Pilhas: Fundamentos e implementação da estrutura em Java

Veja neste artigo os fundamentos da estrutura de dados pilha (LIFO), bem como a implementação de uma pilha simples em Java.

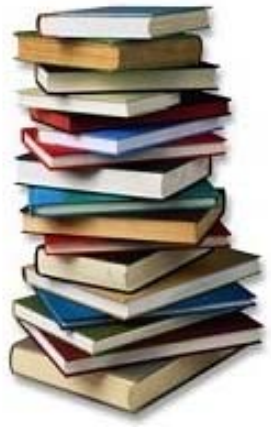
Uma questão importante no estudo da computação é o entendimento das estruturas de dados, dentre as quais temos filas, pilhas, listas ligadas, entre outras. Vamos entender aqui o funcionamento das pilhas e como implementar uma pilha simples em Java.

O que é uma estrutura de dados?

A estrutura de dados é o coração de diversos programas bem elaborados, saber qual tipo de estrutura utilizar é essencial para construir um aplicativo de qualidade. A estrutura de dados é na verdade a forma de organizar e armazenar informações para que estas possam posteriormente ser utilizadas de modo eficiente.

O que é uma pilha?

A pilha é uma das estruturas de dados e trabalha com o formato LIFO (o último a entrar é o primeiro a sair, “Last In, First Out”, em inglês). Lembre-se da pilha como uma pilha de livros, em que o primeiro livro que foi inserido na pilha, normalmente é o último que sai dela, enquanto o último adicionado é o primeiro a ser retirado.



A estrutura da pilha, segundo Farias “são estruturas de dados do tipo LIFO (last-in first-out), onde o último elemento a ser inserido, será o primeiro a ser retirado. Assim, uma pilha permite acesso a apenas um item de dados - o último inserido. Para processar o penúltimo item inserido, deve-se remover o último”.

A pilha é considerada uma estrutura de dados simples, sendo fácil de implementar. 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.

Um outro caso de pilha simples de se entender é o caso do entregador de pizza.

Mudar a ordem: a primeira pizza no baú deve ser a última a ser entregue e a última pizza do baú, a primeira a ser entregue. Neste caso, ao chegar na casa do cliente, o entregador apenas pega a primeira pizza que está no baú e entrega ao cliente.

Este exemplo foi proposto inicialmente por Takai, apresentando uma rota de entrega de quatro pizzas, sendo a seguinte ordem:

1° entrega: Portuguesa
2° entrega: Frango com catupiry
3° entrega: Calabresa
4º entrega: Quatro queijos

Assim, para armazenar no baú, a ordem deve ser invertida, ficando da seguinte forma:

Portuguesa (topo do baú)
Frango com catupiry
Calabresa
Quatro queijos

Para as tarefas temos: a criação da pilha, o empilhamento - push (ato de colocar uma caixa de pizza sobre a outra), o ato de desempilhar - pop (na hora que o entregador tira a caixa de pilha para entregar ao cliente), além de uma verificação se a pilha está cheia ou vazia (ato que o entregador faz ao verificar o baú).

Implementando nossa pilha em Java

Primeiramente, vamos criar um array para armazenar nossa pilha. Imaginando o baú, temos uma capacidade de 10 pizzas, por exemplo, já um caminhão, dependendo do tamanho dele e da quantidade de produtos a serem entregues, pode armazenar de 1 a centenas de entregas, e assim por diante. Em nosso caso, primeiro vamos criar o array, depois vamos definir um tamanho de teste para 10 itens em nossa pilha.

Listagem 1: Declarando o array

```
public Object[] pilha;
```

Foi utilizado o tipo Object para armazenar textos ou números, uma forma mais genérica, apenas como teste de pilha.

Listagem 2: Variável para armazenar a posição atual na pilha

```
public int posicaoPilha;
```

Aqui foi criado uma variável que exibe a posição atual na pilha, se a posição for 10, então chegamos ao topo da pilha e não poderão ser inseridos mais itens.

Agora no construtor será definido o tamanho da pilha e inicializado a posição.

Listagem 3: Inicializando a pilha

```
public Pilha() {
        this. posicaoPilha = -1;
// indica que esta nula, vazia, pois a posição zero
```

Agora vamos criar uma função que verifique se a pilha está vazia (isEmpty).

Listagem 4: Função para verificar se a pilha está vazia

```
public boolean pilhaVazia() {
        if (this. posicaoPilha == -1) {
            return true;
        // Verifica que o o atributo posicaoPilha é igual a -1,
        //se for, a pilha é nula, ou seja, ainda esta vazia,
        //retornando verdadeiro.
        }
        return false;
}
```

Agora é necessário verificarmos qual o tamanho da pilha atual, ou seja, o entregador olha quantas pizzas ainda tem dentro do baú para entregar.

Listagem 5: Função que retorna a quantidade de itens na pilha

```
public int tamanho() {
        if (this.pilhaVazia()) {
            return 0; // aqui indica que não tem nenhum conteúdo dentro da pilha
        }
        return this. posicaoPilha + 1;
        // aqui indica quantos itens tem dentro da pilha, somando-se 1,
        //pois a pilha inicia no zero. Logo, se tivermos 3 itens na pilha,
        // o atributo posicaoPilha vai exibir 2.
        //Para sabermos quantos itens há, precisamos somar um.
}
```

Agora é necessário um método para empilhar itens dentro da pilha, ou seja, a forma que o entregador pega a pizza e insere sobre a última pizza que está no baú.

Listagem 6: Função para empilhar itens

```
public void empilhar(Object valor) {
        // push
        if (this. posicaoPilha < this.pilha.length - 1) {
        // verifica se a posicaoPilha é menor do que o tamanho da pilha,
        //se for, então é inserido o valor na pilha e ao mesmo tempo é feito
        //o incremento do atributo posicaoPilha
            this.pilha[++posicaoPilha] = valor;
        }
}
```

Depois de criada uma forma de empilhar, temos de desempilhar, ou seja, hora do entregador retirar a pizza do baú e entregar ao cliente.

Listagem 7: Função para remover itens da lista

```
public Object desempilhar() {
        //pop
        if (pilhaVazia()) {
            return null;
        // primeiro verificamos se a pilha esta vazia,
        //se estiver, nada será realizado
        }
        return this.pilha[this. posicaoPilha --];
        // aqui retorna o que tem na última posição da pilha
        //e decrementa o atributo posicaoPilha
}
```

Agora vamos ver o resultado da pilha completa em funcionamento:

```
    public Object[] pilha;
    public int posicaoPilha;

    public Pilha() {
        this.posicaoPilha = -1;
// indica que esta nula, vazia, pois posição um indica que contém informação
        this.pilha = new Object[1000];
// criando uma pilha com 1000 posições
    }

    public boolean pilhaVazia() {
        //isEmpty
        if (this.posicaoPilha == -1) {
            return true;
        }
        return false;
    }

    public int tamanho() {
        //is
        if (this.pilhaVazia()) {
            return 0;
        }
        return this.posicaoPilha + 1;
    }

    public Object exibeUltimoValor() {
        //top
        if (this.pilhaVazia()) {
            return null;
        }
        return this.pilha[this.posicaoPilha];
    }

    public Object desempilhar() {
        //pop
        if (pilhaVazia()) {
            return null;
        }
        return this.pilha[this.posicaoPilha--];
    }

    public void empilhar(Object valor) {
        // push
        if (this.posicaoPilha < this.pilha.length - 1) {
            this.pilha[++posicaoPilha] = valor;
        }
    }

    public static void main(String args[]) {
        Pilha p = new Pilha();
        p.empilhar("Portuguesa ");
        p.empilhar("Frango com catupiry ");
        p.empilhar("Calabresa ");
        p.empilhar("Quatro queijos ");
        p.empilhar(10);
                while (p.pilhaVazia() == false) {
            System.out.println(p.desempilhar());
        }
    }
}
```

Pronto, a pilha está funcionando. Nosso entregador pode empilhar e desempilhar as pilhas sem maiores problemas.

Conclusão

por **Fabio Gomes** (10) (0)

Ficou com alguma dúvida?