Pilhas

Nas aulas passadas aprendemos sobre *listas ligadas* e *duplamente ligadas*. tais listas possuíam células que apontavam para outras, anteriores e posteriores. Vimos nos exercícios que o Java já tem tudo isso implementado por meio da Classe *LinkedList*.

Neste capítulo veremos uma outra estrutura de dados, cuja principal diferença em relação aos outros tipos de estruturas de dados, é guardar os diversos estados de uma aplicação para que no futuro, se necessário, seja possível voltar a estes estados. A essa estrutura damos o nome de ***Pilha***.

Vamos criar um pacote e, dentro dele, a Classe "Pilha". As operações que teremos nessa pilha são:

```
package ed.pilha

public class Pilha {

    public void insere(String nome) {

    }

    public String remove() {
        return "";
    }

    public boolean vazia() {
        return false;
    }
}
```

A *Pilha* segue a regra de inserção de elementos um após o outro e a remoção funciona da mesma forma, do último para o primeiro elemento.

Para começar a implementar, não começemos do zero. Já temos uma parte do código feita, pois a fizemos nas aulas de listas. Vamos utilizar a implementação que o Java nos oferece.

```
package ed.pilha

import java.util.LinkedList;
import java.util.List;

public class Pilha {

private List<String> nomes = new LinkedList<String>();

...
```

Vamos criar um documento para teste para daí começar a implementar os métodos:

```
package ed.pilha

public class TesteDaPilha {

    public static void main(String[] args) {
        Pilha pilha = new Pilha();
    }
}
```

O que já devemos implementar também é o *toString*:

```
@Override
public String toString() {
    return nomes.toString();
}
```

Método *insere*

Implementar o método utilizando o conceito de *Pilha* é bem fácil, pois sempre seguiremos uma ordem. Então o método `insere` ficará assim:

```
public void insere(String nome) {
    nomes.add(nome);
}
```

Testando:

```
pilha.insere("Mauricio");
System.out.println(pilha);

pilha.insere("Guilherme");
System.out.println(pilha);
```

O que retorna:

```
[Mauricio]
[Mauricio, Guilherme]
```

Método *remove*

Aqui basta chamar o "remove" do *LinkedList* passando o elemento na casa `nomes.size()-1`:

```
public String remove() {
    return nomes.remove(nomes.size()-1);
}
```

para testar vamos pedir para imprimir cada elemento que será removido e, depois, a lista final:

```
String r1 = pilha.remove();
System.out.println(r1);

String r2 = pilha.remove();
System.out.println(r2);

System.out.println(pilha);
```

O que nos retorna:

```
Guilherme
Mauricio
[]
```

De fato, os elementos foram removidos começando do final da lista.

Método *vazia*

Este método indica se a lista está vazia ou não. Temos duas maneiras de implemetá-lo:

```
public boolean vazia() {
    return nomes.size() == 0;
}
```

Ou usando a função do *LinkedList*:

```
public boolean vazia() {
    return nomes.isEmpty();
}
```

Para testar, vamos imprimir o comando booleano `System.out.println(pilha.vazia())` antes e depois de inserir elementos na lista. Veremos que retornará

```
true
false
```

De fato, antes a lista estava vazia e, após inserirmos os elementos, ela não estará mais.

O Java também já possui uma Classe própria para *pilhas*, cujo nome é **Stack**. Substituindo os nomes de nossos métodos para os da Classe do Java, temos:

- insere -> *push*
- remove -> *pop*

Podemos escrever no arquivo de teste:

```
Stack<String> stack = new Stack<String>();
stack.push("Mauricio");
stack.push("Marcelo");

System.out.println(stack);
```

O quê imprime `[Mauricio, Marcelo]`. E para remover:

```
stack.pop();
System.out.println(stack)
```

O quê imprime `[mauricio]`.

Método *peek*

Como vimos, o *pop* remove o último elemento da pilha. O método *peek* trabalha em cima desse elemento também, porém sem removê-lo. Ele apenas o retorna. Se temos, então, a pilha `[Mauricio, Marcelo]`,

```
String nome = stack.peek();
System.out.println(nome);
```

nos retorna `Marcelo`.

Usabilidade das *pilhas*

O conceito de *pilhas* é muito utilizado por compiladores e autômatos. Esta estrutura de dados tem muita usabilidade em ciências da computação. O próprio, e muito conhecido, comando "Desfazer" dos editores de texto, de código, de imagens, etc tem como base as *pilhas*. Podemos também brincar com palavras e inverter a ordem de suas letras utilizando as pilhas.