LISTA

Listas são estruturas de dados comumente usadas em todas as linguagens de programação.

Neste módulo, vamos estudar a API List do Java. Começaremos com as operações básicas e, em seguida, entraremos em coisas mais avançadas (como uma comparação de diferentes tipos de lista, como ArrayList e LinkedList).

Lista é uma estrutura de dados:

- Homogênea (dados do mesmo tipo)
- Ordenada (elementos acessados por meio de posições)
- Inicia vazia, e seus elementos são alocados sob demanda
- Cada elemento ocupa um "nó" (ou nodo) da lista

Tipo (interface): List

- Classes que implementam: ArrayList, LinkedList, etc.

Vantagens:

- Tamanho variável
- Facilidade para se realizar inserções e deleções

Desvantagens:

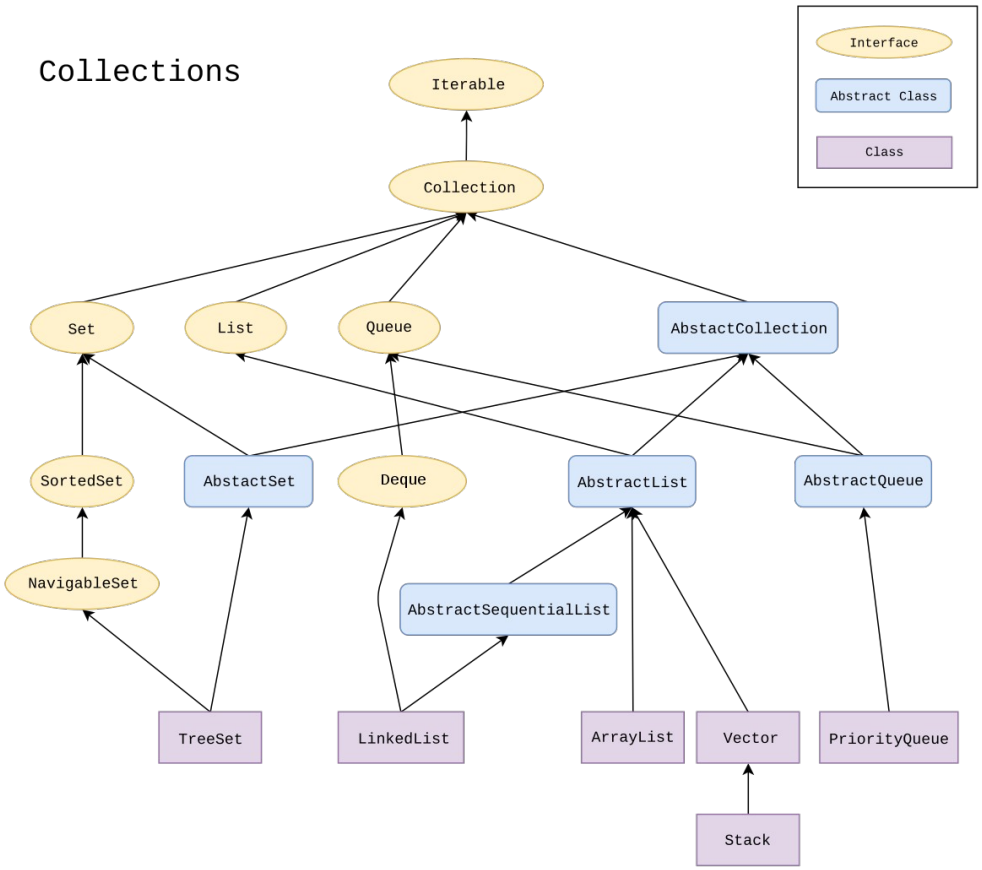
- Acesso sequencial aos elementos *

Definição de listas

Listas são coleções ordenadas de objetos. Nesse sentido, são semelhantes às sequências matemáticas. Eles são conjuntos diferentes, no entanto, que não têm uma determinada ordem.

Algumas coisas para manter em mente: as listas podem ter duplicatas e elementos nulos. Eles são tipos de referência ou de objeto e, como todos os objetos em Java, são armazenados no heap.

Uma lista em Java é uma interface e existem muitos tipos de lista que implementam essa interface.



Hierarquia da coleção

Usarei ArrayList nos primeiros exemplos, porque é o tipo de lista mais comumente usado.

ArrayList é basicamente uma matriz redimensionável. Quase sempre, você deseja usar ArrayList em vez de arrays regulares, pois eles fornecem muitos métodos úteis.

```
import java.util.ArrayList;
import java.util.List;

public class CreateArrayList {
	public static void main(String[] args) {
		ArrayList<Integer> list0 = new ArrayList<>();

		// Makes use of polymorphism
		List list = new ArrayList<Integer>();

		// Local variable with "var" keyword, Java 10
		var list2 = new ArrayList<Integer>();
	}
}
```

Nos colchetes angulares (<>) especificamos o tipo de objetos que iremos armazenar.

Tenha em mente que o **tipo entre colchetes deve ser um tipo de objeto e não um tipo primitivo.** Portanto, temos que usar invólucros de objeto, classe Integer em vez de int, Double em vez de double, e assim por diante.

Existem muitas maneiras de criar um ArrayList, apresentamos três formas de criá-las no código acima.

A primeira maneira é criando o objeto da classe ArrayList concreta especificando ArrayList no lado esquerdo da atribuição.

O segundo trecho de código faz uso de polimorfismo usando lista no lado esquerdo. Isso torna a atribuição fracamente acoplada à classe ArrayList e nos permite atribuir outros tipos de listas e alternar para uma implementação de List diferente facilmente.

Podemos ver que todas as atribuições resultam do mesmo tipo:

```
System.out.println(list0.getClass());
System.out.println(list.getClass());
System.out.println(list.getClass());
```

Resultado:

```
class java.util.ArrayList
class java.util.ArrayList
class java.util.ArrayList
```

Também podemos especificar a capacidade inicial da lista.

```
List list = new ArrayList<>(20);
```

Isso é útil porque sempre que a lista fica cheia e você tenta adicionar outro elemento, a lista atual é copiada para uma nova lista com o dobro da capacidade da lista anterior. Tudo isso acontece nos bastidores.

Esta operação torna nossa complexidade Em), no entanto, queremos evitá-lo. A capacidade padrão é 10, portanto, se você sabe que armazenará mais elementos, deve especificar a capacidade inicial.

Como adicionar e atualizar elementos de lista em Java

Para adicionar elementos à lista, podemos usar o adicionar método. Também podemos especificar o índice do novo elemento, mas tenha cuidado ao fazer isso, pois pode resultar em um IndexOutOfBoundsException.

```
import java.util.ArrayList;

public class AddElement {
	public class AddElement {
		ArrayList<String> list = new ArrayList<>();
		list.add("hello");
		list.add(1, "world");
		System.out.println(list);
	}
}
```

Resultado:

```
[hello, world]");
```

Podemos usar o conjunto método para atualizar um elemento.

```
		list.set(1, "from the otherside");
		System.out.println(list);
```

Resultado:

```
		[hello, world]
		[hello, from the otherside]
```

Como recuperar e excluir elementos de lista em Java

Para recuperar um elemento da lista, você pode usar o obter método e forneça o índice do elemento que você deseja obter.

```
import java.util.List;
import java.util.ArrayList;

public class GetElement {
	public static void main(String[] args) {
		List list = new ArrayList<String>();
		list.add("hello");
		list.add("SoulCode");
	}
}
```

Resultado:

```
	SoulCode
```

A complexidade desta operação em ArrayList é O (1) uma vez que usa uma matriz regular de acesso aleatório em segundo plano.

Para remover um elemento da ArrayList, o remover método é usado.

```
	list.remove(0);
```

Isso remove o elemento no índice 0, que é “olá” neste exemplo.

Também podemos chamar o método remove com um elemento para localizá-lo e removê-lo. Lembre-se de que ele só remove a primeira ocorrência do elemento se estiver presente.

```
	public static void main(String[] args) {
		List list = new ArrayList<String>();
		list.add("hello");
		list.add("SoulCode");
		list.add("SoulCode");

		list.remove("SoulCode");
		System.out.println(list);
	}
}
```

Resultado:

```
	[hello, SoulCode]
```

Para remover todas as ocorrências, podemos usar o deletar tudo método da mesma maneira.
Esses métodos estão dentro da interface List, portanto, todas as implementações de List os possuem (seja ArrayList, LinkedList ou Vector).

Como obter o comprimento de uma lista em Java

Para obter o comprimento de uma lista ou o número de elementos, podemos usar o Tamanho() método.

```
import java.util.ArrayList;
import java.util.List;

public class GetSize {
	public static void main(String[] args) {
		List list = new ArrayList();
```

```
                list.add("Welcome");
                list.add("to my post");
                System.out.println(list);
        }
}
```

Resultado:

```
        2
```

Listas bidimensionais em Java

É possível criar listas bidimensionais, semelhantes a arrays 2D.

```
ArrayList<ArrayList<Integer>> listOfLists = new ArrayList<>();
```

Usamos essa sintaxe para criar uma lista de listas e cada lista interna armazena inteiros. Mas ainda não inicializamos as listas internas. Precisamos criá-los e colocá-los nesta lista:

```
import java.util.ArrayList;
import java.util.List;
int numberOfLists = 3;
for (int i = 0; i < numberOfLists; i++) {
        listOfLists.add(new ArrayList<>());
}
```

Estou inicializando minhas listas internas e adicionando 3 listas neste caso. Também posso adicionar listas mais tarde, se precisar.

Agora podemos adicionar elementos às nossas listas internas. Para adicionar um elemento, precisamos primeiro obter a referência à lista interna.

Por exemplo, digamos que queremos adicionar um elemento à primeira lista. Precisamos obter a primeira lista e, em seguida, adicionar a ela.

```
listOfLists.get(0).add(1);
```

Aqui está um exemplo para você. Tente adivinhar a saída do segmento de código abaixo:

```
public static void main(String[] args) {
        ArrayList<ArrayList<Integer>> listOfLists = new ArrayList<>();
        System.out.println(listOfLists);
        int numberOfLists = 3;
```

```
        for (int i = 0; i < numberOfLists; i++) {
                listOfLists.add(new ArrayList<>());
        }

        System.out.println(listOfLists);

        listOfLists.get(0).add(1);
        listOfLists.get(1).add(2);
        listOfLists.get(2).add(0,3);

        System.out.println(listOfLists);
}
```

Resultado:

```
        []
        [[], [], []]
        [[1], [2], [3]]
```

Observe que é possível imprimir as listas diretamente (ao contrário de arrays regulares) porque eles substituem o para sequenciar() método.

Métodos úteis em Java

Existem alguns outros métodos e atalhos úteis que são usados com frequência.
Nesta seção, quero familiarizá-lo com alguns deles, para que você tenha mais facilidade em trabalhar com listas.

Como criar uma lista com elementos em Java

É possível criar e preencher a lista com alguns elementos em uma única linha.
Existem duas maneiras de fazer isso.

O que se segue é a velha escola:

```
public static void main(String[] args) {
 List<String> list = Arrays.asList(
        “SoulCode”,
        “Força”,
        “Fé”,
        “Foco”);
}
```

Você precisa ter cuidado com uma coisa ao usar esse método: *Arrays.asList* retorna uma lista imutável. Portanto, se você tentar adicionar ou remover elementos após criar o objeto, obterá um

UnsupportedOperationException.

Você pode ficar tentado a usar final palavra-chave para tornar a lista imutável, mas não funcionará como esperado.

Ele apenas garante que a referência ao objeto não mude – não se importa com o que está acontecendo dentro do objeto. Portanto, permite inserir e retirar.

```
        final List<String> list2 = new ArrayList<>();
        list2.add("Com grandes poderes, grandes responsabilidades!");
        System.out.println(list2);
```

Resultado:

```
        [Com grandes poderes, grandes responsabilidades!]
```

Agora, vamos examinar a maneira moderna de fazer isso:

```
        ArrayList<String> friends =  new ArrayList<>(List.of("Gulbike", "Sinem", "Mete"));
```

o Lista de O método foi enviado com o Java 9. Este método também retorna uma lista imutável, mas podemos passá-lo para o construtor *ArrayList* para criar uma lista mutável com esses elementos. Podemos adicionar e remover elementos desta lista sem problemas.

Como criar uma lista com N cópias de algum elemento em Java

Java fornece um método chamado ***Ncopies*** que é especialmente útil para benchmarking. Você pode preencher uma matriz com qualquer número de elementos em uma única linha.

```
public class NCopies {
        public static void main(String[] args) {
                List<String> list = Collections.nCopies(10, "Bootcamp");
                System.out.println(list);
        }
}
```

Resultado:

```
[Bootcamp, Bootcamp, Bootcamp, Bootcamp, Bootcamp, Bootcamp, Bootcamp, Bootcamp, Bootcamp, Bootcamp]
```

Como clonar uma lista em Java

Como mencionado anteriormente, as listas são tipos de referência, portanto, as regras de passando por referência aplicar a eles.

```
public static void main(String[] args) {
        List list1 = new ArrayList<String>();
        list1.add("Soul");
        List list2 = list1;
        list1.add("Code");

        System.out.println(list1);
        System.out.println(list2);
}
```

Resultado:

```
        [Soul,  Code]
        [Soul,  Code]
```

A variável list1 contém uma referência à lista. Quando o atribuímos a list2, ele também aponta para o mesmo objeto. Se não quisermos que a lista original seja alterada, podemos clonar a lista.

```
        ArrayList list3 = (ArrayList) list1.clone();
        list3.add(" Bootcamp");

        System.out.println(list1);
        System.out.println(list3);
```

Resultado:

```
        [Soul,  Code]
        [Soul,  Code, Bootcamp]
```

Como clonamos a lista1, a lista3 contém uma referência ao seu clone neste caso. Portanto, a lista1 permanece inalterada.

Como copiar uma lista para um array em Java

Às vezes, você precisa converter sua lista em uma matriz para passá-la para um método que aceita uma matriz. Você pode usar o seguinte código para fazer isso:

```
List<Integer> list = new ArrayList<>(List.of(1, 2));
Integer[] toArray = list.toArray(new Integer[0]);
```

Você precisa passar um array e o *toArray* método retorna esse *array* após preenchê-lo com os elementos da lista.

Como classificar uma lista em Java

Para classificar uma lista, podemos usar *Collection.sort*. Ele classifica em ordem crescente por padrão, mas você também pode passar um comparador para classificar com lógica personalizada.

```
List<Integer> toBeSorted = new ArrayList<>(List.of(3,2,4,1,-2));
Collections.sort(toBeSorted);
System.out.println(toBeSorted);
```

Resultado:

```
[-2, 1, 2, 3, 4]
```