

The slide features a light gray background with several decorative elements: a large white circle in the top-left corner, a solid purple circle in the top-left area, a purple horizontal bar in the top-right corner, a small purple circle in the bottom-right area, a large white circle in the bottom-right corner, and a purple horizontal bar in the bottom-left corner.

Programação II: Collections

Prof^a. Tainá Isabela




Collections

Em Java, as Collections representam um conjunto de classes e interfaces que permitem armazenar e manipular grupos de objetos.

Elas fornecem recursos poderosos para **lidar com listas, conjuntos e mapas**, facilitando as operações com esses dados.

As Collections são parte da API de Coleções Java (java.util) e são essenciais para desenvolver programas eficientes e organizados.




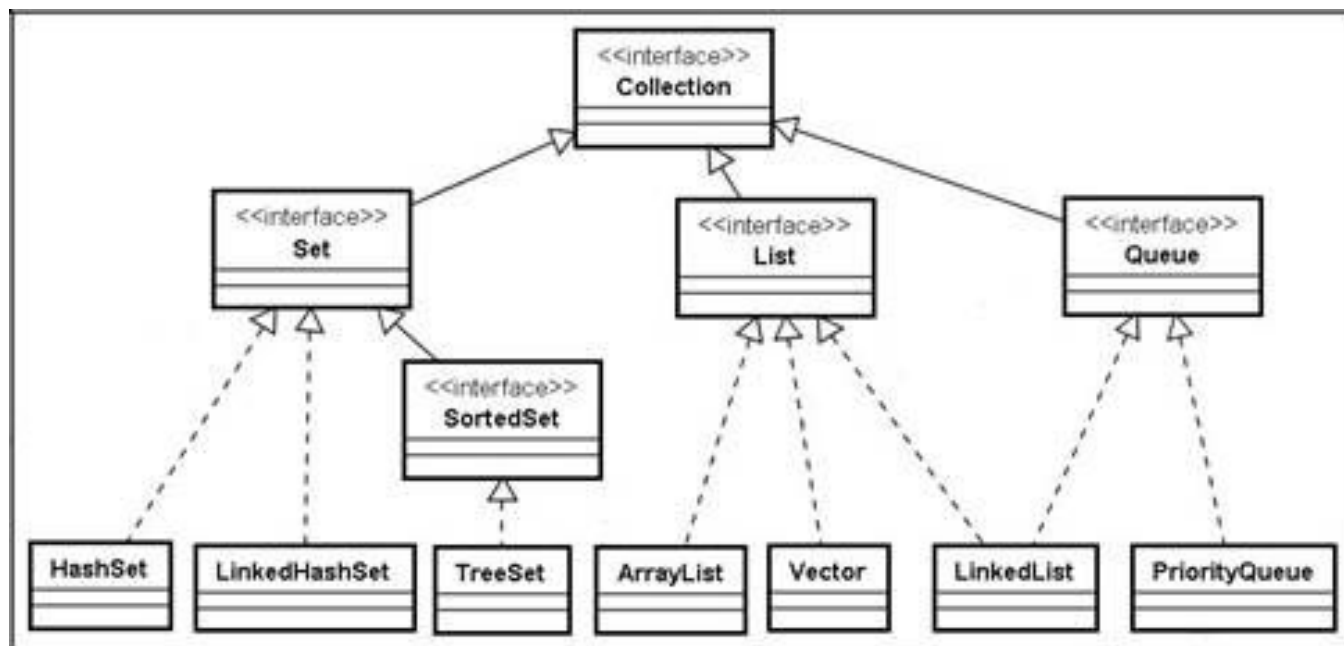
Motivação

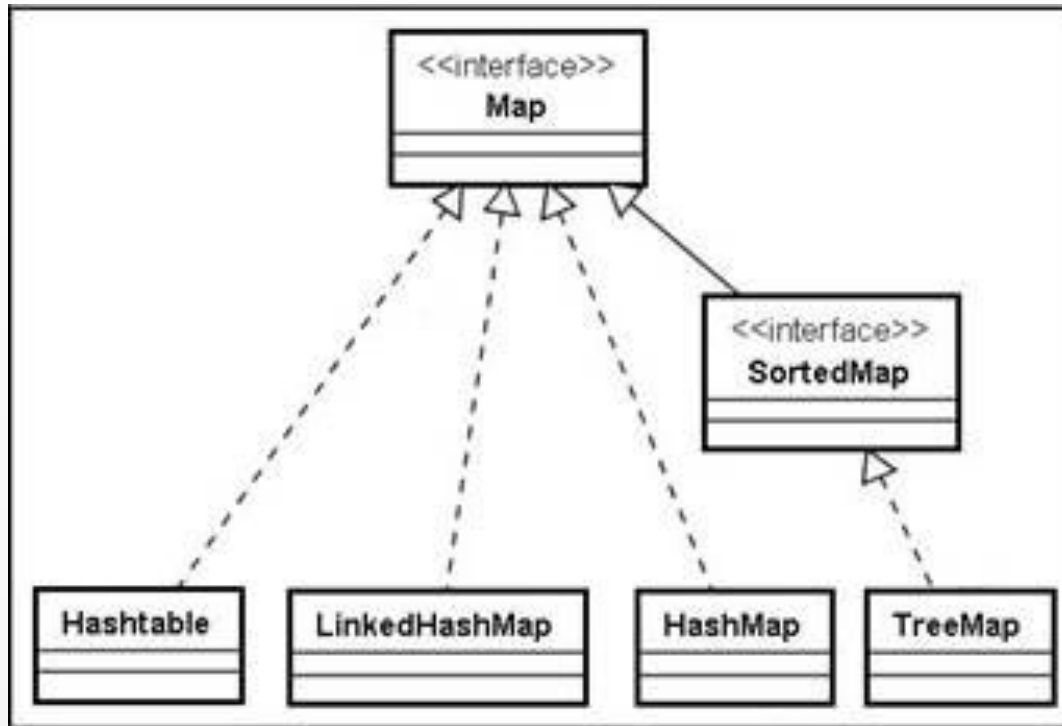
1. **Organização e Estruturação:** As Collections permitem organizar dados de forma estruturada, facilitando a busca, inserção e remoção de elementos.
2. **Algoritmos de Alta Performance:** As implementações das Collections utilizam algoritmos otimizados para realizar operações com eficiência, como busca binária, ordenação e filtragem.
3. **Reuso de Código:** As interfaces e classes genéricas das Collections permitem que você reutilize seu código para trabalhar com diferentes tipos de objetos, tornando o desenvolvimento mais modular e flexível.
4. **Segurança Tipo (Type Safety):** As Collections em Java fornecem segurança de tipo, garantindo que apenas objetos do tipo especificado possam ser adicionados à coleção, evitando erros de tipos em tempo de execução.



Collections Framework

1. **Interfaces:** tipos abstratos que representam as coleções;
 2. **Implementações:** são as implementações concretas das interfaces;
 3. **Algoritmos:** são os métodos que realizam as operações sobre os objetos das coleções, tais como busca e ordenação.
- 





Interfaces

1. **List:** Representa uma coleção ordenada, onde os elementos são acessados por um índice. As listas permitem elementos duplicados e implementações populares são o ArrayList e o LinkedList.
2. **Set:** Representa uma coleção que não permite elementos duplicados. Implementações comuns incluem o HashSet e o TreeSet.
3. **Map:** Representa uma estrutura de chave-valor, onde cada elemento é armazenado como uma combinação de uma chave única e um valor associado a essa chave. O HashMap e o TreeMap são exemplos comuns de implementações de Map.

Interfaces

4. **Queue:** um tipo de coleção para manter uma lista de prioridades, onde a ordem dos seus elementos, definida pela implementação de Comparable ou Comparator, determina essa prioridade. Com a interface fila pode-se criar filas e pilhas;

Implementações					
Interfaces	Tabela de Espalhamento	Array Redimensionável	Árvore	Lista Ligada	Tabela de Espalhamento + Lista Ligada
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Queue					
Map	HashMap		TreeMap		LinkedHashMap

ArrayList

É como um array cujo tamanho pode crescer. A busca de um elemento é rápida, mas inserções e exclusões não são. Podemos afirmar que as inserções e exclusões são lineares, o tempo cresce com o aumento do tamanho da estrutura.

Esta implementação é preferível quando se deseja acesso mais rápido aos elementos. Por exemplo, se você quiser criar um catálogo com os livros de sua biblioteca pessoal e cada obra inserida receber um número sequencial (que será usado para acesso) a partir de zero;

LinkedList

Implementa uma lista ligada, ou seja, cada nó contém o dado e uma referência para o próximo nó. Ao contrário do ArrayList, a busca é linear e inserções e exclusões são rápidas. Portanto, prefira LinkedList quando a aplicação exigir grande quantidade de inserções e exclusões.

Um pequeno sistema para gerenciar suas compras mensais de supermercado pode ser uma boa aplicação, dada a necessidade de constantes inclusões e exclusões de produtos;

HashSet

o acesso aos dados é mais rápido que em um TreeSet, mas nada garante que os dados estarão ordenados. Escolha este conjunto quando a solução exigir elementos únicos e a ordem não for importante.

Poderíamos usar esta implementação para criar um catálogo pessoal das canções da nossa discografia;

TreeSet

Os dados são classificados, mas o acesso é mais lento que em um HashSet. Se a necessidade for um conjunto com elementos não duplicados e acesso em ordem natural, prefira o TreeSet.

É recomendado utilizar esta coleção para as mesmas aplicações de HashSet, com a vantagem dos objetos estarem em ordem natural;

LinkedHashSet

É derivada de HashSet, mas mantém uma lista duplamente ligada através de seus itens. Seus elementos são iterados na ordem em que foram inseridos.

Opcionalmente é possível criar um LinkedHashSet que seja percorrido na ordem em que os elementos foram acessados na última iteração. Poderíamos usar esta implementação para registrar a chegada dos corredores de uma maratona;

HashMap

Baseada em tabela de espalhamento, permite chaves e valores null. Não existe garantia que os dados ficarão ordenados. Escolha esta implementação se a ordenação não for importante e desejar uma estrutura onde seja necessário um ID (identificador).

Um exemplo de aplicação é o catálogo da biblioteca pessoal, onde a chave poderia ser o ISBN (International Standard Book Number);

TreeMap

Implementa a interface SortedMap. Há garantia que o mapa estará classificado em ordem ascendente das chaves. Mas existe a opção de especificar uma ordem diferente.

Use esta implementação para um mapa ordenado. Aplicação semelhante a HashMap, com a diferença que TreeMap perde no quesito desempenho;

LinkedHashMap

Mantém uma lista duplamente ligada através de seus itens. A ordem de iteração é a ordem em que as chaves são inseridas no mapa. Se for necessário um mapa onde os elementos são iterados na ordem em que foram inseridos, use esta implementação.

O registro dos corredores de uma maratona, onde a chave seria o número que cada um recebe no ato da inscrição, é um exemplo de aplicação desta coleção.

Interface Iterator

As interfaces que estendem Collection herdam o método `iterator()`. Quando este método é chamado por um collection ele retorna uma interface `Iterator`.

Após essa chamada, usamos os métodos de `Iterator` para percorrer um collection do início ao fim e até remover seus elementos.

ITERATOR

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

Aluno a = new Aluno("João da Silva", "Linux básico", 0);
Aluno b = new Aluno("Antonio Sousa", "OpenOffice", 0);
Aluno c = new Aluno("Lúcia Ferreira", "Internet", 0);
lista.add(a);
lista.add(b);
lista.add(c);
lista.add(d);
System.out.println(lista);
Aluno aluno;
Iterator<Aluno> itr = lista.iterator();
while (itr.hasNext()) {
    aluno = itr.next();
    System.out.println(aluno.getNome());
}
```

Vector

Vector é uma implementação da interface List que é sincronizada e, portanto, é segura para uso em ambientes concorrentes. Ele é semelhante ao ArrayList, mas possui operações sincronizadas, o que o torna mais lento em comparação com o ArrayList em operações não sincronizadas.

É recomendado para cenários onde a **sincronização é necessária**.

LIST

```
Import java.util.ArrayList;  
Import java.util.List;
```

```
List <Integer> numeros = new ArrayList<>();  
numeros.add(10);  
numeros.add(5);  
numeros.add(20);
```

```
Int primeiroNumero = numeros.get(0);
```

```
for (int numero:numeros) {  
    System.out.println(numero);  
}
```

SET

```
import java.util.HashSet;  
import java.util.Set;
```

```
Set <String> frutas = new HashSet<>();  
frutas.add("Maçã");  
frutas.add("Banana");  
frutas.add("Laranja");  
frutas.add("Maçã");
```

```
Boolean contemMaca = frutas.contains("Maçã");
```

```
frutas.remove("laranja");
```

MAP

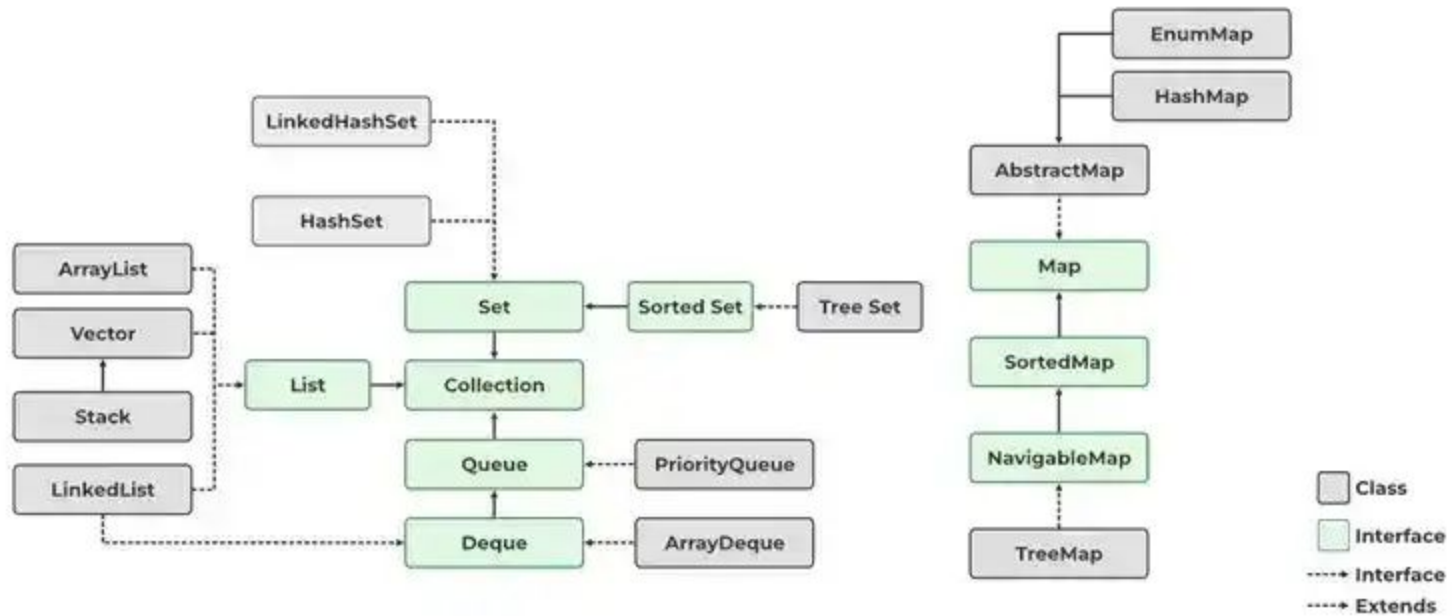
```
import java.util.HashMap;
import java.util.Map;

Set <String, Integer> idades = new HashMap<>();
frutas.add("João", 30);
frutas.add("Maria", 25);
frutas.add("Carlos", 28);

int idadeMaria = idades.get("Maria");

Boolean contemCarlos = idades.containsKey("Carlos");

idades.remove("João");
```



Momento Questionário



Atividades

1. Crie um programa em Java que: Utilize um `ArrayList<Integer>` para armazenar 10 números inteiros inseridos pelo usuário. Exiba todos os números. Calcule e mostre a soma total e a média dos números inseridos. Remova os números pares e exiba a lista atualizada.
Dica: use métodos como `add()`, `removeIf()` e loops `for` ou `for-each`.
2. Crie um programa que: Leia diversas palavras do teclado (até o usuário digitar “fim”). Armazene essas palavras em um `HashSet<String>`. Exiba todas as palavras únicas digitadas (sem repetições). Verifique se a palavra “Java” foi digitada.
Dica: use `contains()` e loops para percorrer o conjunto
3. Implemente um programa que: Utilize um `HashMap<String, Integer>` para armazenar pares nome → idade. Permita cadastrar pelo menos 5 pessoas. Peça um nome ao usuário e exiba a idade correspondente, se existir. Remova uma pessoa pelo nome e mostre o mapa atualizado.
Dica: use métodos `put()`, `get()`, `remove()` e `containsKey()`.