Collections Framework

Em java existe uma api que encapsula uma série de implementações de estruturas de dados, tais como listas , filas , pilhas e arvores de busca.

O Collecrions Framework é responsável por trazer essas estruturas e implementações de forma amigável.

Hoje veremos as principais implementações desse Framework e as principais diferenças entre elas, o objetivo e entender um pouco mais afundo como essas implementações funcionam.

Array

Antes de começar vamos verificar o funcionamento dos Arrays.

Os Arrays em Java são coleções de tamanho fixo baseadas em índices. A principal vantagem de se trabalhar com Arrays, é que podemos representar vários valores com o mesmo nome. O que melhora a legibilidade do código. Para usar um array basta declarar uma variável do tipo desejado usando a seguinte sintaxe:

**Tipo[] nomeDaVariavel = new Tipo[TAMANHO];**

É necessário especificar o tamanho do array, caso contrário ele retorna erro de compilação.

O tamanho do array deve ser um numero inteiro, caso contrário ele retorna a seguinte Runtime Exception: **NegativeArraySizeException.**

Os valores padrão dos Arrays são:

* "0" → no caso de primitivas;
* "Null" → no caso de Objetos;
* "false" → no caso de booleano.

Em alguns casos podemos ter algumas dificuldades para trabalhar com Arrays. Os arrays trabalham com tamanho fixo, logo não podemos redimensionar os mesmos. Se houver necessidade de um Array maior, teríamos que copiar todo o conteúdo de um array para o outro. Não se consegue fazer uma busca por um determinado elemento sem saber o seu índice. Para saber quantas posições do array foram populadas, e preciso criar um método auxiliar.

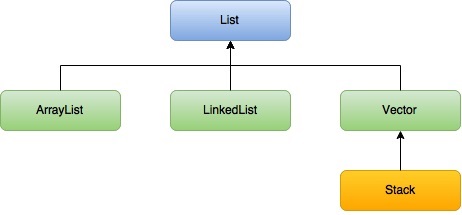
API Collections

Com objetivo de suprir as lacunas deixadas pelos Arrays, o collections Framework traz uma forma mais dinâmica para se trabalhar com listas e colleções.

O framework e dividido em varias interfaces, cujas principais são, List, Set, Map e Queue( filas). Essas interfaces possuem implementações que trazem facilidades para aumentar a produtividade do programador como, ordenação, alocação dinâmica de elementos, busca direta por elementos, sem a necessidade de saber qual o índice do mesmo.

List

A interface List, contida no pacote java.util.List, traz implementações que permitem criar listas ordenadas e com elementos duplicados. Suas principais implementações são: ArrayList, LinkedList e Vector. A imagem abaixo demonstra a hierarquia de algumas classes da interface List.

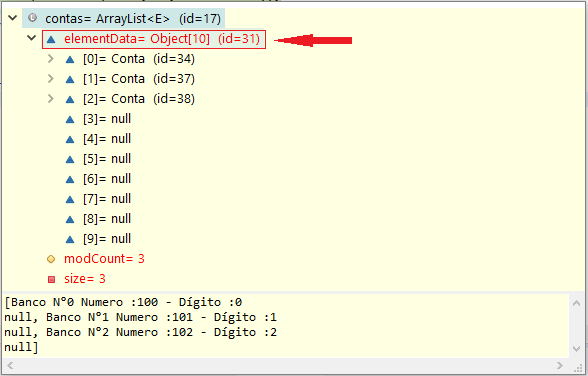


A interface List traz alguns métodos que são herdados de Object, Collection e metodos adicionais.

|  |  |
| --- | --- |
| add(E e) | lastIndexOf(Object o) |
| add(int index, E element) | listIterator() |
| addAll(Collection<? extends E> c) | listIterator(int index) |
| addAll(int index, Collection<? extends E> c) | remove(int index) |
| clear() | remove(Object o) |
| contains(Object o) | removeAll(Collection<?> c) |
| containsAll(Collection<?> c) | retainAll(Collection<?> c) |
| equals(Object o) | set(int index, E element) |
| get(int index) | size() |
| hashCode() | subList(int fromIndex, int toIndex) |
| indexOf(Object o) | toArray() |
| isEmpty() | toArray(T[ ] a) |
| iterator() |  |

ArrayList

O ArrayList é uma das implementações mais usadas da interface List. Apesar de não ser exatamente um array, ele trabalha com um array internamente, como mostra a imagem abaixo.



A classe ArrayList preenche a lacuna deixada pelos Arrays, como acesso direto aos elemento via métodos get() e set(), sem a necessidade de saber o índice do mesmo, não possui tamanho fixo pois os *ArrayList* começa com um tamanho padrão de 10 posições mas o tamanho inicial pode ser definido l no construtor, por exemplo: “*new ArrayList<Conta>(40);”.* Após preencher todas as posições, e forem adicionados mais itens, o ArrayList aumenta sua capacidade em 50%, ou seja , se um ArrayList começar com 10 posições, quando atigir a capacidade, ele aumenta mais 5 posições totalizando 15 e assim por diante.

LinkedList

O linkedList apresenta uma proposta diferente em termos de implementação. Diferente do ArrayList, ele implementa uma lista duplamente encadeada.

equals e hashcode

Muitas vezes no dia-a-da do desenvolvimento Java sobrescrevemos os métodos equals e hashcode em nossas classes por várias razões, dentre elas porque alguma IDE o faz de maneira automática ou mesmo porque seguimos recomendações de boas práticas encontradas em artigos, livros, blogs, etc.

Porém não paramos para analisar qual a utilidade desses métodos e quais são os seus reais benefícios para a aplicação.

O método hashCode() é utilizado para organizar os elementos de uma coleção em um mesmo bucket (balde ou compartimento).

[**http://blog.caelum.com.br/ensinando-que-e-o-hashcode/**](http://blog.caelum.com.br/ensinando-que-e-o-hashcode/)

**http://blog.algaworks.com/entendendo-o-equals-e-hashcode/**

**[https://en.wikipedia.org/wiki/Hash\_table#Choosing\_a\_good\_hash\_function](https://en.wikipedia.org/wiki/Hash_table" \l "Choosing_a_good_hash_function)**

**[http://docs.oracle.com/javase/6/docs/api/java/lang/Object.html#hashCode](http://docs.oracle.com/javase/6/docs/api/java/lang/Object.html" \l "hashCode)()**

[**http://www.matera.com/br/2015/01/15/desvendando-os-metodos-equals-e-hashcode/**](http://www.matera.com/br/2015/01/15/desvendando-os-metodos-equals-e-hashcode/)

[**http://www.onlinetutorialspoint.com/java/comparable-in-java.html**](http://www.onlinetutorialspoint.com/java/comparable-in-java.html)

[**http://www.onlinetutorialspoint.com/java/how-array-works-in-java.html**](http://www.onlinetutorialspoint.com/java/how-array-works-in-java.html)

**[https://www.caelum.com.br/apostila-java-orientacao-objetos/collections-framework/#16-15-exercicios-collections](https://www.caelum.com.br/apostila-java-orientacao-objetos/collections-framework/" \l "16-15-exercicios-collections)**

**[https://www.caelum.com.br/apostila-java-orientacao-objetos/excecoes-e-controle-de-erros/#11-8-o-que-colocar-dentro-do-try](https://www.caelum.com.br/apostila-java-orientacao-objetos/excecoes-e-controle-de-erros/" \l "11-8-o-que-colocar-dentro-do-try)**

[**http://blog.caelum.com.br/lidando-com-exceptions/?utm\_source=Apostila\_HTML&utm\_campaign=FJ-11&utm\_medium=referral**](http://blog.caelum.com.br/lidando-com-exceptions/?utm_source=Apostila_HTML&utm_campaign=FJ-11&utm_medium=referral)

[**http://www.devmedia.com.br/usando-generics-em-java/28981**](http://www.devmedia.com.br/usando-generics-em-java/28981)

[**http://www.makeinjava.com/difference-bw-checked-unchecked-exception-class-hierarchy-example/**](http://www.makeinjava.com/difference-bw-checked-unchecked-exception-class-hierarchy-example/)