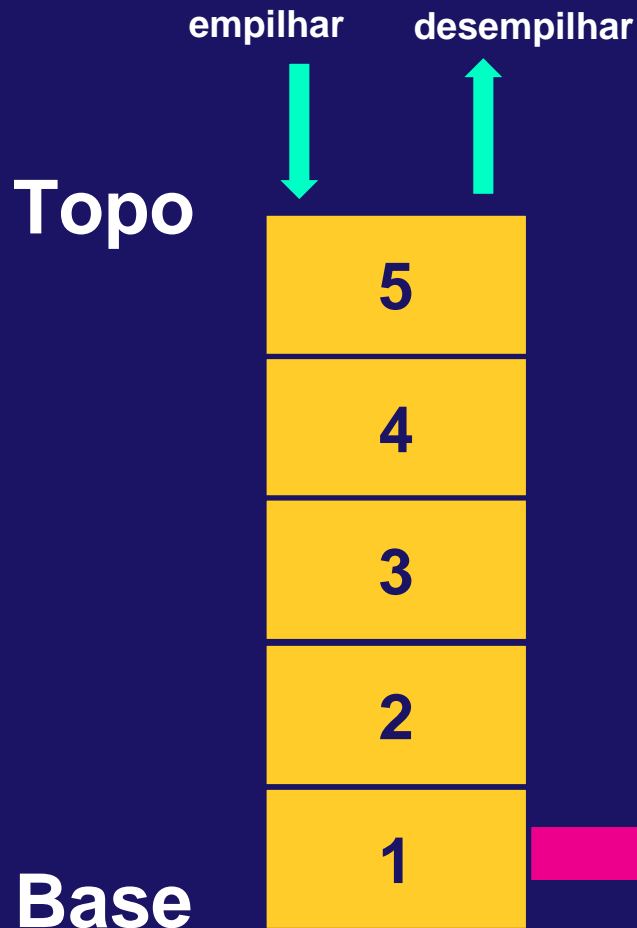
PILHA é uma lista em que todas as operações de inserção, remoção e acesso são feitas num mesmo extremo, denominado topo.

Quanto um item é inserido numa pilha, ele é colocado em seu topo e, em qualquer instante, apenas o item no topo da pilha pode ser removido. Devido a essa política de acesso, os itens são removidos da pilha na ordem inversa àquela em que foram inseridos, ou seja, **o último a entrar é o primeiro a sair.** **Por isso, pilhas são também denominadas listas LIFO (Last-In/First-Out).**



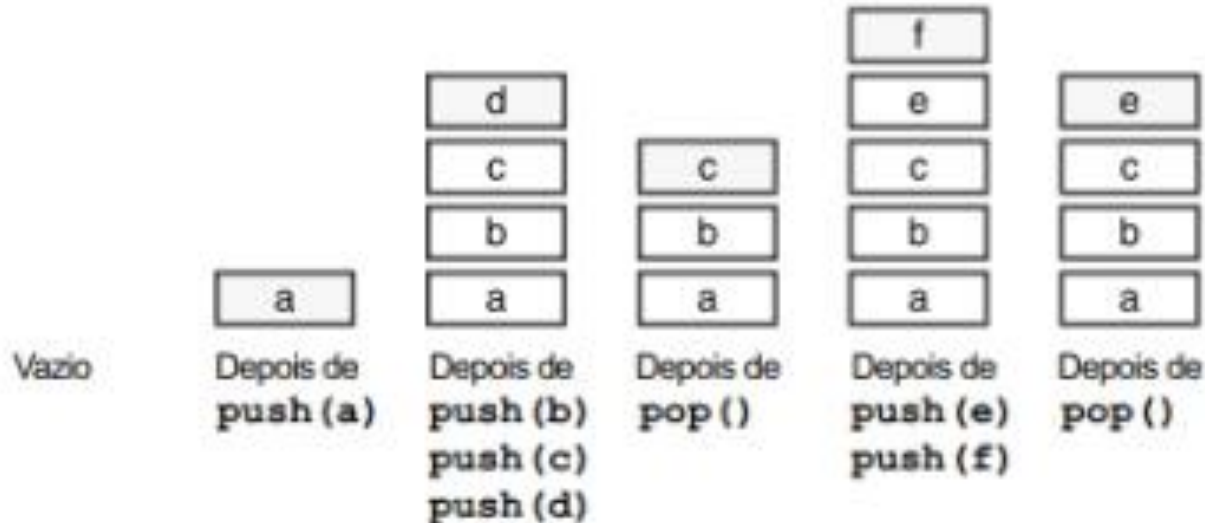
PILHAS

Na implementação de uma pilha, em apenas uma das extremidades, que chamamos de topo, é possível realizar a manipulação dos dados. A outra extremidade da estrutura chamamos de base.



PILHAS

Outros exemplos cotidianos de pilhas incluem pratos e tigelas em um armário de cozinha ou CDs em um pino. Embora você continuamente adicione mais papéis ao topo das pilhas na sua escrivaninha, essas pilhas não se qualificam porque muitas vezes você precisa remover um papel perdido do meio. Com uma pilha genuína, o item obtido a seguir é sempre aquele adicionado mais recentemente.



PILHAS

A principal propriedade de uma pilha é a sua capacidade de inverter a ordem de uma sequência. Essa propriedade é útil em varias aplicações em computação.


Por exemplo, num navegador web, conforme as páginas vão sendo acessadas, seus endereços vão sendo inseridos numa pilha. Em qualquer instante durante a navegação, o endereço da última página acessada está no topo da pilha. Quando o botão voltar é clicado, o navegador remove um endereço da pilha e recarrega a página correspondente.

PILHAS



A medida que o botão voltar é clicado, o navegador remove um endereço da pilha e recarrega a página correspondente. Sendo assim, as páginas são representadas na ordem inversa àquela em que foram visitadas.


O controle do fluxo de execução de um programa é outro exemplo interessante de uso. Durante a execução, sempre que uma função é chamada, antes de passar o controle a ela, um endereço de retorno correspondente é inserido em uma pilha.



PILHAS



Quando a função termina sua execução, o endereço no topo da pilha é removido e a execução continua a partir dele. Assim, a última função que passa o controle é a primeira a recebe-lo de volta.



PILHAS

- Convertendo expressões infixas na forma pós-fixa e avaliando expressões pós-fixas. O operador em uma expressão infixa, como $3 + 4$, aparece entre seus dois operandos, enquanto o operador em uma expressão pós-fixa, como $3\ 4\ +$, segue seus dois operandos.
- Algoritmos de backtracking (retrocesso), (que ocorrem em problemas como prova automatizada de teoremas e nos jogos).

PILHAS


- Gerenciando a memória do computador para suportar chamadas de função e método.
- Dando suporte ao recurso de desfazer em editores de texto, processadores de texto, programas de planilha, programas de desenho e aplicativos semelhantes.
- Mantendo um histórico de endereços visitados por um navegador da web.

PILHAS em JAVA



Um tipo de pilha não é integrado ao JAVA.

Se necessário, os programadores Java podem usar um ARRAY para emular uma pilha. Basta você considerar o final de um array como o topo de uma pilha e a primeira posição/local do array como a base.



OPERAÇÕES EM PILHAS

As funções associadas à pilha são:

- **Pilha(int tam)** cria uma nova pilha vazia e recebe como parâmetro um valor inteiro que representa a quantidade de itens a serem adicionados na estrutura.
- **empilhar (item)** insere um novo item na pilha. A operação necessita do item e não retorna coisa alguma.
- **desempilhar()** remove o item que está no topo da pilha. Não necessita parâmetros e retorna o item removido. A pilha é modificada.
- **vazia()** testa se a pilha está vazia. Não necessita parâmetros e retorna um valor booleano; valor do tipo **boolean**, **True** or **False**.
- **cheia()** testa se a pilha está cheia. Não necessita parâmetros e retorna um valor booleano; valor do tipo **boolean**, **True** or **False**.
- **exibePilha()** imprime na tela/console os itens existentes na pilha. Não necessita parâmetros e caso a pilha esteja vazia, será fornecido uma mensagem de aviso.

Uso de classes e métodos (POO)

Como em qualquer linguagem de programação orientada a objetos, a implementação de um tipo abstrato de dados como uma pilha é feita através da criação de uma nova classe. As operações sobre uma pilha são implementadas como métodos. Além disso, para implementar uma pilha, que é uma coleção de elementos, faz sentido utilizar o poder e simplicidade das coleções primitivas fornecidas pelo Python. Nós usaremos um array (objects).

A classe Pilha()

parte 1

Lembre-se que quando executarmos o código nada acontece além da definição da classe. Nós devemos criar um objeto **Pilha** e então usá-lo.

```
public class Pilha {
    int tamanho;
    int topo;
    Object vetor[];

    Pilha(int tam) {
        topo = -1;
        tamanho = tam;
        vetor = new Object[tam];
    }

    public boolean vazia () {
        if (topo == -1)
            return true;
        else
            return false;
    }

    public boolean cheia () {
        if (topo == tamanho - 1)
            return true;
        else
            return false;
    }

    public void empilhar(Object elem) {
        if (cheia() == false) {
            topo++;
            vetor[topo] = elem;
        }
        else {
            System.out.println("Pilha Cheia");
        }
    }
}
```

A classe Pilha()

parte 2

Lembre-se que quando executarmos o código nada acontece além da definição da classe. Nós devemos criar um objeto **Pilha** e então usá-lo.

```
public Object desempilhar(){
    Object valorDesempilhado;
    if (vazia() == true)
        valorDesempilhado = "Pilha Vazia";
    else{
        valorDesempilhado = vetor[topo];
        topo--;
    }
    return valorDesempilhado;
}

public void exhibePilha(){
    if (vazia() == true)
        System.out.println("Pilha Vazia");
    else{
        for(int i = topo; i >= 0; i--) {
            System.out.println("Elemento "
                               + vetor[i] + " posicao" + i);
        }
    }
}
```

Uso da classe Pilha() na Função main()

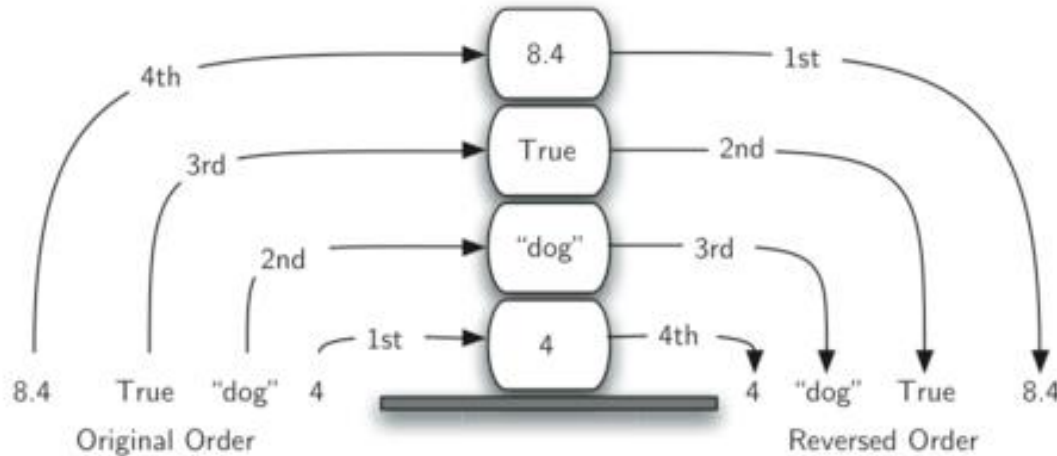
É preciso instanciar/criar um objeto que irá herdar todos os métodos definidos na classe Pilha(). Com os métodos implementados, podem adicionar e remover valores da Pilha e também consultar se está vazia ou não e também imprimir o seu conteúdo.

```
Pilha P = new Pilha(5);
Object valor;
int i = 0;
int opcao = 1;

while (opcao != 4) {
    opcao = Integer.parseInt
        (JOptionPane.showInputDialog(null,
            "Escolha uma Opção: \n" +
                "1-Inserir um elemento na fila \n" +
                "2-Excluir elemento da fila \n" +
                "3-Exibir elementos da fila\n" +
                "4-Sair" + "\n"));
    switch (opcao) {
        case 1 :
            valor = JOptionPane.showInputDialog
                (null,
                    ""Empilhar elemento
                    Digite um valor""");
            P.empilhar(valor);
            break;
        case 2 :
            System.out.println
                ("Elemento desempilhado " +
                    P.desempilhar());
            break;
        case 3 :
            P.exibePilha();
            break;
        default: JOptionPane.showMessageDialog(null,"Sair");
    }
}
```

Reversão em Pilhas

A ordem que eles são removidos é exatamente a inversa da ordem em que foram colocados. As pilhas são fundamentalmente importantes, pois elas podem ser usadas para reverter a ordem dos itens.



PILHA - EXERCÍCIOS

1) Escreva um programa em JAVA que leia números inteiros e armazene em uma pilha. A entrada de dados deve ser interrompida quando o usuário informar o número zero e/ou esgotar a quantidade definida de elementos a serem armazenados na estrutura. Por último, imprima os elementos na ordem em que foram removidos da pilha.

PILHA - EXERCÍCIOS

2) Construa um programa em JAVA utilizando uma pilha que resolva o seguinte problema:

Armazene as placas dos carros que estão estacionados numa rua sem saída estreita. Dado uma placa verifique se o carro está estacionado na rua. Caso esteja, informe a sequência de carros que deverá ser retirada para que o carro em questão consiga sair.

PILHA - EXERCÍCIOS

3) Faça uma adaptação em seu programa que manipula pilhas para que possa exibir o número de elementos da pilha que possuem valor par.

4) Escreva um programa em JAVA que cria 2 pilhas (N e P) e solicita ao usuário para informar números inteiros para preencher um array. Para cada valor do array:

se positivo, inserir na pilha P;

se negativo, inserir na pilha N;

se zero, retirar um elemento de cada pilha.

O array de números inteiros deve ter 8 elementos.

PILHA - EXERCÍCIOS

5) Construa um programa que solicite ao usuário que digite a url de um site que será acessado. Ao receber uma nova URL, armazene na pilha. Possibilite ao usuário, resgatar as URLs acessadas anteriormente na ordem de visitaç o ao solicitar “Voltar”  s p ginas anteriores.

REFERÊNCIA BIBLIOGRÁFICA

SLIDES DE AULA

Lógica de Programação e estrutura de dados com aplicações em JAVA
Sandra Puga e Gerson Rissetti
2ª. Edição

Dúvidas?

Dúvidas fora do horário de aula:
bruno.cunha@facens.br