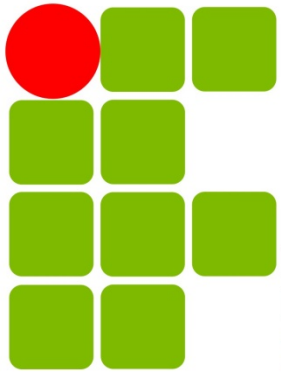


INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PARAÍBA

Collections Framework: List & Set

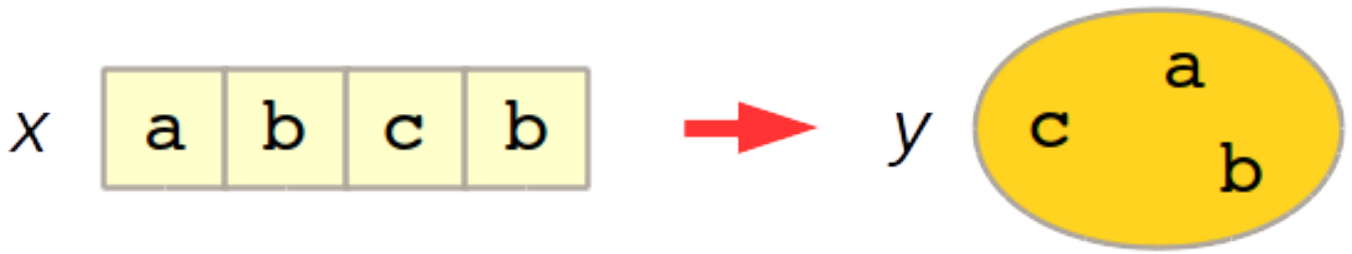
Programação Orientado a Objetos

Katyusco de F. Santos

Campina Grande/PB – 2017

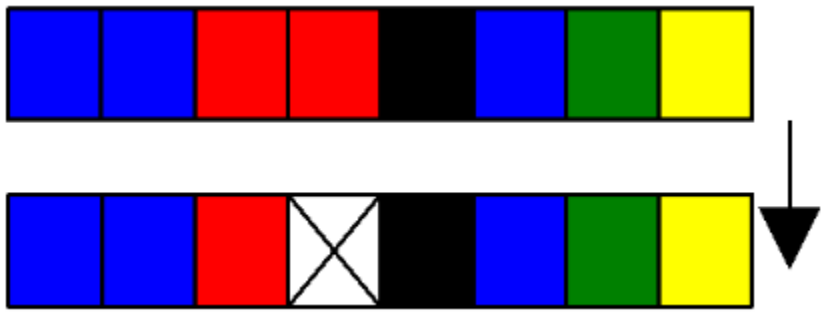
O que estudaremos nesta aula?

- O que são coleções?
- Como (e quando) utilizar lists e sets em Java?
- Como manipular estas estruturas?



Por que não utilizar arrays?

- Não é possível **redimensionar** arrays;
- Não é possível **buscar diretamente por um determinado elemento** cujo índice não se sabe;
- Não é possível **saber quantas posições do array já foram populadas** sem criar métodos auxiliares.



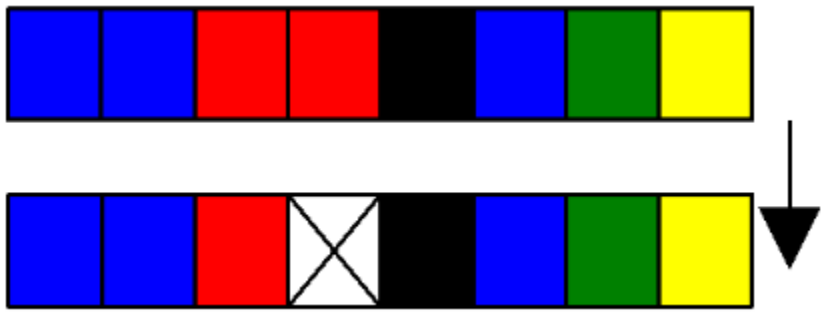
Retire a quarta Conta

conta[3] = **null**;

Por que não utilizar arrays?

- Não é possível redimensionar arrays;
- Não é pos… …nado elemento
- Não é pos… …oram populadas sem criar métodos auxiliares.

No array abaixo, o que acontece quando precisarmos inserir um novo elemento?



Retire a quarta Conta

conta[3] = **null**;

Coleções em Java

Estrutura de dados para armazenamento de objetos

Coleções em Java

Estrutura de dados para armazenamento de objetos

Mantém os dados organizados seguindo alguma lógica e disponibiliza operações para o usuário manipular os dados

Coleções em Java

Estrutura de dados para armazenamento de objetos

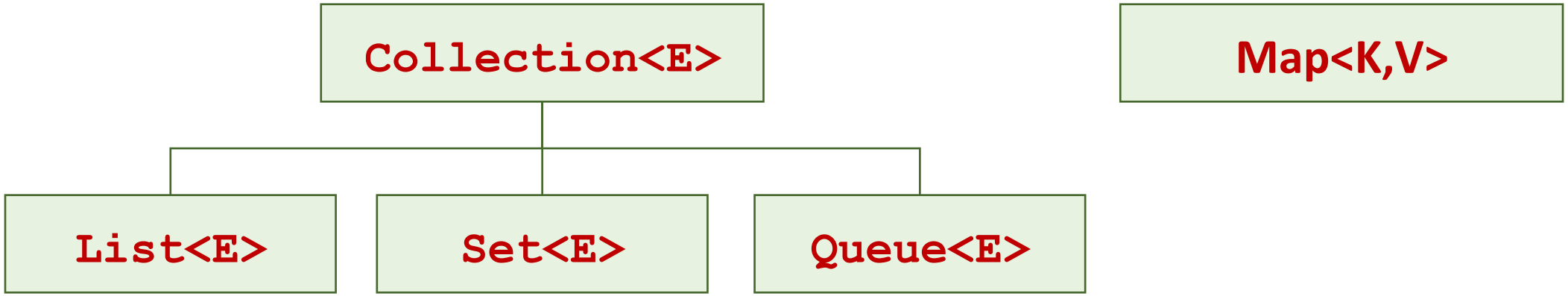
- Em Java, uma coleção é um objeto, onde é possível, de forma geral, adicionar, remover, acessar e pesquisar elementos;
- Há vários tipos de coleção: listas, conjuntos, mapas.

API para Utilizar Coleções em Java

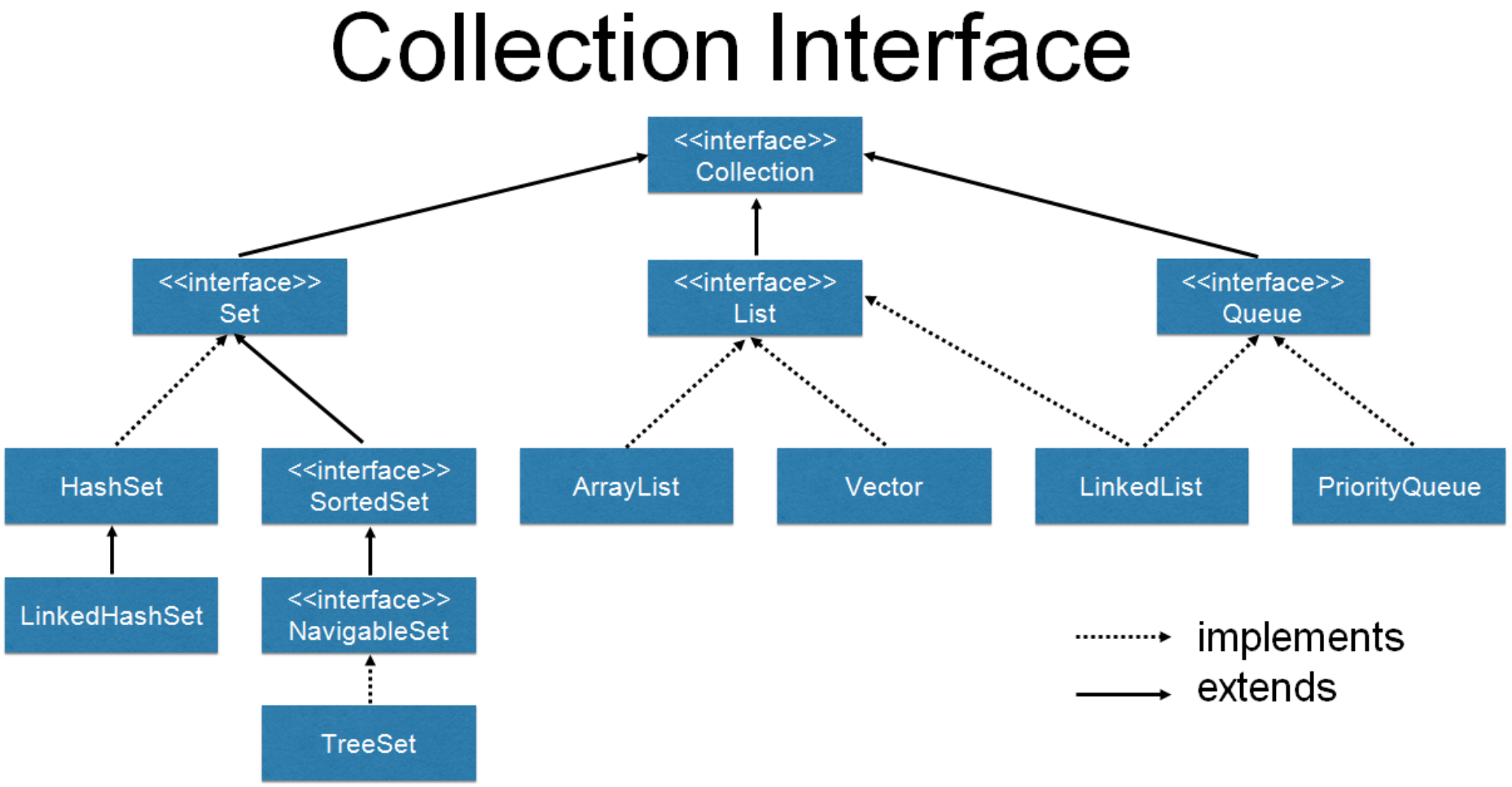
- **A API disponibiliza diversas funcionalidades para a manipulação de conjunto de dados. Principais componentes:**
  - **Interfaces**: Representam os principais tipos de coleções que a API java disponibiliza como Listas e Conjuntos;
  - **Implementação**: Classes concretas que implementam uma ou mais interfaces, que implementam algum tipo de coleção, como o ArrayList ou o HashSet;
  - **Algoritmos**: Implementação de alguns algoritmos disponíveis, como algoritmos para ordenação e busca.

Interfaces que Compõem Coleções em Java

Disponível no pacote java.util.

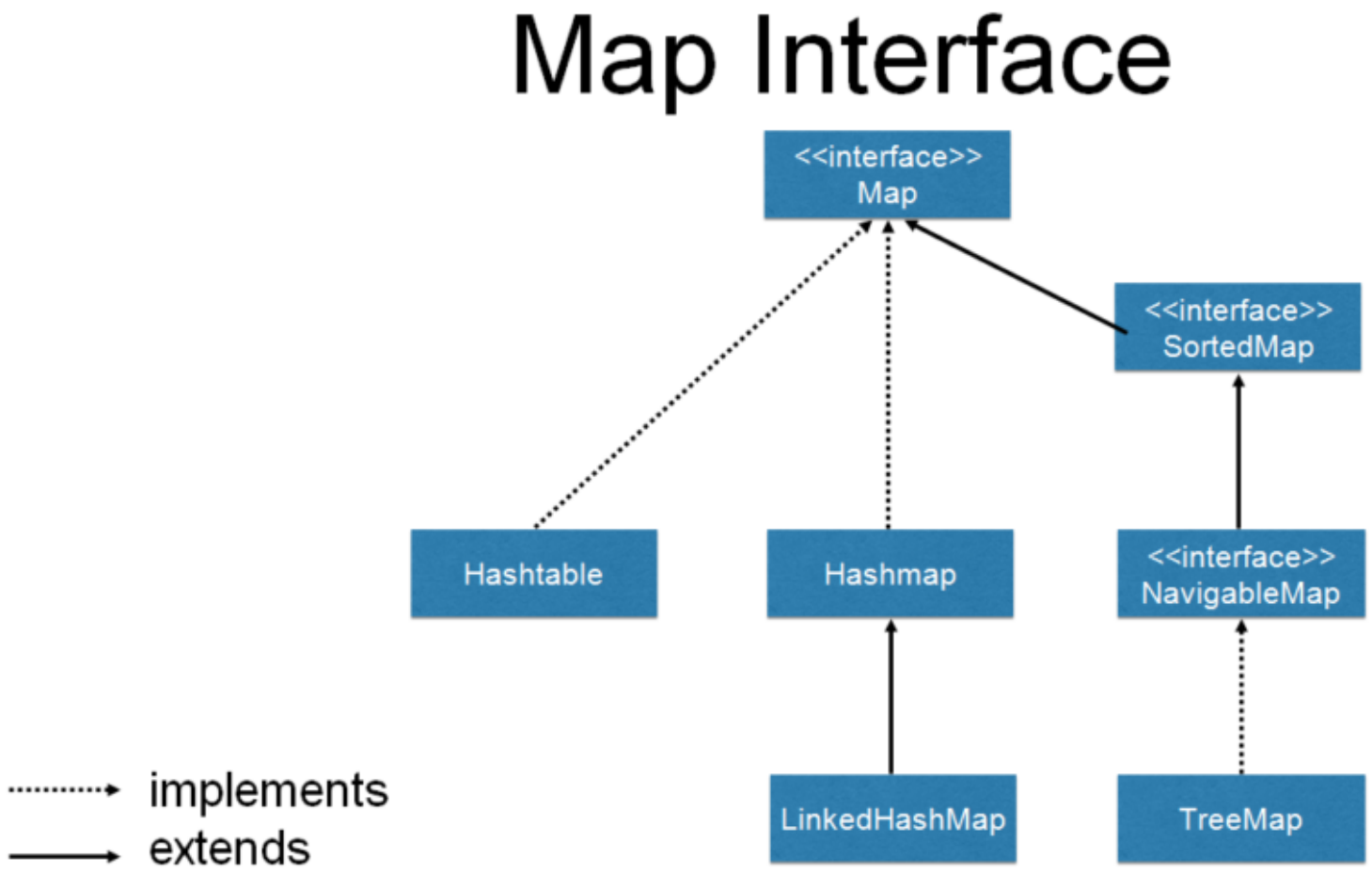


Classes Concretas sobre Collection Interface



Fonte: https://www.slideshare.net/MarcusBiel/java-collections-framework-inroduction

Classes Concretas sobre Map Interface



Fonte: https://www.slideshare.net/MarcusBiel/java-collections-framework-inroduction

Interface `java.util.List`

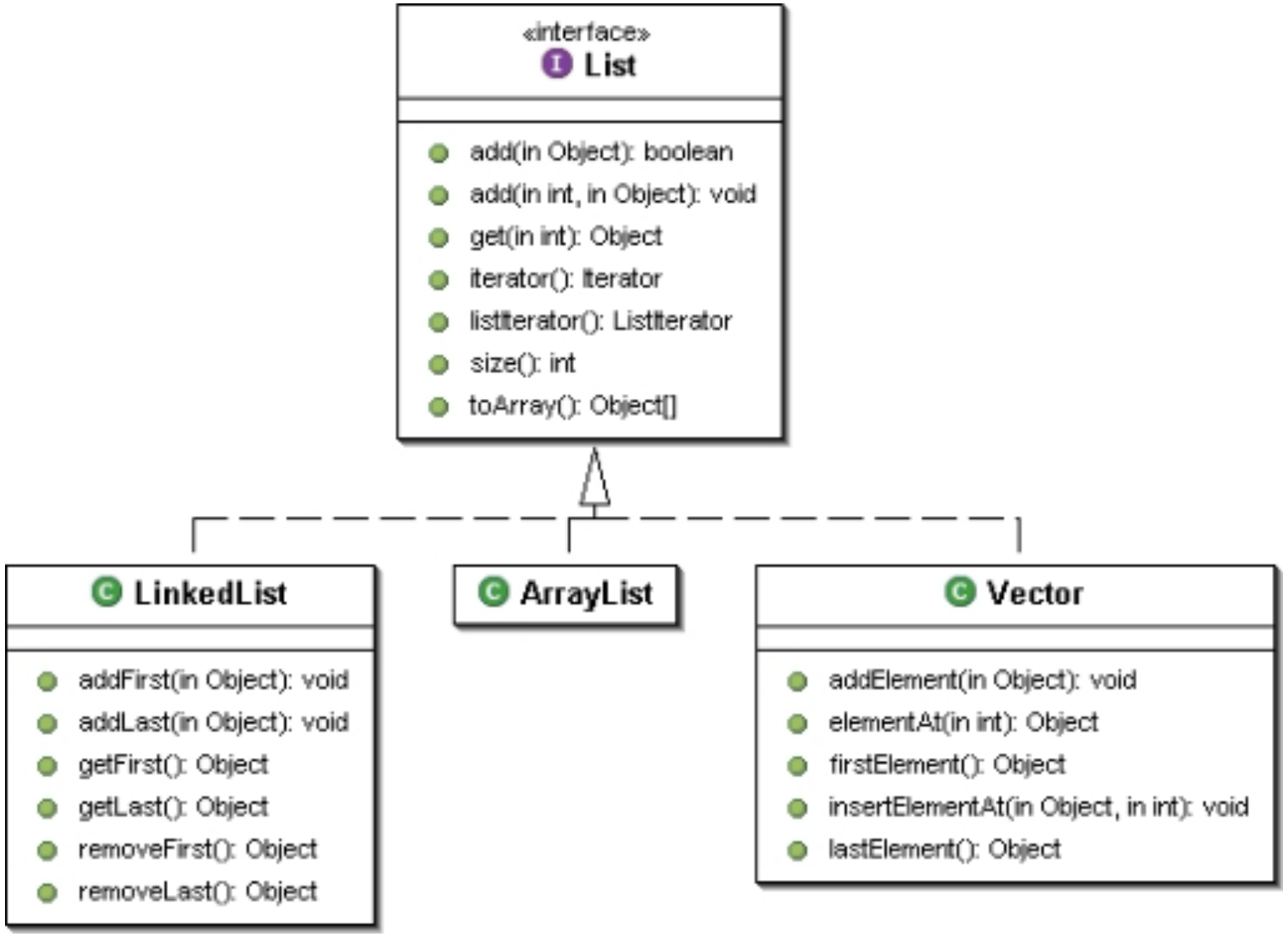
- Listas em Java
  - **Lista**: coleção organizada de forma linear que permite elementos duplicados
    - Elementos possuem uma ordem predeterminada (cada elemento possui um antecessor – com exceção do primeiro – e um sucessor – com exceção do último);
  - Ordem definida durante a inserção do elemento na lista;
  - Implementações: `ArrayList` e `LinkedList`.

Interface `java.util.List`

- **Listas em Java**
  - Lista: coleção organizada de forma linear que permite el
    - mento
  - Ordem definida durante a inserção do elemento na lista;
  - **Implementações: `ArrayList` e `LinkedList`.**

`ArrayList` é mais rápida na busca por elementos, do que a `LinkedList`, que é mais rápida na inserção

Interface `java.util.List`



Interface `java.util.List`

- **Criação de uma lista em Java**

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

ou

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

```
ArrayList<Object> lista = new ArrayList<Object>();
```

Programação Genérica (Generics)

Interface `java.util.List`

- **Manipulando uma lista em Java**

```
lista.add("Manoel");
lista.add("Joaquim");
lista.add("Maria");
```

Inserindo elementos na lista

```
lista.add(0,"Manoel");
lista.add(1,"Joaquim");
lista.add(2,"Maria");
```

Inserindo elementos em uma posição específica na lista

Interface `java.util.List`

- **Manipulando uma lista em Java**

```
lista.size();
```

Retorna o total de elementos da lista

```
lista.contains("Maria");
```

Retorna `true` se o elemento em questão está na lista; `false`, caso contrário

```
lista.get(indice);
```

Retorna o elemento presente no índice passado como parâmetro

Interface `java.util.List`

- **Manipulando uma lista em Java**

```
lista.remove("Maria");
```

Remove um objeto específico da lista (neste caso, a String “Maria”)

```
lista.remove(indice);
```

Remove o elemento presente no índice passado como parâmetro

Interface `java.util.List`

- **Manipulando uma lista em Java**

```
for (int i = 0; i < contas.size(); i++) {
    System.out.println(contas.get(i));
}
```

Escreve todos os elementos presentes na lista por ordem de inserção

Interface `java.util.List`

- Manipulando objetos de uma lista em Java

```
for (int i = 0; i < contas.size(); i++) {
    Conta conta = (Conta) contas.get(i);
    System.out.println(conta.getSaldo());
}
```

Escreve o saldo de todas as contas armazenadas na lista `contas`

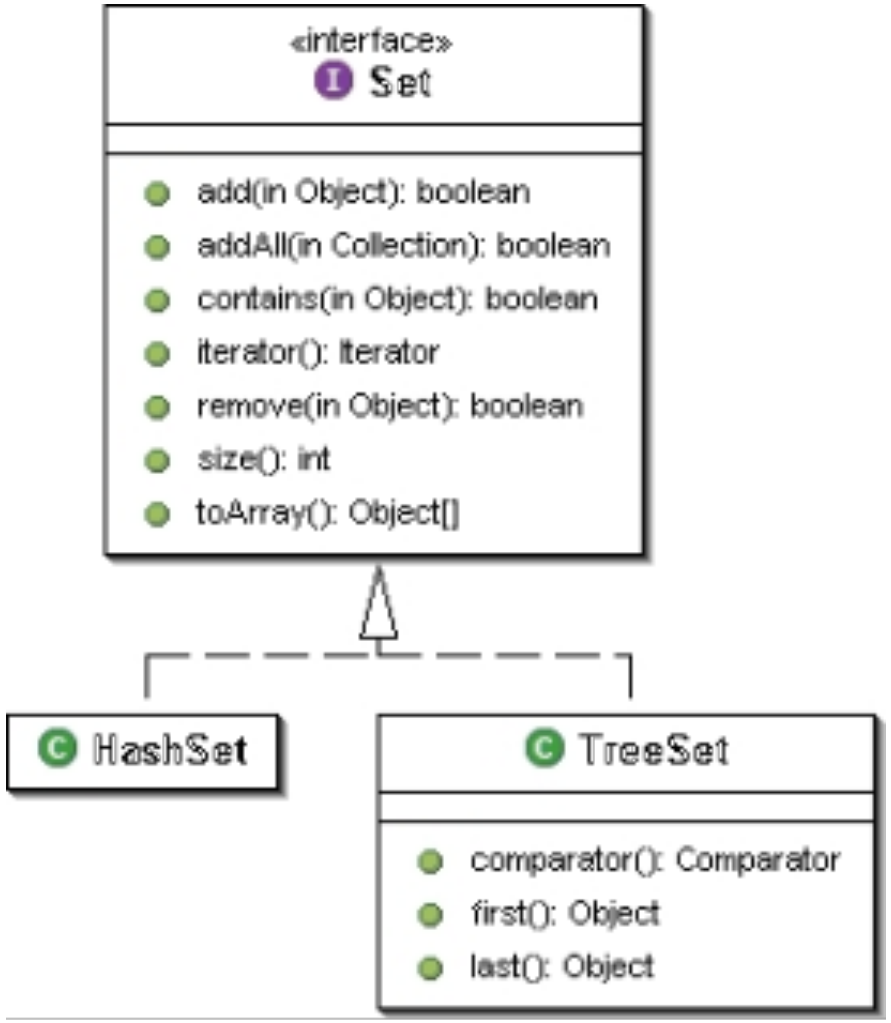
Interface `java.util.List`

- Algumas considerações sobre listas em Java
  - Algumas listas, como a ArrayList, têm acesso aleatório aos seus elementos (acesso é feito de forma imediata);
  - A `LinkedList` fornece métodos adicionais para obter e remover o primeiro e último elemento de uma lista;
  - É possível misturar objetos em uma única lista Java (contanto que se adicione o typecast adequado).

Interface `java.util.Set`

- Conjuntos em Java
  - **Conjunto**: coleção que **não** permite elementos duplicados e não há ordem pré-determinada;
  - Implementações: `HashSet`, `LinkedHashSet` e `TreeSet`.

Interface `java.util.Set`



Interface `java.util.Set`

- **Criação de um conjunto em Java**

```
Set conjunto = new HashSet();
```

ou

```
HashSet conjunto = new HashSet();
```

```
HashSet<Object> conjunto = new HashSet<Object>();
```

Programação Genérica (Generics)

Interface `java.util.Set`

- **Adicionando elementos a um conjunto**

```
conjunto.add("Manoel");
conjunto.add("Joaquim");
conjunto.add("Maria");

conjunto.add("Joaquim");
```

A segunda String "Joaquim" não será adicionada ao conjunto!

Interface `java.util.Set`

- **Algumas considerações sobre conjuntos em Java**
  - **O uso de um `Set` pode parecer desvantajoso, já que ele não armazena a ordem e não aceita elementos repetidos;**
  - **Não há métodos que trabalham com índices;**
  - **Sua vantagem está em um desempenho superior em pesquisas por elementos (e.g., `HashSet`), quando comparado com as Lists;**
  - **`TreeSets` já vêm ordenados de acordo com características estabelecidas pelo programador.**

Percorrendo Coleções em Java

- **Para evitar problemas de indexação, ao percorrer uma coleção podemos usar o enhanced-for ou o Iterator**

Lembrando que, em Sets, não existe a ideia de "ordem". Portanto, não é possível acessar elementos do conjunto via índice.

Percorrendo Coleções em Java

- Para evitar problemas de indexação, ao percorrer uma coleção podemos usar o **enhanced-for** ou o Iterator

```
Set<String> conjunto = new HashSet<String>();

conjunto.add("java");
conjunto.add("vraptor");
conjunto.add("scala");

for (String palavra : conjunto) {
  System.out.println(palavra);
}
```

Percorrendo Coleções em Java

- Para evitar problemas de indexação, ao percorrer uma coleção podemos usar o enhanced-for ou o **Iterator**

```
Set<String> conjunto = new HashSet<>(String);

conjunto.add("java");
conjunto.add("vraptor");
conjunto.add("scala");

Iterator<String> i = conjunto.iterator();
while (i.hasNext()) {
      String palavra = i.next();
      System.out.println(palavra);
}
```

A cada chamada do método `next`, o `Iterator` retorna o próximo objeto do conjunto.

Ordenando listas em Java

- A ordenação de uma lista sempre se dará por meio de um critério de ordenação
  - Este critério será utilizado para comparar os objetos presentes na lista.

O método `sort()` necessita que todos os objetos da lista sejam comparáveis

Ordenando listas em Java

- **Exemplo:**

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

lista.add("java");
lista.add("vraptor");
lista.add("scala");

System.out.println("Lista por ordem de inserção: ");
System.out.println(lista);

Collections.sort(lista);

System.out.println("Lista por ordem alfabética: ");
System.out.println(lista);
```

Ordenando listas em Java

- Para criar objetos comparáveis, é necessário implementar a interface `java.lang.Comparable`
  - Método `int compareTo(Object)`: retorna zero, se o objeto comparado for igual ao objeto dado; retorna um número negativo, se este objeto for menor que o objeto dado; e retorna um número positivo, se este objeto for maior que o objeto dado.

Ordenando listas em Java

- **Exemplo:**

```
public class ContaCorrente extends Conta
             implements Comparable<ContaCorrente> {

  public int compareTo(ContaCorrente outra) {
    if (this.saldo < outra.saldo) {
      return -1; }

    if (this.saldo > outra.saldo) {
      return 1; }

    return 0;
  }
}
```

Exercícios

1. Crie um programa para ler palavras a partir do teclado e armazená-las em uma lista (determine um critério de parada para a leitura). Após a leitura, escreva todas as palavras por ordem de leitura.
2. Altere o código do exercício anterior para armazenar as palavras lidas em um conjunto.
3. Crie a classe `Pessoa`, que deve possuir um construtor que recebe como parâmetro o seu nome e a sua idade. A classe `Pessoa` deve implementar a interface `Comparable`, de forma a ordenar as idades de uma lista com objetos Pessoa por idade (de forma crescente). OBS: Crie getters e setters.

Dicas de Leitura & Referências Utilizadas

- Apostila Java e Orientação a Objetos – Capítulo 16 Collections Framework
- Javadoc do Collection Framework
- Javadoc da interface Comparable