Compiladores

Encontro 10 – Java – Introduzindo a classe HashMap

Prática

Prof. Luiz Augusto Rodrigues
luiz.a.rodrigues@cogna.com.br

Assuntos Abordados

- Introdução
- Utilização
- Funcionamento
- Exemplos

Introdução a classe HashMap em Java

Introdução

Em Java, a manipulação de coleções de dados é uma tarefa comum no desenvolvimento de sistemas, seja para armazenar dados temporários, agrupar informações, realizar buscas ou organizar dados com critérios específicos.

Dentro desse contexto, a Java Collections Framework oferece diversas classes prontas para uso, e uma das mais versáteis é a classe HashMap.

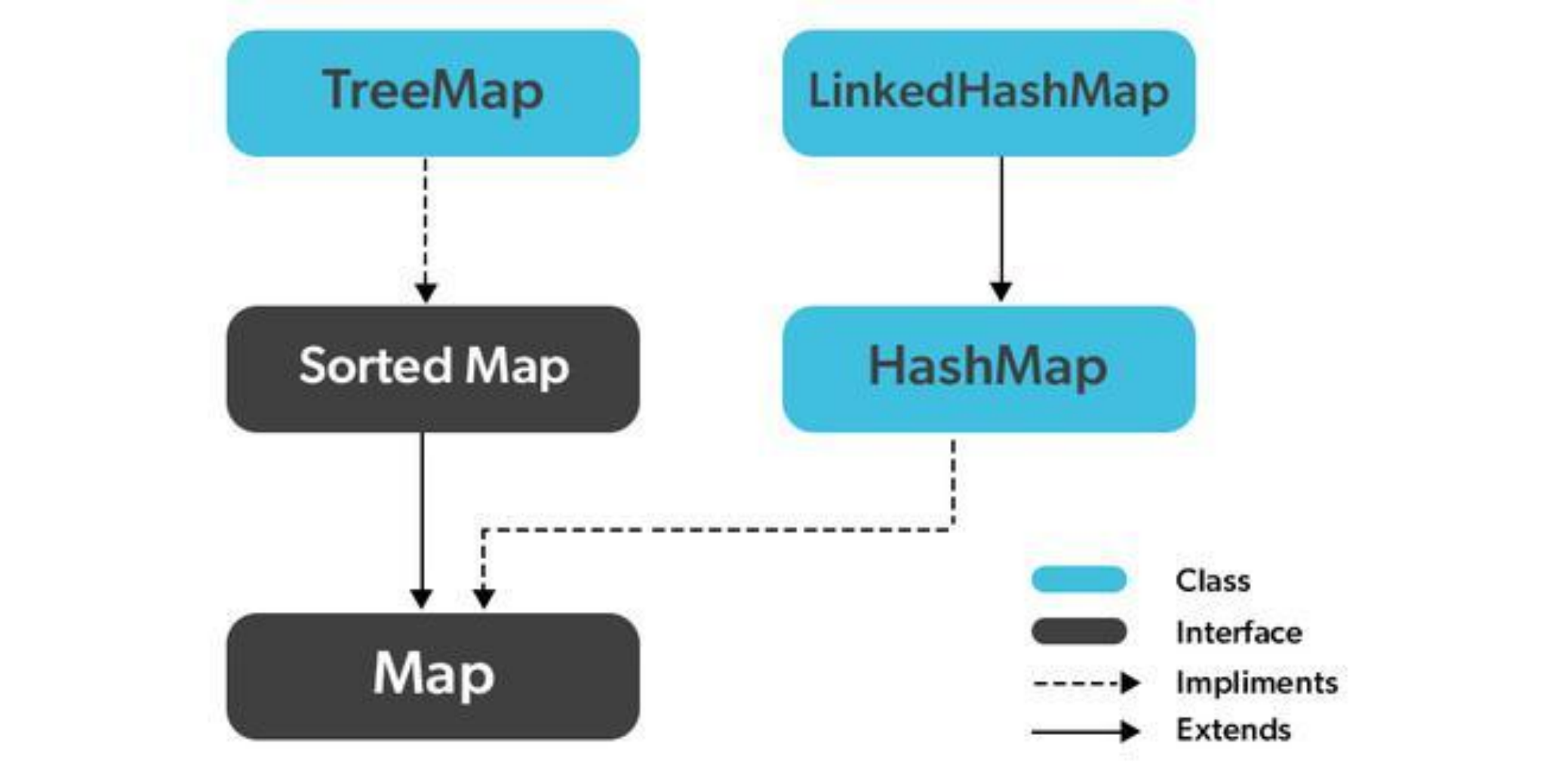
Introdução

A classe HashMap pertence ao pacote java.util e implementa a interface Map<K, V>, onde K representa o tipo da chave e V o tipo do valor.

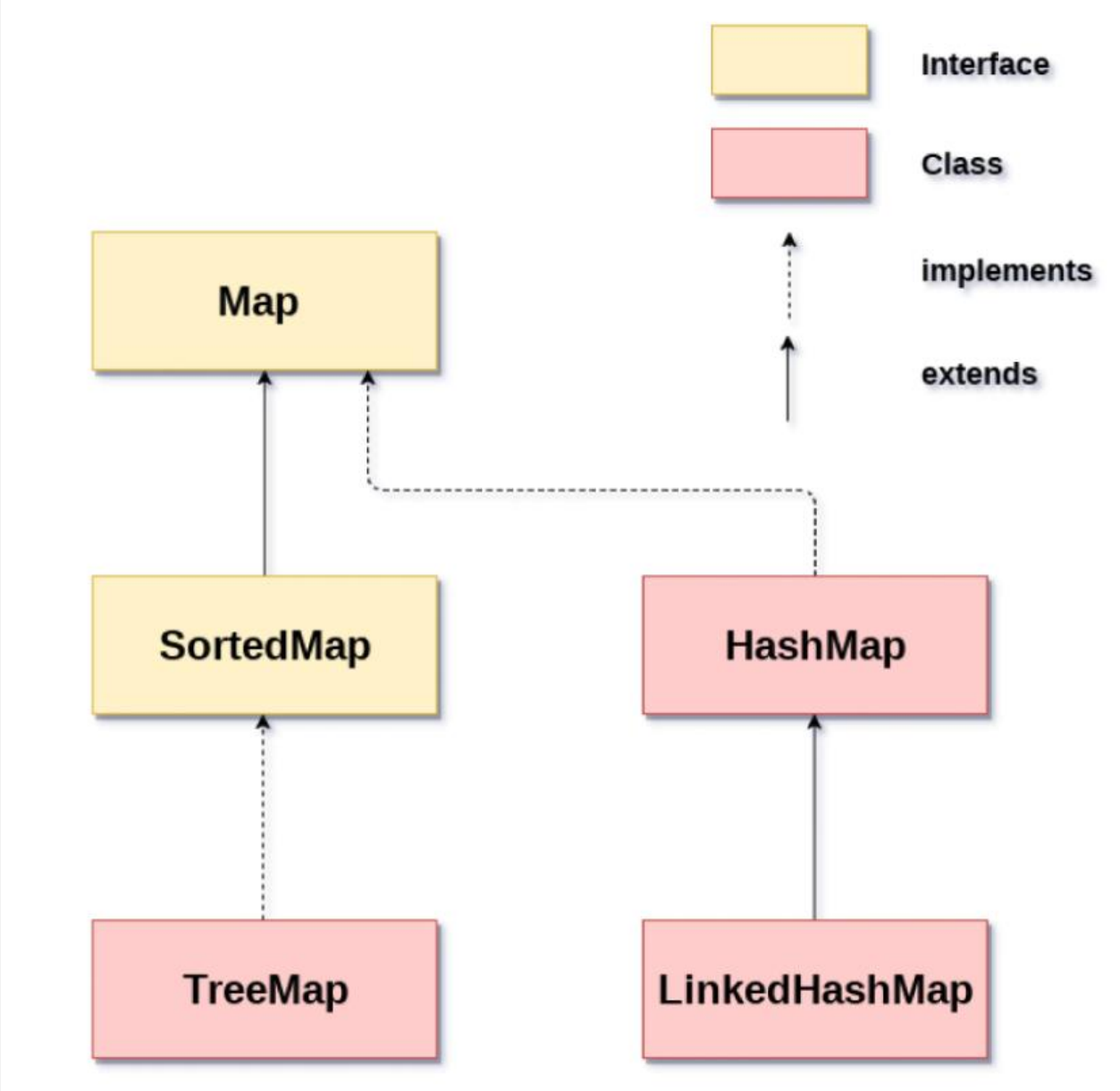
A principal função de um HashMap é armazenar pares chave-valor, permitindo a associação entre uma chave única e um valor.

É uma estrutura de dados amplamente utilizada quando é necessário acesso rápido e eficiente aos dados por meio de uma chave.

Árvore de Heranças



Árvore de Heranças



Utilização da Classe HashMap em Java

Utilização

O HashMap é baseado em uma estrutura de tabela de dispersão (hash table). Isso significa que ele utiliza uma função de hash para mapear chaves a índices em uma tabela interna, o que possibilita, na média, operações de acesso, inserção e remoção com complexidade O(1) — ou seja, muito rápidas, mesmo com grande volume de dados.

Utilização

É uma excelente escolha quando:

- Você precisa armazenar dados associados, como nome e telefone, código de produto e preço, ou login de usuário e senha;
- O acesso frequente a esses dados ocorre por meio da chave;
- A ordem dos elementos não importa (diferente, por exemplo, de uma TreeMap que mantém os pares ordenados por chave).

Funcionamento

Funcionamento

Internamente, o HashMap utiliza uma tabela onde os pares chave-valor são armazenados em buckets (compartimentos). A chave é processada por uma função de hash, que retorna um número inteiro correspondente a um índice da tabela. Esse índice determina onde o par será armazenado.

Funcionamento

Caso duas chaves diferentes gerem o mesmo índice (colisão), o HashMap resolve o problema armazenando os pares em uma lista encadeada ou estrutura semelhante dentro do bucket correspondente.

A versão atual do Java (desde Java 8) também pode usar árvores balanceadas (red-black trees) para otimizar o desempenho quando muitos elementos colidem em um mesmo bucket.

Métodos

Métodos	Descrição
put(K key, V value)	Adiciona ou atualiza um par chave-valor
get(Object key)	Retorna o valor associado à chave
remove(Object key)	Remove a entrada com a chave informada
containsKey(Object key)	Verifica se a chave existe
containsValue(Object value)	Verifica se o valor existe
size()	Retorna o número de pares armazenados
clear()	Remove todos os pares do mapa
keySet()	Retorna um conjunto com todas as chaves
values()	Retorna uma coleção com todos os valores
entrySet()	Retorna um conjunto de pares (chave/valor)

Considerações

- Chaves duplicadas não são permitidas. Se você usar put() com uma chave já existente, o valor anterior será sobrescrito.
- Valores duplicados são permitidos, ou seja, dois pares podem ter o mesmo valor, desde que as chaves sejam diferentes.
- O HashMap permite uma única chave null, e pode conter múltiplos valores null.
- Não é thread-safe: em ambientes concorrentes, deve-se usar ConcurrentHashMap ou sincronizar manualmente.

Exemplo

Imagine que você está construindo um sistema para gerenciar o estoque de uma cafeteria. Você precisa armazenar os nomes dos produtos (como "Café", "Chá", "Açúcar") e as respectivas quantidades disponíveis no estoque.

Um HashMap<String, Integer> se encaixa perfeitamente nesse cenário:

- Chave: nome do produto (String)
- Valor: quantidade em estoque (Integer)

Você pode adicionar, consultar e atualizar rapidamente a quantidade de produtos com base em seu nome.

Código

```
import java.util.HashMap;

public class ExemploHashMap {
    public static void main(String[] args) {
        HashMap<String, Integer> estoque = new
HashMap<>();

        // Adicionando produtos e quantidades
        estoque.put("Café", 50);
        estoque.put("Chá", 30);
        estoque.put("Açúcar", 20);

        // Acessando um valor
        System.out.println("Quantidade de Café: " +
estoque.get("Café"));

        // Atualizando um valor
        estoque.put("Café", 60);

        // Verificando se uma chave existe
        if (estoque.containsKey("Chá")) {
            System.out.println("Temos chá no estoque.");
        }

        // Removendo um item
        estoque.remove("Açúcar");

        // Iterando com entrySet()
        for (var entry : estoque.entrySet()) {
            System.out.println(entry.getKey() + " => " +
entry.getValue());
        }
    }
}
```

Código – saída

Quantidade de Café: 50

Temos chá no estoque.

Chá => 30

Café => 60

Prática

Encerramento

Dúvidas e sugestões, entre em contato pelo whatsapp da disciplina, ou mande um e-mail para luiz.a.rodrigues@cogna.com.br