

ECM251 - Linguagens I

Teoria - Java Collections - Parte 1
Prof. Murilo Zanini de Carvalho



Antes de começar!

Clone seu repositório do Github

- Lembre-se sempre antes de iniciar uma aula, clonar seu repositório remoto e realizar as atividades nele.
- Para cada atividade desenvolvida, criar um novo diretório.



GitHub

Retirado de
(https://miro.medium.com/max/4000/0*MZMI76wKo2FQLqG0.png), em 07/03/2021

Abstração

- Capacidade de representar elementos (físicos com ou não) por suas características e comportamentos.

Encapsulamento

- Isolar a forma como a abstração é realizada, fornecendo uma interface de acesso a ela.

Herança

- Transmissão de características de um elemento geral para elementos mais específicos.

Polimorfismo

- Possibilidade de executar o método correto de acordo com a situação/referência utilizada.

ATENÇÃO PARA AULA DE HOJE!!

Implemente
os códigos
da aula, isso
vai ser
importante!!

When your code compiles
after 253 failed attempts



Retirado de
(<https://www.testbytes.net/wp-content/uploads/2019/06/Untitled-63.png>), em
22/05/2021

Java Collections

The Java platform includes a *collections framework*. A *collection* is an object that represents a group of objects (such as the classic Vector class). A collections framework is a unified architecture for representing and manipulating collections, enabling collections to be manipulated independently of implementation details. The primary advantages of a collections framework are that it:

- **Reduces programming effort** by providing data structures and algorithms so you don't have to write them yourself.
- **Increases performance** by providing high-performance implementations of data structures and algorithms. Because the various implementations of each interface are interchangeable, programs can be tuned by switching implementations.
- **Provides interoperability between unrelated APIs** by establishing a common language to pass collections back and forth.
- **Reduces the effort required to learn APIs** by requiring you to learn multiple ad hoc collection APIs.
- **Reduces the effort required to design and implement APIs** by not requiring you to produce ad hoc collections APIs.
- **Fosters software reuse** by providing a standard interface for collections and algorithms with which to manipulate them.

The collections framework consists of:

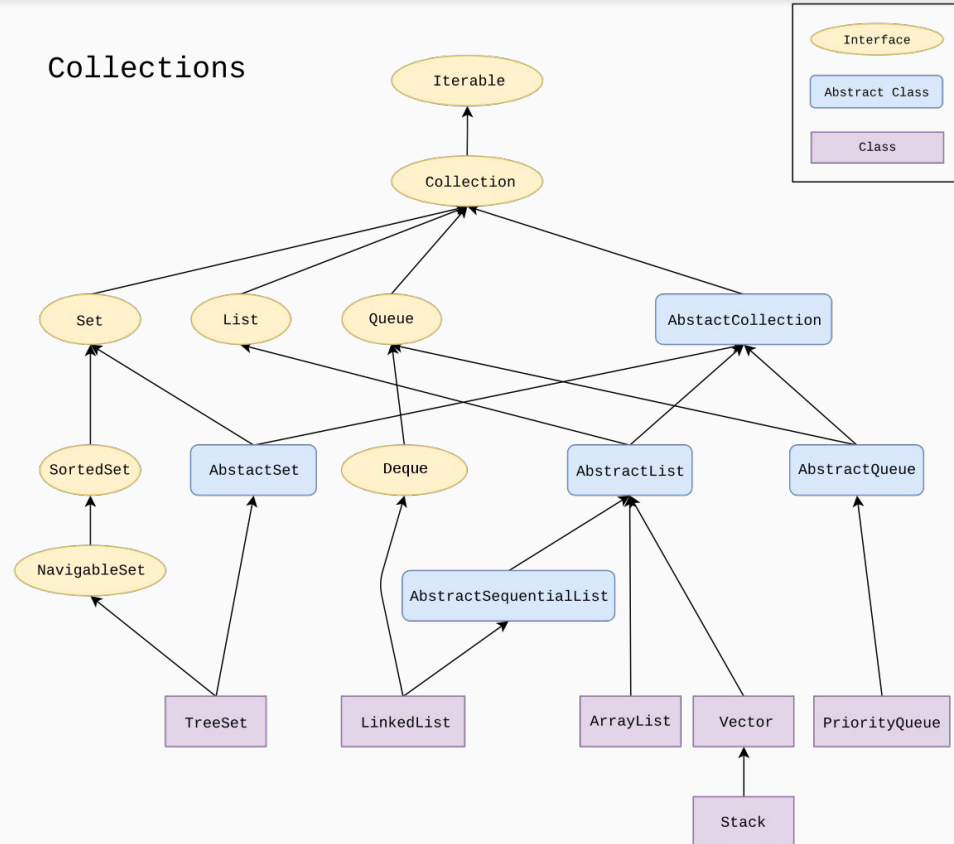
- **Collection interfaces.** Represent different types of collections, such as sets, lists, and maps. These interfaces form the basis of the framework.
- **General-purpose implementations.** Primary implementations of the collection interfaces.
- **Legacy implementations.** The collection classes from earlier releases, Vector and Hashtable, were retrofitted to implement the collection interfaces.
- **Special-purpose implementations.** Implementations designed for use in special situations. These implementations display nonstandard performance characteristics, usage restrictions, or behavior.
- **Concurrent implementations.** Implementations designed for highly concurrent use.
- **Wrapper implementations.** Add functionality, such as synchronization, to other implementations.
- **Convenience implementations.** High-performance "mini-implementations" of the collection interfaces.
- **Abstract implementations.** Partial implementations of the collection interfaces to facilitate custom implementations.
- **Algorithms.** Static methods that perform useful functions on collections, such as sorting a list.
- **Infrastructure.** Interfaces that provide essential support for the collection interfaces.
- **Array Utilities.** Utility functions for arrays of primitive types and reference objects. Not, strictly speaking, a part of the collections framework, this feature was added to the Java platform at the same time as the collections framework and relies on some of the same infrastructure.

Java Collections

- <https://docs.oracle.com/javase/8/docs/technotes/guides/collections/overview.html>
- <https://docs.oracle.com/javase/tutorial/collections/interfaces/index.html>
- <https://docs.oracle.com/javase/8/docs/technotes/guides/collections/index.html>
- <https://docs.oracle.com/javase/8/docs/api/?java/util/Collections.html>
- <https://www.devmedia.com.br/java-collections-como-utilizar-collections/18450>
- <https://docs.oracle.com/javase/tutorial/collections/intro/index.html>

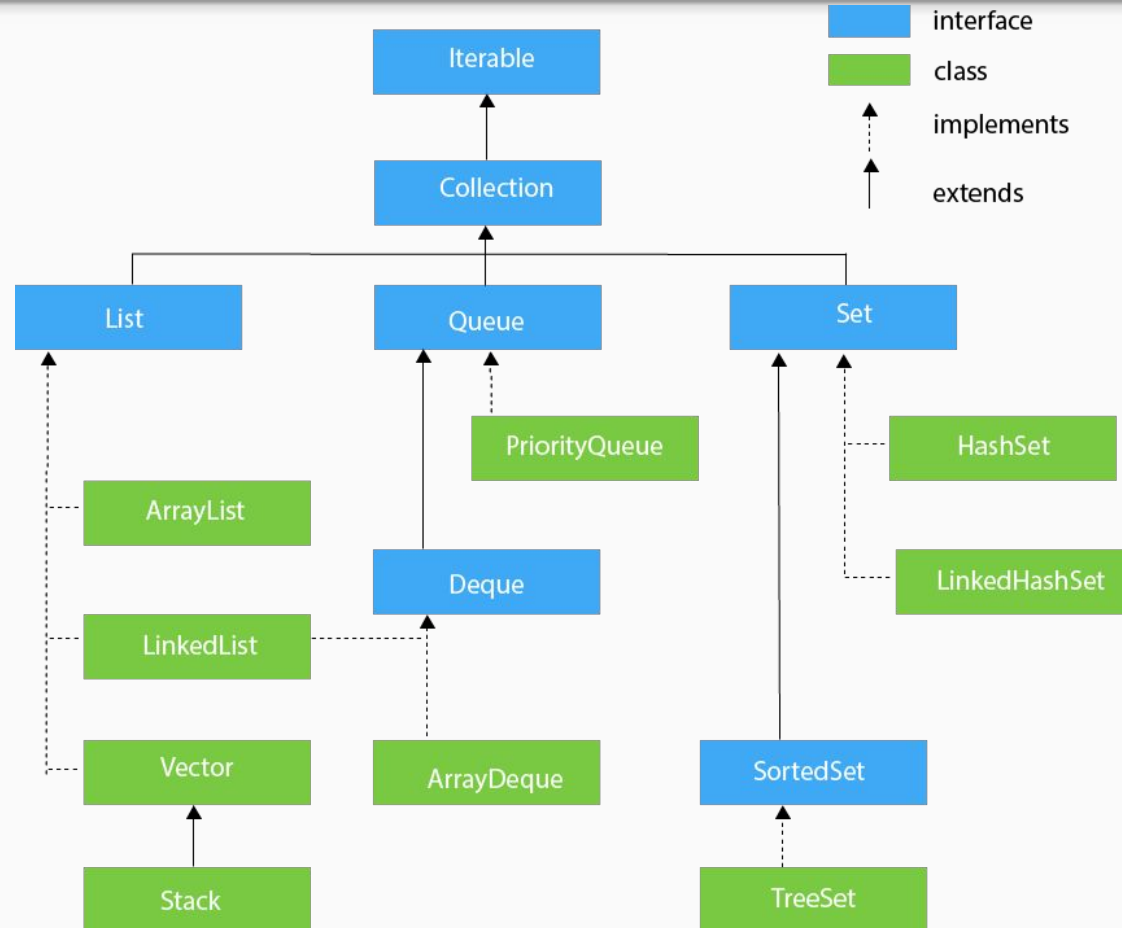
Java Collections

Collections



Retirado de
(https://upload.wikimedia.org/wikipedia/commons/thumb/a/ab/Java.util.Collection_hierarchy.svg/1200px-Java.util.Collection_hierarchy.svg.png), em 09/05/2020.

Java Collections



Retirado de
(<https://static.javatpoint.com/images/java-collections-hierarchy.png>), em 09/05/2020.

Arrays

Arrays

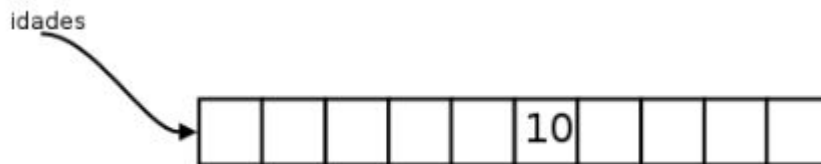
O `int[]` é um tipo. Um **array** é sempre um objeto, portanto, a variável `idades` é uma referência. Vamos precisar criar um objeto para poder usar a array. Como criamos o objeto-array?

```
idades = new int[10];
```

O que fizemos foi criar um array de `int` de 10 posições e atribuir o endereço no qual ela foi criada. Podemos ainda acessar as posições do array:

```
idades[5] = 10;
```

Resultando em:



Trecho completo:

```
int[] idades;  
  
idades = new int[10];  
  
idades[5] = 10;
```

O código acima altera a sexta posição do array. No Java, os índices do array vão de **0** a **n-1**, onde **n** é o tamanho dado no momento em que você criou o array. Se você tentar acessar uma posição fora desse alcance, um erro ocorrerá durante a execução.

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:
```

Array de Objetos

Mesmo que seja comum ouvirmos a expressão “**array de objetos**”, temos que ter em mente que na verdade possuímos um **array** de referências para os objetos, e portanto eles precisam ser criados individualmente.

```
Conta[] minhasContas;  
minhasContas = new Conta[10];
```

Quantas objetos contas foram criadas aqui?

Na verdade, nenhuma. Foram criados 10 espaços que você pode utilizar para guardar uma referência a uma Conta. Por enquanto, eles se referenciam para lugar nenhum (null).

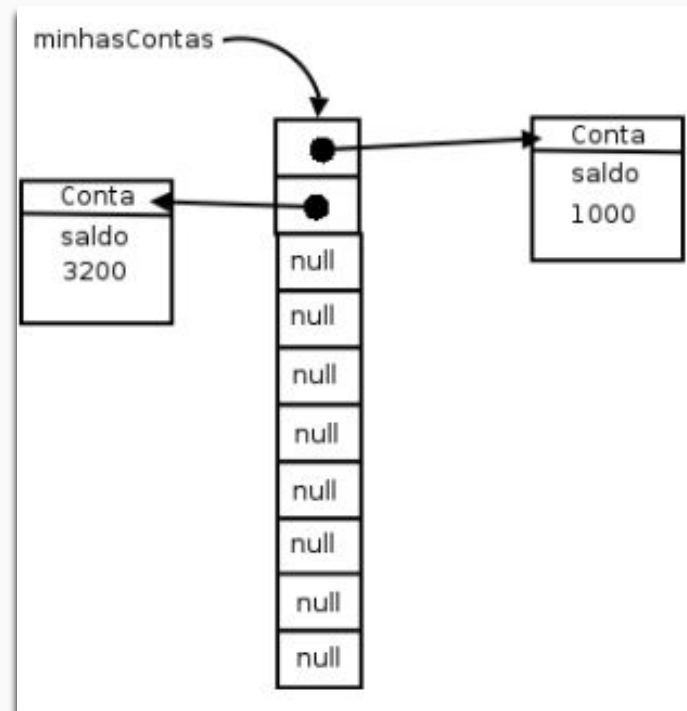
Array de Objetos

Você deve popular seu array antes.

```
Conta contaNova = new Conta();  
contaNova.saldo = 1000.0;  
minhasContas[0] = contaNova;
```

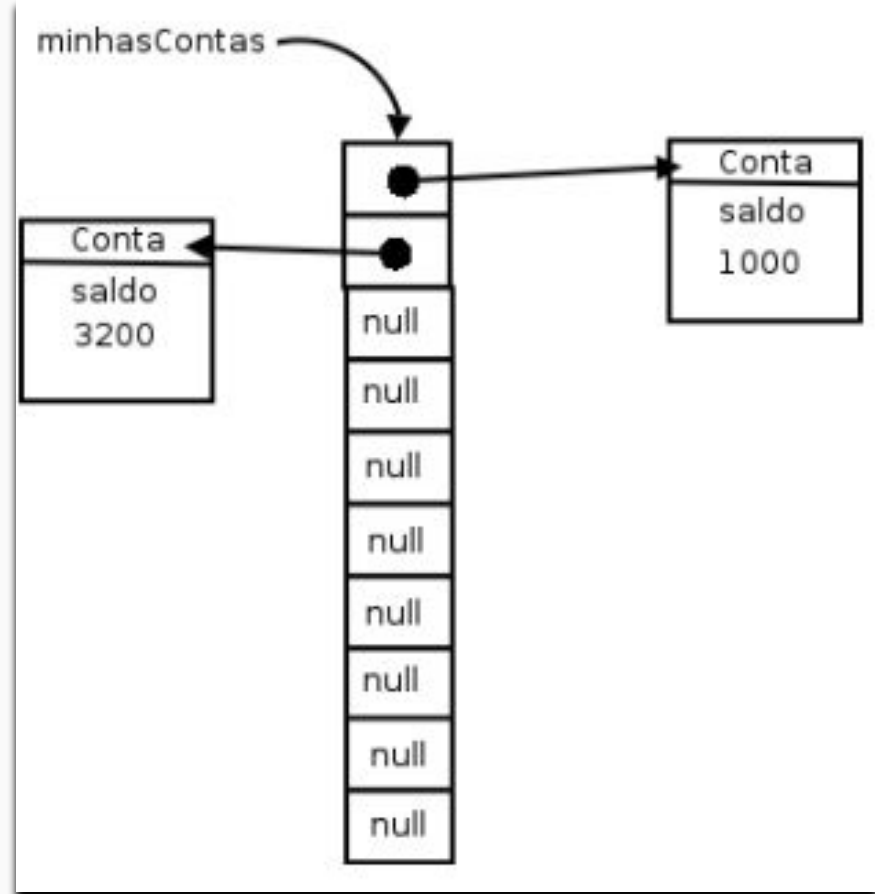
Ou você pode fazer assim:

```
minhasContas[1] = new Conta();  
minhasContas[1].saldo = 3200.0;
```



Array de Objetos

Uma **array** de tipos primitivos guarda valores, uma array de objetos guarda **referências**.



Percorrer um **array** é muito simples, basta utilizar alguma estrutura de repetição. Sugestões?

```
public static void main(String args[]) {  
    int[] idades = new int[10];  
    for (int i = 0; i < 10; i++) {  
        idades[i] = i * 10;  
    }  
    for (int i = 0; i < 10; i++) {  
        System.out.println(idades[i]);  
    }  
}
```

Um pouco mais de Arrays

Arrays como parâmetros

```
private static void printArray(int[] arr)
{
    for(int i = 0; i < arr.length; i++)
        System.out.print(arr[i] + " ");
    System.out.println();
}
```

Arrays como retorno de funções

```
public static int[] getIntegers(int number) {  
    System.out.println("Enter " + number + " integer values.\r");  
    int[] values = new int[number];  
  
    for(int i=0; i<values.length; i++) {  
        values[i] = scanner.nextInt();  
    }  
  
    return values;  
}
```

Classe ArrayList

O Java, por padrão, possui uma série de recursos prontos (APIs) para que possamos tratar de estrutura de dados, também chamados de coleções (collections).

Podemos dizer que ArrayList é uma classe para coleções.

Você pode criar seus objetos - através de uma classe - e agrupá-los através de ArrayList e realizar, nessa coleção, várias operações, como: **adicionar** e **retirar** elementos, ordená-los, procurar por um elemento específico, apagar um elemento específico, limpar o ArrayList dentre outras possibilidades.

ArrayList - Documentação

O **List** é uma interface e o **ArrayList** é a classe que a implementa.

<https://docs.oracle.com/javase/8/docs/api/java/util/List.html>

<https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html>

Coding Time!!

Coding Time

Crie uma classe pessoa e adicione os seus elementos em um ArrayList.

Percorra e exiba os itens dessa lista.

Adicione um novo elemento na lista.

Exiba novamente os elementos da lista.

Observação: Lambda Expressions ->

(https://www.w3schools.com/java/java_lambda.asp#:~:text=Lambda%20Expressions%20were%20added%20in,the%20body%20of%20a%20method.)

Observações com as classes Array

Lembre-se que a classe **ArrayList** implementa a interface **List** utilizando um **array**. A seguir verifique a representação um **array** de inteiros na memória.

Index	Address	Value
0	100	34
1	104	18
2	108	91
3	112	57
4	116	453
5	120	68
6	124	6

Um **array** possui seus valores em endereços sequenciais de memória. O passo é definido pelo tipo de dado que o **array** possui, por exemplo um valor inteiro possui 4 bytes, então o passo de um **array** de **inteiros** será 4.

Um **array** de **double** teria um passo de 8 bytes.

Então quando é requisitado o acesso a um índice do **array**, através de uma simples conta é possível ir diretamente ao endereço de memória desse índice.
(end_Inicial + índice*tam_word)

Ex. índice 3 de um **array** de inteiros:

$i = 3$

Address = $100 + 3 * 4$ (4=n. em bytes de um inteiro)

Address = 112

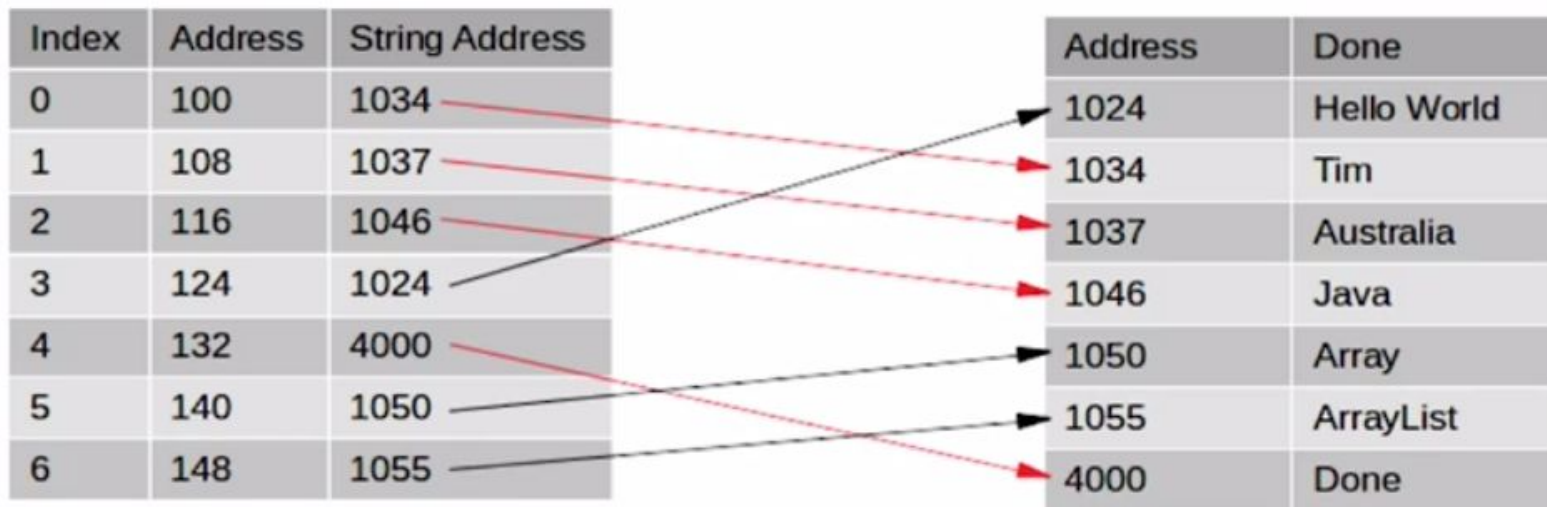


Index	Address	Value
0	100	34
1	104	18
2	108	91
3	112	57
4	116	453
5	120	68
6	124	6



Arrays

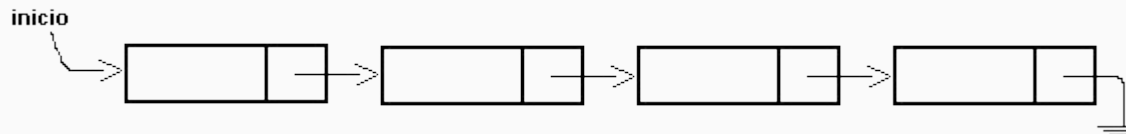
Mas... E se tenho uma **array** de **Strings** que pode ser de tamanho variável?
Como faço ?



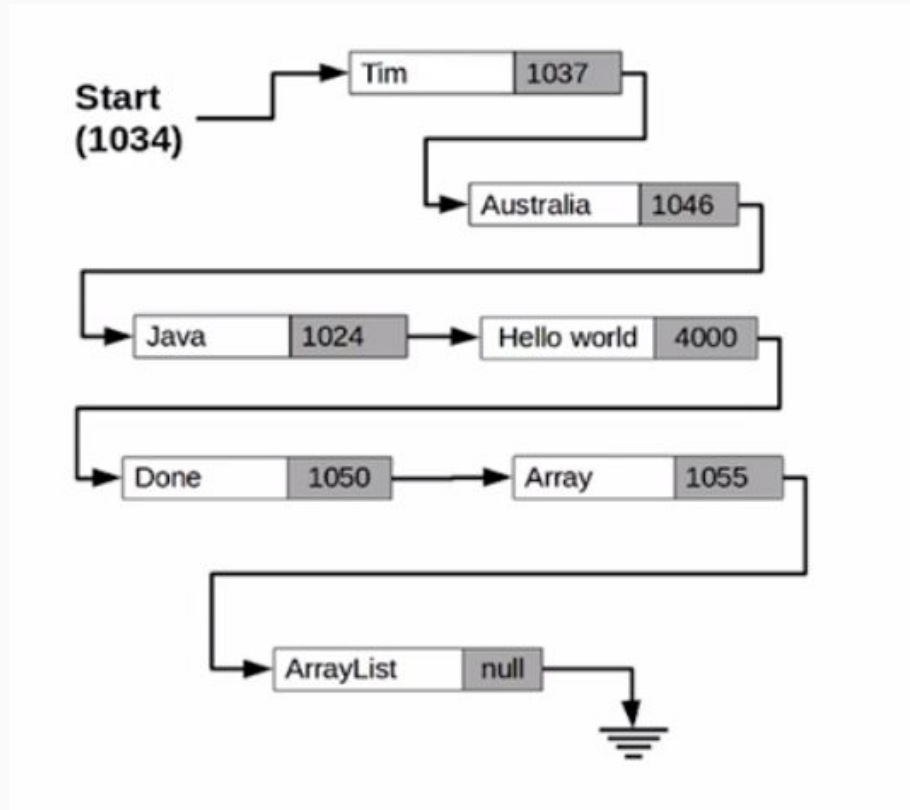
Linked List

As listas ligadas ou encadeadas são conjuntos de elementos encadeados, onde cada elemento contém uma ligação com um ou mais elementos da lista.

A grande diferença entre as listas e as pilhas ou filas é que as listas não possuem regras para o acesso de seus elementos. Ou seja, a inclusão, exclusão ou consulta dos elementos da lista podem acontecer de forma aleatória.



Exemplo de implementação de uma Lista Ligada.

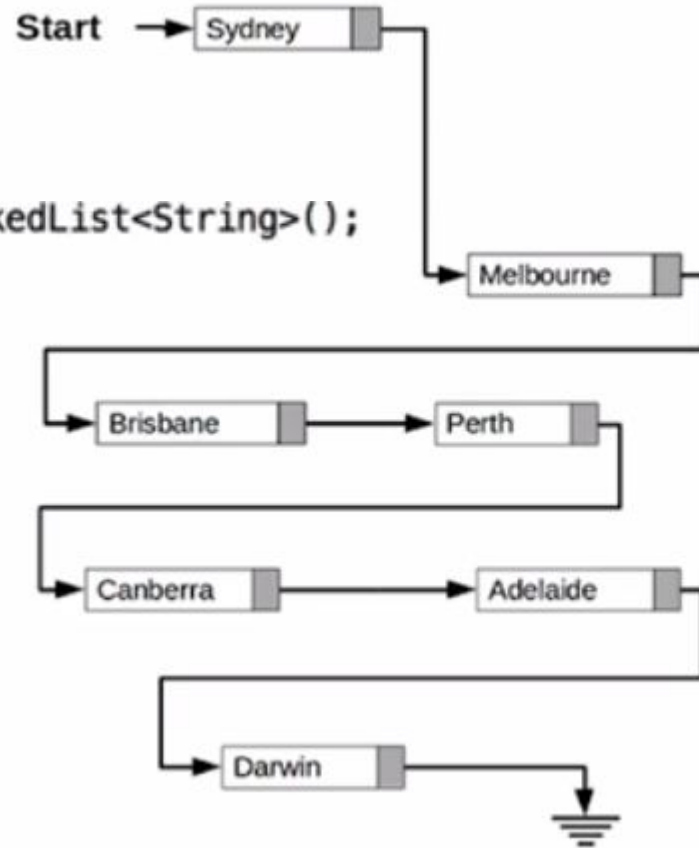


Operações com LinkedList

Criação de uma LinkedList

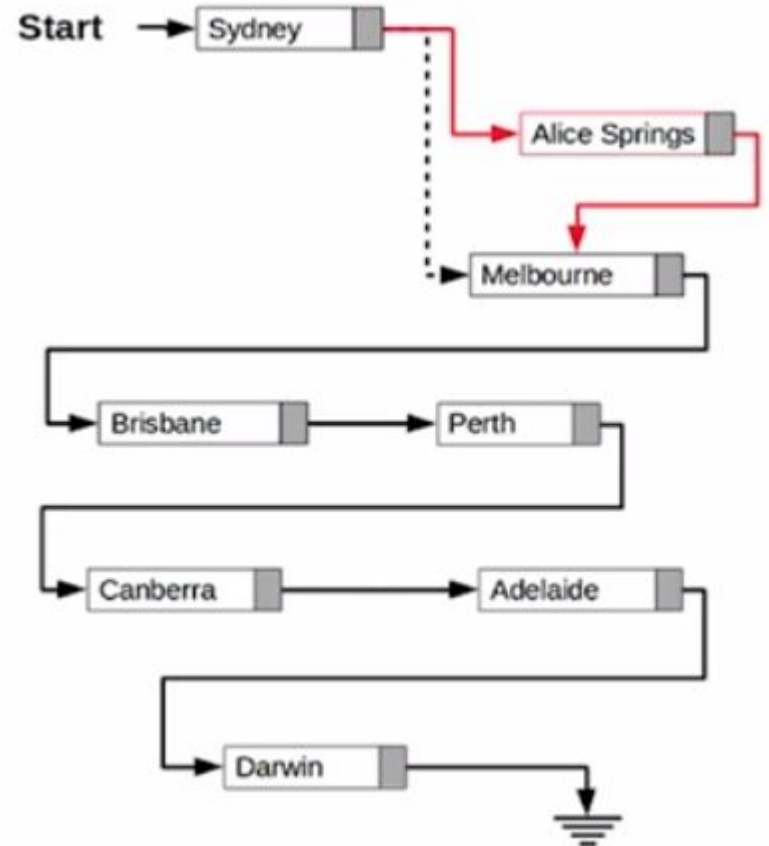
Imagine uma lista de cidades.

```
LinkedList<String> placesToVisit = new LinkedList<String>();  
placesToVisit.add("Sydney");  
placesToVisit.add("Melbourne");  
placesToVisit.add("Brisbane");  
placesToVisit.add("Perth");  
placesToVisit.add("Canberra");  
placesToVisit.add("Adelaide");  
placesToVisit.add("Darwin");
```



Adicionar um novo elemento no LinkedList

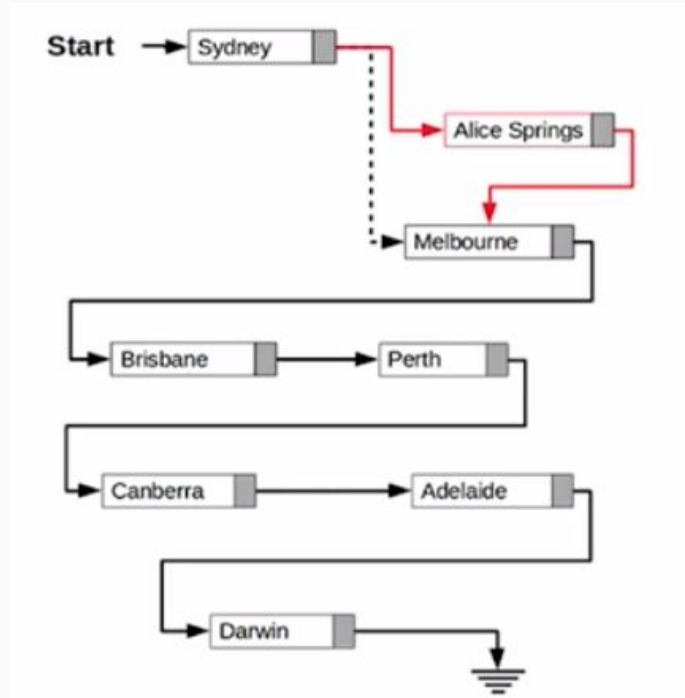
Para adicionar um elemento, basta que o ponteiro do item anterior a posição de inserção aponte para o novo elemento, e o novo elemento aponte para o item a seguir da lista.



Adicionar um novo elemento no LinkedList

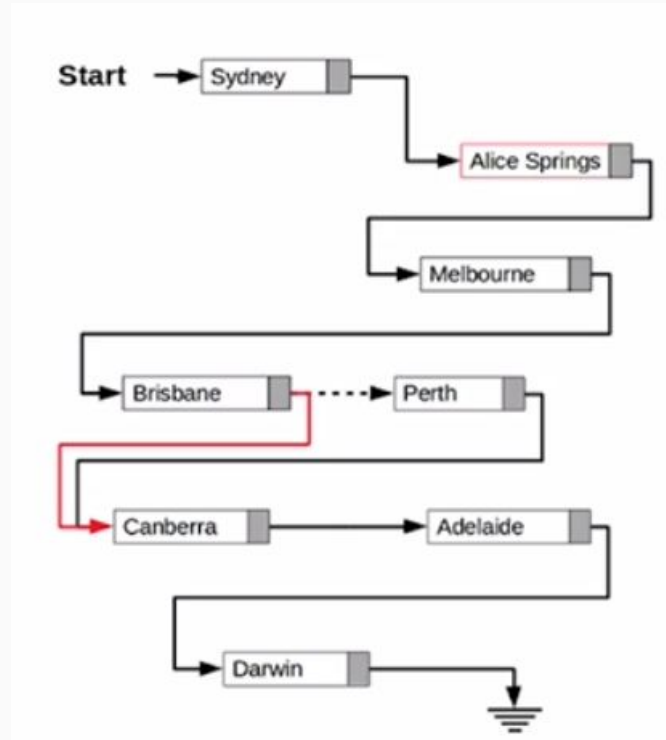
E adicionar um elemento em um LinkedList no Java?

```
placesToVisit.add(1, "Alice Springs");
```



Removendo um novo elemento no LinkedList

```
placesToVisit.remove(4);
```



Iterator

Iterator (Iterador) é um **padrão de projeto**, também conhecido como **Design Pattern**, que visa simplificar a iteração sobre um conjunto de objetos.

Os Padrões de Projeto também conhecidos como **Design Patterns** (em inglês) são soluções já encontradas, testadas e comprovadas e que podemos aplicar aos projetos sem ter que reinventar a roda.

Diversos Padrões de Projeto já foram catalogados e são um conjunto de boas práticas que devemos seguir e utilizar em projetos de software orientados a objetos.

Padrões de Projetos basicamente descrevem soluções para problemas recorrentes no desenvolvimento de sistemas de software orientados a objetos. Um padrão de projeto estabelece um nome e define o problema, a solução, quando aplicar esta solução (contexto) e suas consequências.

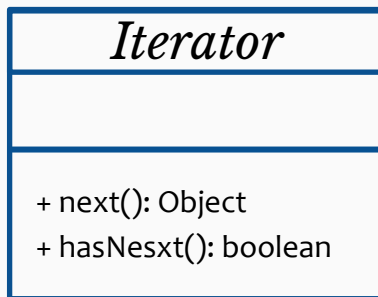
Além disso, os Padrões de Projeto também definem um vocabulário comum que facilita a comunicação, documentação e aprendizado dos sistemas de software.

Veremos mais sobre o Padrão de projeto **Iterator** que é utilizado em diversos projetos e na API do Java, inclusive na interface **List**.

Recomendação: <https://refactoring.guru/pt-br/design-patterns>

Como funciona o Iterator?

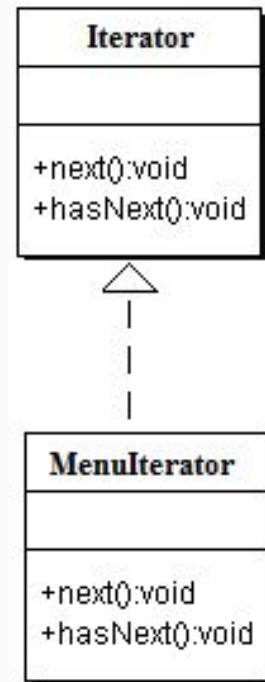
O Padrão de Projeto **Iterator** tem como objetivo encapsular uma iteração. O Padrão de Projeto **Iterator** depende de uma interface chamada **Iterator**, conforme pode ser vista abaixo:



Como funciona o Iterator?

Depois que já possuímos a interface podemos implementar **iteradores** para qualquer tipo de coleção de objetos sejam matrizes, listas, hashtables, etc.

Para cada coleção de objetos que queira-se encapsular a iteração cria-se uma implementação para a interface **Iterator** definida abaixo.



Iterator na API do Java

O **Iterator** é uma interface definida no Java (`java.util.Iterator`), e pode ser encontrado nas classes que implementam a interface **List**.

Interface List<E>

Type Parameters:

E - the type of elements in this list

All SuperInterfaces:

`Collection<E>`, `Iterable<E>`

<https://www.geeksforgeeks.org/difference-between-an-iterator-and-listiterator-in-java/>

Iterator<E>

`iterator()`

Returns an iterator over the elements in this list in proper sequence.

int

`lastIndexOf(Object o)`

Returns the index of the last occurrence of the specified element in this list, or -1

ListIterator<E>

`listIterator()`

Returns a list iterator over the elements in this list (in proper sequence).

Verifique a diferença no código abaixo:

```
private static void printList(LinkedList<String> linkedList) {  
    for (int i = 0; i < linkedList.size(); i++) {  
        System.out.println( linkedList.get(i) );  
    }  
    System.out.println("=====");  
}
```

```
private static void printList(LinkedList<String> linkedList) {  
    Iterator<String> i= linkedList.iterator();  
    while(i.hasNext()) {  
        System.out.println("Now visiting " + i.next());  
    }  
    System.out.println("=====");  
}
```


Reflexões

Perguntas?



Retirado de
(<https://cdn-icons-png.flaticon.com/512/1268/1268705.png>), em 02/03/2022