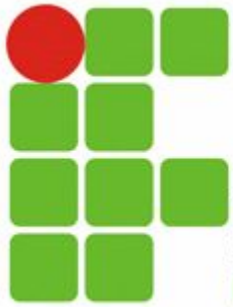


INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
FLUMINENSE

Interface LIST (java.util.List) e suas implementações

Disciplina
PROGRAMAÇÃO ORIENTADA A OBJETOS

Prof. Roberta B Tôrres

Listas (java.util.List)

- Um dos recursos da API de Collections são as **Listas**.
- Uma **Lista** é uma coleção que permite elementos duplicados e mantém uma ordenação específica entre os elementos.
- Em um conjunto do tipo **List**, os objetos são organizados um após o outro em sequência. Podem ser inseridos no final da lista (**método add()**) ou em uma posição específica da lista (**método add(index, object)**).
- Como em um array, há uma ordem bem definida no armazenamento (sendo uma coleção ordenada), mas com a vantagem de não ter um tamanho fixo.

Listas (java.util.List)

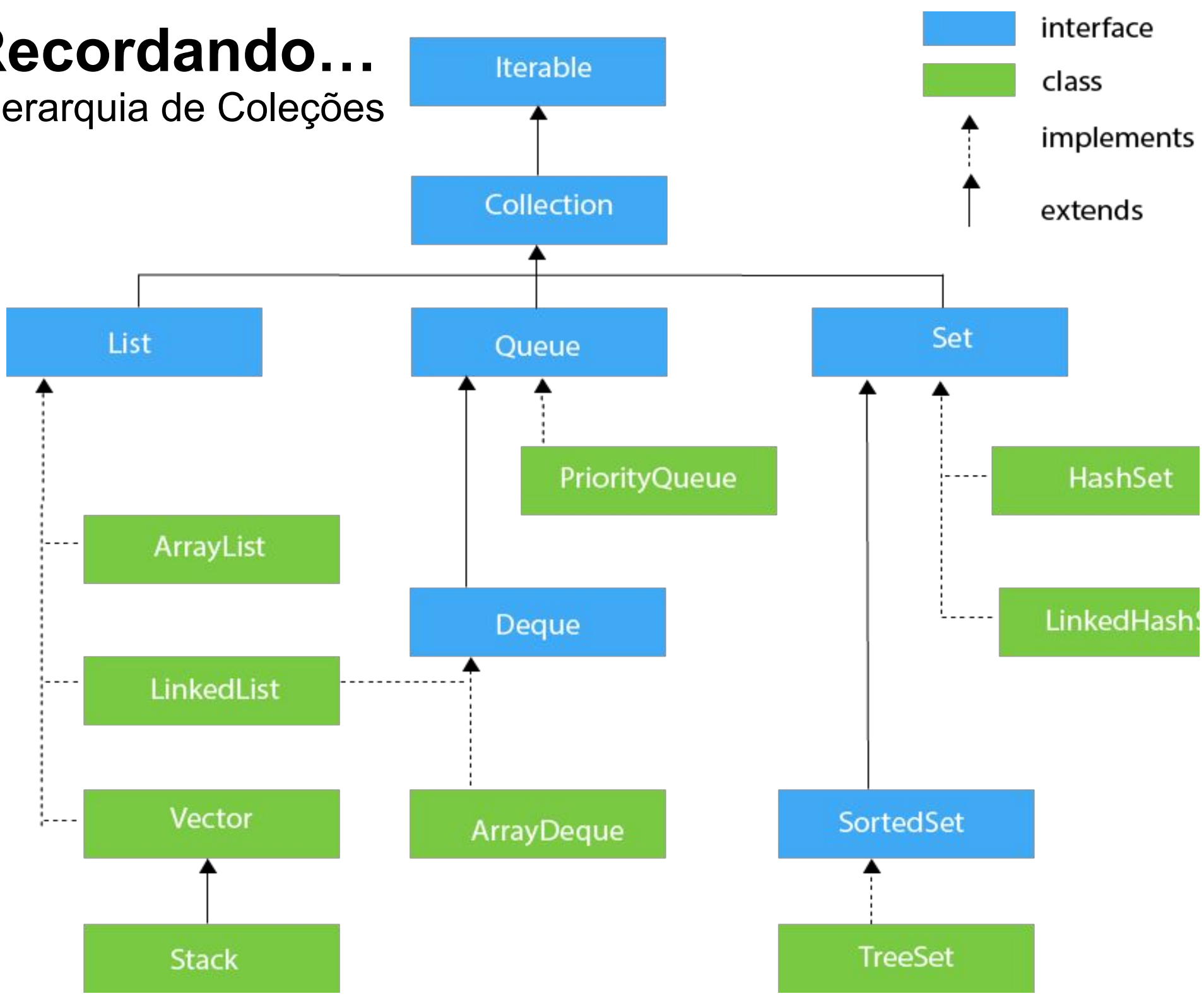
- Como em um array, **List** dá muita importância ao índice, ou seja, a posição que o elemento ocupa na coleção, fornecendo vários métodos relacionados a ele, tais como:
  - **get(int index)**: Retorna o objeto armazenado na posição informada no parâmetro.
  - **indexOf(E e)**: Retorna um índice da posição da primeira vez que o objeto passado por parâmetro aparece na coleção. Retorna -1, se o objeto não for encontrado.
  - **add(E e)**: Insere o objeto passado por parâmetro no final da lista.
  - **add(int index , E e)**: Insere o objeto passado na lista, na posição **index** indicada. Neste caso, os demais elementos são reordenados nas posições seguintes.
  - **remove(int index)**: Remove o objeto que está na posição index.

Listas (java.util.List)

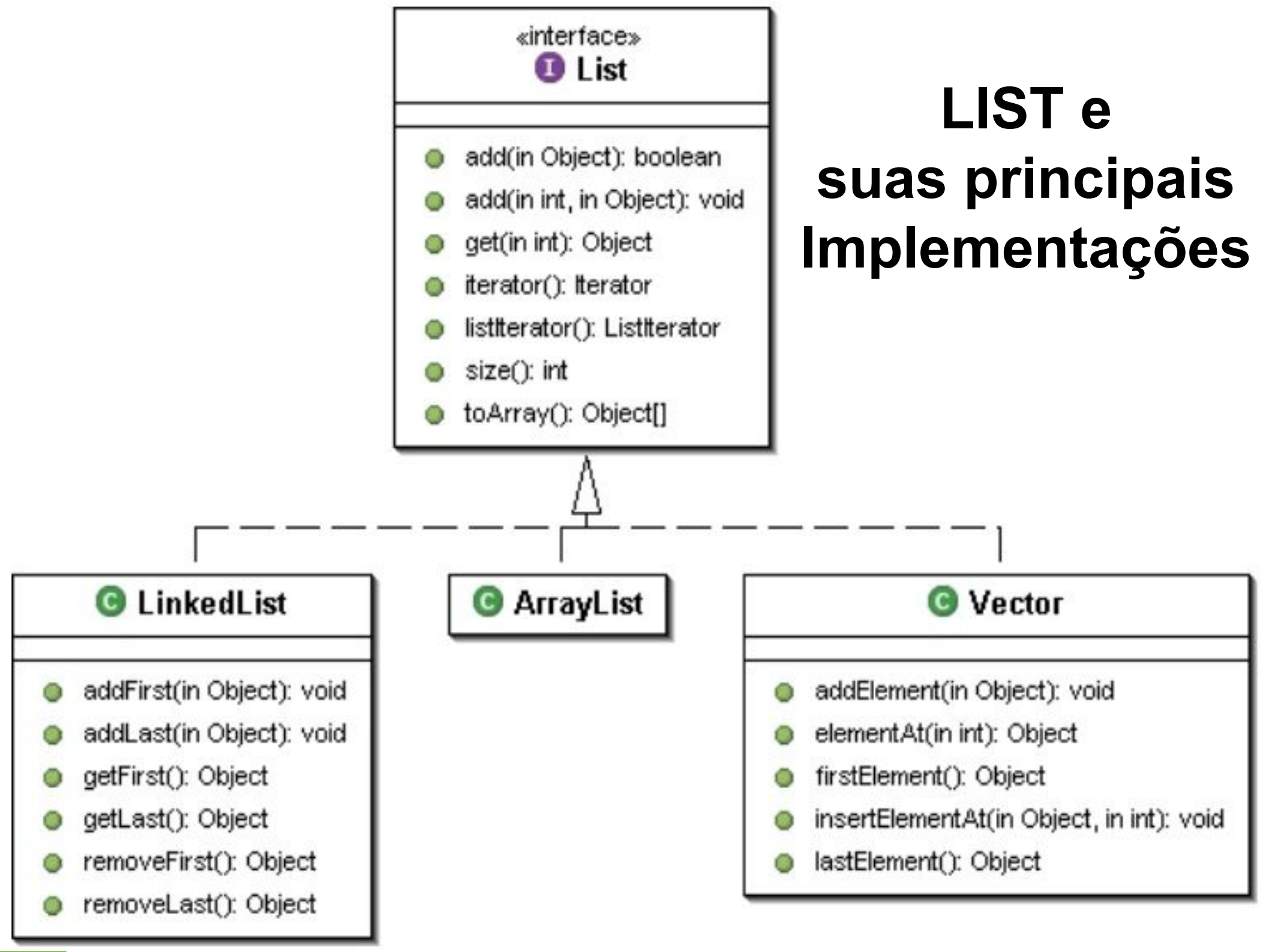
- As implementações mais utilizadas da interface **List** são:
  - **ArrayList:** que trabalha com um array interno para gerar uma lista. É mais rápida na pesquisa do que sua concorrente, a **LinkedList** (que é mais rápida na inserção e remoção de itens nas pontas).
  - **LinkedList** é uma implementação de Lista utilizando uma lista ligada.

Recordando…

Hierarquia de Coleções



LIST e suas principais Implementações



Listas (java.util.List)

- LinkedList
  - Armazena cada um de seus elementos em um espaço de memória que sempre possui uma referência para o próximo item.
  - Mais rápido para inserção/exclusão (insert/remove)
  - Mais lento para obter seus valores (get)
- ArrayList
  - Armazena seus elemento em um array interno, reformulando-o a cada inserção ou remoção.
  - Mais rápido para obter seus valores (get)
  - Mais lento para inserção/exclusão (insert/remove)
- Vector
  - Antiga implementação da interface List que garante a integridade de seus dados quando acessado por processos concorrentes (threads)

Listas com Objetos diferentes

Em qualquer lista, é possível colocar qualquer ***Object***. Com isso, é possível misturar objetos:

```
ContaCorrente cc = new ContaCorrente();

List lista = new ArrayList();
lista.add("Uma string");  // objeto String
lista.add(cc);...         //objeto ContaCorrente
```

Mas e depois, na hora de recuperar esses objetos? Como o método **get** devolve um ***Object***, precisamos fazer o **cast**.

Mas com uma lista com vários objetos de tipos diferentes, isso pode não ser tão simples...

Listas com Generics

Geralmente, não nos interessa uma lista com vários tipos de objetos misturados. Para restringir as listas a um mesmo tipo de objeto pode-se usar o recurso de **Generics**.

Generic é um nova extensão adicionada na linguagem Java a partir da versão 1.5, a qual provê em tempo de compilação uma verificação de type-safety de código, removendo riscos de ***ClassCastException*** durante a execução, o qual era um erro comum antes da versão 1.5.

Essa verificação permite validar se o que está sendo atribuído a uma instância de uma classe está de acordo com o especificado.

EXEMPLO

***List*<*ContaCorrente*>** contas = new ArrayList**<ContaCorrente>**();

Listas com Generics

Repare no parâmetro ao lado de **List** e **ArrayList**: ele indica que nossa lista foi criada para trabalhar exclusivamente com objetos do tipo **ContaCorrente**. Isso nos traz uma segurança em tempo de compilação.

```
List<ContaCorrente> contas = new ArrayList<ContaCorrente>();
contas.add(c1);
contas.add(c3);
contas.add(c2);
```

Logo, o comando contas.add("uma string"); não compila mais!!

Listas com Generics

O uso de **Generics** também elimina a necessidade de casting (tipagem), já que, seguramente, todos os objetos inseridos na lista serão do tipo ***ContaCorrente***:

```
for(int i = 0; i < contas.size(); i++) {
    ContaCorrente cc = contas.get(i); // sem casting!
    System.out.println(cc.getSaldo());
}
```

A partir do Java 7, ao instanciar um tipo genérico na mesma linha de sua declaração, não é necessário passar os tipos novamente, basta usar **new ArrayList<>()**. É conhecido como operador diamante:

```
List<ContaCorrente> contas = new ArrayList<>();
```

Listas com Generics Exemplo

```
List<Cliente> lista = new ArrayList<Cliente>();

lista.add(new Cliente("João", "Gold", "6781-9874"));
lista.add(new Cliente("Manuel", "Bronze", "4532-7125"));
lista.add(new Cliente("Joaquim", "Silver", "7945-0257"));
lista.add(1, new Cliente("Maria", "Gold", "7801-2068"));

Cliente c = lista.get(1);
System.out.println("Cliente 1: " + c.getNome());

lista.remove(2);
System.out.println("Total de clientes: " + lista.size());
```

Método sort (ordenação)

Vimos anteriormente que as listas são percorridas de maneira pré-determinada de acordo com a inclusão dos itens. Mas, muitas vezes, queremos percorrer a nossa lista de maneira ordenada.

A classe `Collections` traz um método estático `sort` que recebe um `List` como argumento e o ordena por ordem crescente. Por exemplo:

```
List<String> lista = new ArrayList<>();
lista.add("Sérgio");
lista.add("Paulo");
lista.add("Guilherme");

// repare que o toString de ArrayList foi sobrescrito:
System.out.println(lista);
Collections.sort(lista);
System.out.println(lista);
```

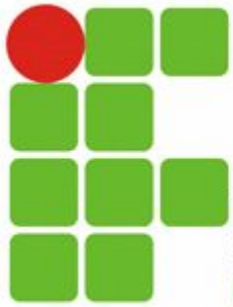
Lista com Generics e método sort - Exemplo

No Tópico **Exemplos de Arrays (Tema 09)**, faça download do arquivo **Exemplo06.java** e e execute esta classe. A mesma manipula um objeto **List com Generics e método sort**.

Veja mais sobre Ordenação de Objetos em:

https://www.caelum.com.br/apostila-java-orientacao-objetos/collections-framework/

No Tópico 15.5 - **Ordenação: Collections.sort**, busque pela explicação sobre Ordenação implementando a interface **java.lang.Comparable**.



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
FLUMINENSE

Classe ArrayList

ArrayList

ArrayList não é um array!

É comum confundirem uma `ArrayList` com um array, porém ela não é um array. O que ocorre é que, internamente, ela usa um array como estrutura para armazenar os dados, porém este atributo está propriamente encapsulado e você não tem como acessá-lo. Repare, também, que você não pode usar `[]` com uma `ArrayList`, nem acessar atributo `length`. Não há relação!

ArrayList

Para criar um `ArrayList`, basta chamar o construtor:

```
ArrayList lista = new ArrayList();
```

É sempre possível abstrair a lista a partir da interface `List`:

```
List lista = new ArrayList();
```

Note que, em momento algum, dizemos qual é o tamanho da lista.

Pode-se acrescentar quantos elementos quiser, que a lista cresce conforme for necessário.

- **ArrayList** armazena seus itens em um array que é iniciado com um determinado tamanho (normalmente 10).
- Ao inserir um elemento, ele verifica se é necessário aumentar o array, se sim ele cria um novo array e copia todos os objetos do antigo para o novo (e maior) e descarta o antigo.

ArrayList - Exemplo

Extraído de
https://www.botecodigital.dev.br/java/uma-visao-sobre-o-framework-collections-do-java/

```java
// criando um arraylist que só irá aceitar String
ArrayList<String> c = new ArrayList<String>();

c.add("Rodrigo"); //adicionando ao final da lista, posição 0
c.add("Maria");   //adicionando ao final da lista, posição 1
c.add("Thiago");  //adicionando ao final da lista, posição 2
c.add("João");    //adicionando ao final da lista, posição 3

//pegando o a quantidade de objetos - saída: Tamanho: 4
System.out.println("Tamanho: " + c.size() );

//pegando um objeto especifico - saída: A string na posição 2: Thiago
System.out.println("A string na posição 2: " + c.get(2) );

//pegando a posição de um determinado objeto - saída: A posição da String 'Maria':
System.out.println("A posição da String 'Maria': " + c.indexOf("Maria") );

//removendo a String na posição 1, ou seja "Maria"
c.remove( 1 );
```

Acesso aleatório com GET

- Algumas listas, como a **ArrayList**, têm acesso aleatório aos seus elementos: a busca por um elemento em uma determinada posição é feita de maneira imediata, sem que a lista inteira tenha que ser percorrida (o que chamamos de acesso sequencial).

- Neste caso, o acesso através do método **get(int)** é muito rápido.

- Uma **ArrayList** é uma excelente alternativa a um array comum, já que temos todos os benefícios de arrays, sem a necessidade de tomar cuidado com remoções, falta de espaço etc.

ArrayList - Exemplo

Extraído de
https://www.botecodigital.dev.br/java/uma-visao-sobre-o-framework-collections-do-java/

```
//percorrendo através de  um iterator(objeto responsável por percorrer uma coleção)
Iterator<String> i = c.iterator();// pegando o iterator da coleção
while( i.hasNext() ){//enquanto tiver um próximo objeto na coleção
    //retorna o próximo elemento da coleção e incrementa o contador interno do itarator
    String nome = i.next();

    System.out.println( nome );
}

System.out.println("============================");

// adicionando na posição 1 a String "Mario", sendo que a o objeto que está na
// posição 1 passa para a posição 2, o objeto na posição 2 para 3 e assim por diante
c.add(1, "Mario");

//percorrendo a lista através de um "for aprimorado"
//para cada objeto presente na lista "c" o objeto será coloca em "nome" e executará o blo
for(String nome : c){
    System.out.println( nome );
}
```

ArrayList - Exemplo

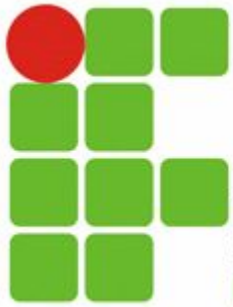
Extraído de
https://www.botecodigital.dev.br/java/uma-visao-sobre-o-framework-collections-do-java/

```
//criando um array com o mesmo tamanho dos objetos da coleção
String[] nomes = new String[ c.size() ];

//colocando os objetos da coleção dentro do array passado por parâmetro
c.toArray( nomes );

//percorrendo o array
for(int j = 0 ; j < nomes.length ; j++){
    System.out.println(nomes[j]);
}
```

No Tópico **Exemplos de Arrays (Tema 09)**, faça download do arquivo **Exemplo07.java** e execute-a. Este arquivo manipula um objeto **ArrayList.**



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
FLUMINENSE

Classe LinkedList

LinkedList

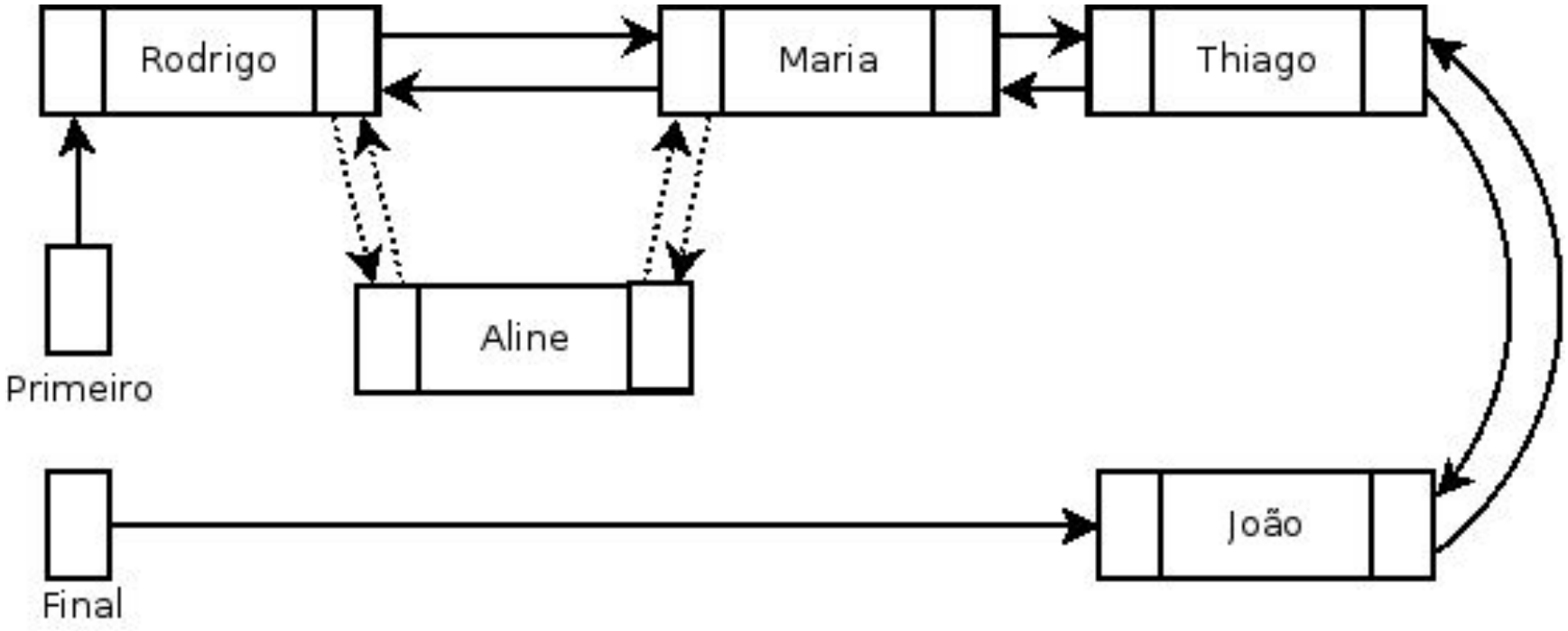
LinkedList armazena seus objetos como uma lista duplamente ligada onde cada um dos elementos adicionado é colocado dentro de outro objeto(Nó), que além de conter o valor armazenado tem uma referência ao próximo *Nó* e ao anterior.

Assim, cada *Nó* pode ser colocado em qualquer lugar da memória. Sendo esta uma vantagem do uso de **LinkedList** em comparação ao **ArrayList**.

Na **ArrayList,** os elementos são inseridos de forma sequencial, um após ao outro, o que causa perda de desempenho ao ter que aumentar o *array* ou mover muitos objetos para inserir um valor no meio da lista com esta estrutura, por exemplo.

LinkedList

Na ***LinkedList***, para inserir no início da lista ou no meio não é necessário mover todos os objetos da memória uma posição para frente, basta alterar as referências ao próximo Nó e ao anterior, tornando as operações de inserção e remoção mais eficientes.



LinkedList - Métodos

A classe **LinkedList** fornece alguns métodos além dos da interface **List,** em especial para inserir no início e no final da lista:

- **addFirst(E e)**: Insere o objeto no início da lista.
- **addLast(E e)**: Insere o objeto no final da lista.
- **getFirst()**: Retorna o primeiro objeto da lista.
- **getLast()**: Retorna o último objeto da lista.
- **removeFirst()**: Remove e retorna o primeiro objeta da lista.
- **removeLast()**: Remove e retorna o último objeto da lista.

LinkedList - Estrutura Fila

- A classe **LinkedList** também implementa a interface **Queue,** sendo assim possível trabalhar com ela como se fosse uma fila.

```
LinkedList<String> queue = new LinkedList<String>();

queue.offer("Rodrigo");
queue.offer("Luiz");
queue.offer("Maria");
queue.offer("Fernando");

System.out.println(queue);
//saida: [Rodrigo, Luiz, Maria, Fernando]

System.out.println(queue.poll());//Rodrigo
System.out.println(queue.poll());//Luiz
System.out.println(queue.poll());//Maria
System.out.println(queue.poll());//Fernando
```

Material Consultado

https://www.devmedia.com.br/diferenca-entre-arraylist-vector-e-linkedlist-em-java/29162

https://www.devmedia.com.br/java-collections-como-utilizar-collections/18450

http://www.mauda.com.br/?p=468 - Explicação de Generics