Conjunto

Conjuntos são **estruturas de dados em que os elementos não podem se repetir**. Ué, mas *"aí é só colocar um **if** antes de adicionar um elemento pra saber se ele existe em alguma list da vida e tá resolvido"*:

```java
List<String> lista = new LinkedList<>();

if(!lista.contains("Roberto")){
    lista.add("Roberto");
}
```

Só que não né mano, s**e você tiver uma lista com 100 mil elementos?** Vai ficar *percorrendo-a inteira toda vez que for fazer uma adição?* É aí que tá o pulo do gato do conjunto.

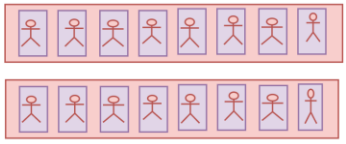
Como funciona um conjunto

A **chave da performance de um conjunto** é a maneira que ele **espalha** os seus **agrupamentos**. Meio abstrato, mas nem tanto.

Uma **organização** que a gente pode fazer pra **espalhar** os **conjuntos** é usar uma **Lista Ligada** dentro de um **Vetor**, isso vai formar meio que uma **tabela(vetor)** composta por **conjuntos (lista ligada)**:

```java
public class Conjunto {

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



Espalhando os conjuntos

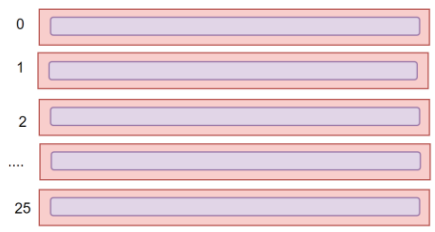
Esse **critério de espalhar os conjuntos é sem dúvida a parte mais importante**, é preciso pensar em como os conjuntos serão espalhados *(java usa um hashcode por padrão).*

Nesse caso vamos ter conjuntos formados de maneira alfabética, ou seja, as palavras serão agrupadas pelas suas letras iniciais. Sabendo **disso minha tabela vai ter 26 posições** (letras do alfabeto) e cada **posição** vai ter seu próprio **conjunto**:

```java
public class Conjunto {

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

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



Mas **como saber onde está cada conjunto?** Nesse caso em específico nossa lógica se baseia em um algoritmo já existente.

A gente precisa pegar a **primeira letra da palavra que for inserida no conjunto** e fazer um **módulo dela por 26** *(a gente pega o codepoint da letra que é um número, dividindo os números das possíveis*

letras por 26 e pegando o resto a gente pode ter variações que vão de 0 até 25, o mesmo tanto de indices da tabela):

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

Inserir e remover dos conjuntos

Sabendo onde cada conjunto está na nossa tabela, podemos **simplesmente inserir (fazendo a lógica para não inserir valores repetidos)** e **remover do conjunto específico**:

```java
public void adiciona(String palavra){

    if(!contem(palavra)){

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

}
```

```java
public void remove(String palavra){

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

}
```

A **grande chave do conjunto** é simplesmente **espalhar os agrupamentos baseado em algum identificador lógico para eles** e então **usar esse identificador para ir diretamente no agrupamento** que ele precisa procurar/inserir/remover um elemento enquanto garante que ele é único de uma maneira mais performática:

```
[[], [], [], [], [Lauro, Lucas], [], [], [], [], [], [], [], [], [], [], [], [], [], [Ana], [], [], [], [], [], []]
```