02

Conjuntos

Capítulo 7 - Conjuntos

Nesta aula veremos um outro tipo de estrutura de dados, o ***Conjunto***.

Se pensarmos em um conjunto da disciplina de matemática, verificamos que uma de suas vantagens é que não há elementos repetidos. Sabemos também que é possível fazer operações com conjuntos, como intersecção, união e saber quais elementos estão contidos. A ideia para essa estrutura é essa.

Para criarmos um conjunto, precisaremos inserir elementos - não repetidos, algo que já vimos com o *LinkedList*. Desta vez iremos começar criando a Classe de Teste para o Conjunto:

```
package ed.conjunto;

import java.util.LinkedList;

public class TesteDeConjunto {

    public static void main(String[] args) {

        LinkedList<String> conjunto = new LinkedList<String>();

    }
}
```

Vamos começar a inserir elementos não repetidos em um conjunto:

```
public static void main(String[] args) {

    LinkedList<String> conjunto = new LinkedList<String>();

    if(!conjunto.contains("Mauricio")) {
        conjunto.add("Mauricio");
    }
}
```

O problema é que essa condição estará dentro de um método "adiciona". Toda vez que chamarmos um *add*, chamaremos um *contains*. O tempo de execução desse algoritmo é muito lento, pois o *contains* varre todo o *array* e a operação de adição, que antes ocorria em tempo constante, agora terá tempo linear.

Listas de listas

Vamos criar a Classe "Conjunto" e pensar em uma maneira de resolver tal problema. Mas antes pensemos na seguinte situação:

> Você entra em um supermercado grande e precisa encontrar um sorvete de chocolate. Para isso, não percorre o lugar inteiro, passando por todos os corredores, até encontrar o item. Nós sabemos que o sorvete fica na seção de congelados,

então você vai direto para lá.

Esta será nossa estratégia. Teremos diversos *LinkedLists*, um para cada "seção" do *array*. Mas como organizar essas listas? Para *Strings*, uma boa prática é pela letra inicial que, nesse caso, dividiria o *array* em 26 pedaços. No final, o que teremos que fazer serão *LinkedLists* de *LinkedLists*, ou seja, listas de listas.

```
package ed.conjunto

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

public class Conjunto {

    private LinkedList<LinkedList<String>> tabela = new LinkedList<LinkedList<String>>();

}
```

Vamos criar as 26 listas por meio de um Construtor:

```
public class Conjunto {

    private LinkedList<LinkedList<String>> tabela = new LinkedList<LinkedList<String>>();

    public Conjunto() {
        for(int i = 0; i < 26; i++) {
            tabela.add(new LinkedList<String>());
        }
    }

}
```

Agora temos 26 listas, uma para cada letra do alfabeto.

Método *calculaIndice...*

Vamos criar um método que nos auxiliará a calcular o índice da lista maior, que vai de 0 a 25.

```
private int calculaIndiceDaTabela(String palavra) {
    return palavra.toLowerCase().charAt(0) % 26;
}
```

Dada uma palavra, conseguimos, assim, calcular o lugar em que ela deve ficar por meio de sua primeira letra.

Tal método é muito importante pois é ele que faz o espalhamento dos dados e retorna sempre o mesmo índice para o mesmo conjunto específico de elementos, que possuam , no caso, a mesma letra inicial.

Métodos *adiciona* e *contem*

O método anterior irá nos ajudar a implementar o "adiciona":

```
public void adiciona(String palavra) {

    int indice = calculaIndiceDaTabela(palavra);
    List<String> lista = tabela.get(indice);
    lista.add(palavra);
}
```

Só falta garantir que o elemento não exista na lista. Para isso criamos o método "contem":

```
private boolean contem(String palavra) {
    int indice = calculaIndiceDaTabela(palavra);
    return tabela.get(indice).contains(palavra);
}
```

No "adiciona":

```
public void adiciona(String palavra) {
    if(!contem(palavra)) {
        int indice = calculaIndiceDaTabela(palavra);
        List<String> lista = tabela.get(indice);
        lista.add(palavra);
    }
}
```

Vamos implemetar o *toString* e testar:

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

Testando a implementação

Para testar fazemos na Classe "Teste DeConjunto":

```
package ed.conjunto;

import java.util.LinkedList;

public class TesteDeConjunto {

    public static void main(String[] args) {

        Conjunto conjunto = new Conjutno();
        conjunto.adiciona("Mauricio");
        System.out.println(conjunto);

        conjunto.adiciona("Mauricio");
        System.out.println(conjunto);
```

```
    }
}
```

Estaremos adicionando a palavra "Mauricio" duas vezes para testar se ela se repetirá no *array* ou não. Rodando o programa teremos:

```
[[], [], ..., [Mauricio], [], [],...]
[[], [], ..., [Mauricio], [], [],...]
```

A palavra foi adicionada apenas uma vez como queríamos. Veremos se palavras com letra inicial igual e diferente caem na mesma e em outra casa, respectivamente:

```
conjunto.adiciona("Mauricio");
System.out.println(conjunto);

conjunto.adiciona("Mauricio");
System.out.println(conjunto);

conjunto.adiciona("Marcelo");
conjunto.adiciona("Guilherme");
System.out.println(conjunto);
```

O que nos retorna:

```
[[], [], ..., [Mauricio], [], [],...]
[[], [], ..., [Mauricio], [], [],...]
[[], [], ..., [Mauricio, Marcelo], [], [],..., [Guilherme]]
```

A estrutura está funcionando bem e rápido.

Usando o *ArrayList*

Nós utilizamos o *LinkedList* para implementar nosso código, porém conseguimos melhorá-lo ainda mais se mudarmos para o *ArrayList*, pois sua função que pega um elemento aleatório de uma lista é muito mais rápida. Então façamos essa mudança:

```
private ArrayList<LinkedList<String>> tabela = new ArrayList<LinkedList<String>>();
```

Teremos, então, aqui um *ArrayList* de *LinkedLists*.

Método *remove*

Esse método precisa ter a condição do elemento estar na lista.

```
public void remove(String palavra) {
    if(contem(palavra)) {
        int indice = calculaIndiceDaTabela(palavra);
        List<String> lista = tabela.get(indice);
        lista.remove(palavra);
```

```
    }
}
```

Vamos testar esse método utilizando as palavras adicionadas anteriormente e faremos:

```
conjunto.remove("Mauricio");
System.out.println(conjunto);
```

O que retorna:

```
[[], [], ..., [Mauricio], [], [],...]
[[], [], ..., [Mauricio], [], [],...]
[[], [], ..., [Mauricio, Marcelo], [], [],..., [Guilherme]]
[[], [], ..., [Marcelo], [], [],..., [Guilherme]]
```

De fato, o elemento foi excluído da lista.

Set

À estrutura de Conjuntos damos o nome de ***Set*** e sua implementação é o *HashSet*:

```
Set<String> conjuntoDoJava = new HashSet<String>();

conjuntoDoJava.add("Mauricio")

...
```

Para espalhar os dados, o *HashSet* se utiliza do método *HashCode*, o qual a Classe *Object* do Java possui:

```
String x = "Guilherme";
x.hashCode();
```