
# Classe HashMap do Java Colletions Framework

Roteiro:

- O que é o `HashMap`?
- Como o `HashMap` funciona?
- Operações principais
- Vantagens e desvantagens
- Exemplos de uso

# O Que é e Como Funciona?

## O que é o HashMap?

O HashMap é uma implementação da interface Map no Java Collections Framework, ideal para armazenar pares de chave-valor. Cada chave é única, enquanto os valores podem ser duplicados. O HashMap é conhecido por sua flexibilidade e desempenho, tornando-o uma escolha popular em diversas aplicações Java.

## Mecanismo Interno

O HashMap funciona com base em uma tabela de hash, onde cada chave é convertida em um índice numérico através de uma função hash. Esses índices determinam a localização dos valores no array interno. Caso haja colisão, ou seja, duas chaves diferentes gerem o mesmo índice, o HashMap utiliza listas encadeadas ou árvores (a partir do Java 8) para gerenciar esses elementos.

# Operações Essenciais do HashMap

1 **put()**

Insere um novo par chave-valor no HashMap. Caso a chave já exista, o valor associado será substituído.

2 **get()**

Retorna o valor associado a uma determinada chave. Se a chave não existir, retorna null.

3 **remove()**

Remove o par chave-valor associado à chave especificada.

4 **containsKey()**

Verifica se a chave especificada já existe no HashMap, retornando true ou false.

5 **containsValue()**

Verifica se o valor especificado já existe no HashMap, retornando true ou false.

6 **size()**

Retorna o número de elementos armazenados no HashMap.

7 **isEmpty()**

Verifica se o HashMap está vazio, retornando true ou false.

8 **clear()**

Remove todos os elementos do mapa

# Exemplo

Exemplos.java

```java
import java.util.HashMap;

public class Exemplos {

    public static void main(String[] args) {
        //Adicionar elementos
        HashMap<String, Integer> mapa = new HashMap<>();
        mapa.put("A", 1);
        mapa.put("B", 2);

        System.out.println(mapa);

        //Buscar elementos
        Integer valorA = mapa.get("A");

        System.out.println(valorA);

        //Remover elementos
        mapa.remove("A");

        System.out.println(mapa);

        //Verificar a presença de elemenos
        boolean temChaveA =  mapa.containsKey("B");

        System.out.println(temChaveA);

        System.out.println(mapa.containsValue(10));

        //Obter o tamanho
        System.out.println(mapa.size());

        //Limpar o mapa
        mapa.clear();
        System.out.println(mapa);



    }
}
```

Run — Exemplos

```
"C:\Program Files\Eclipse Adoptium\j
{A=1, B=2}
1
{B=2}
true
false
1
{}

Process finished with exit code 0
```

# Vantagens do HashMap: Rapidez e Eficiência

## Acesso Rápido

O HashMap oferece acesso rápido aos seus dados, com tempo de execução médio de O(1) para operações de put() e get(). Isso o torna ideal para cenários onde a velocidade é crucial, como caches e gerenciamento de dados dinâmicos.

## Facilidade de Uso

A interface do HashMap é intuitiva e fácil de usar. Sua estrutura simples permite que você adicione, acesse e remova pares de chave-valor com facilidade, tornando-o acessível para programadores de diferentes níveis de experiência.

## Alta Performance

O HashMap é conhecido por seu desempenho excepcional, proporcionando alta velocidade para inserção, remoção e busca de dados. Sua estrutura interna de tabela de hash otimiza a busca e torna o HashMap uma escolha eficiente para diversas aplicações.

# Desvantagens do HashMap

## Ordem de Inserção

O HashMap não mantém a ordem de inserção dos elementos, ou seja, a ordem em que os elementos são adicionados ao HashMap não é garantida. Para manter a ordem, é necessário utilizar alternativas como LinkedHashMap ou TreeMap.

## Uso de Memória

O HashMap pode consumir mais memória em comparação a outras implementações de Map, devido à tabela de hash e possíveis colisões. Sua eficiência de memória pode variar dependendo da distribuição de chaves e do tamanho da tabela de hash.

## Sincronização

O HashMap não é thread-safe por padrão, o que significa que ele não é adequado para uso em ambientes multithread. É necessário usar ConcurrentHashMap ou sincronizar as operações manualmente para garantir a segurança em cenários concorrentes.

# Alternativas ao HashMap

## LinkedHashMap

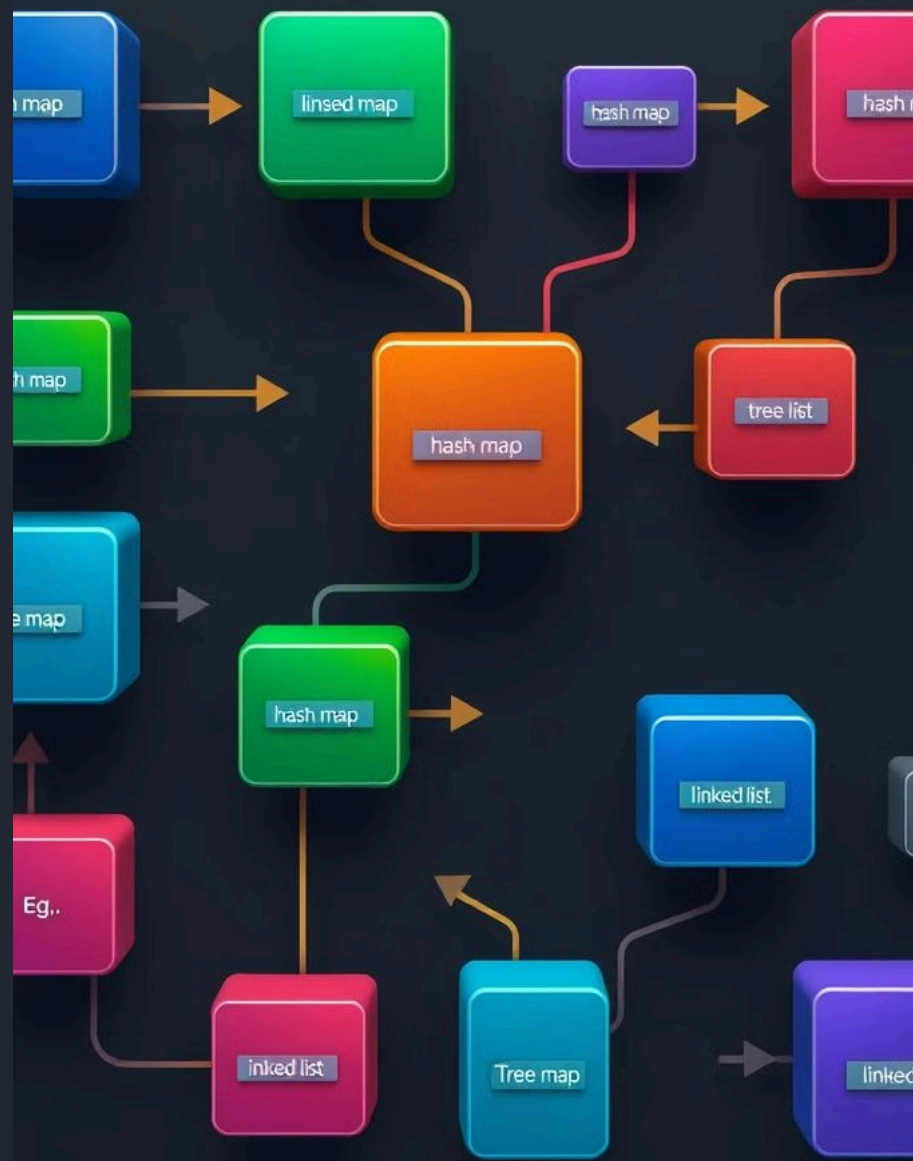
Mantém a ordem de inserção dos elementos, oferecendo flexibilidade para cenários onde a ordem é fundamental.

## TreeMap

Armazena elementos em ordem crescente das chaves, seguindo a interface SortedMap, ideal para cenários que exigem ordenação.

## ConcurrentHashMap

Uma versão thread-safe do HashMap, garantindo a segurança em ambientes multithread, ideal para aplicações concorrentes.




# Usos Práticos do HashMap: Aplicações Concretas

1. Contagem de Frequências: O HashMap permite contabilizar a ocorrência de elementos, como palavras em um texto ou itens em um inventário, usando as chaves como elementos e os valores como contagens.

2. Cache de Dados: O HashMap é ideal para armazenar dados temporários, permitindo acesso rápido a informações frequentemente usadas, otimizando o desempenho de aplicações.

3. Mapeamento de Objetos Complexos: O HashMap pode ser usado para mapear objetos ou identificadores únicos a dados relevantes, facilitando o acesso e a organização de informações complexas.

# Conclusão

O HashMap é uma ferramenta essencial para programadores Java, oferecendo um mecanismo eficiente para armazenar e acessar dados. Sua velocidade, flexibilidade e interface amigável o tornam uma escolha popular em diversas aplicações. Compreender o HashMap e suas alternativas lhe permite dominar o Java Collections Framework, abrindo novas possibilidades para suas soluções de software.