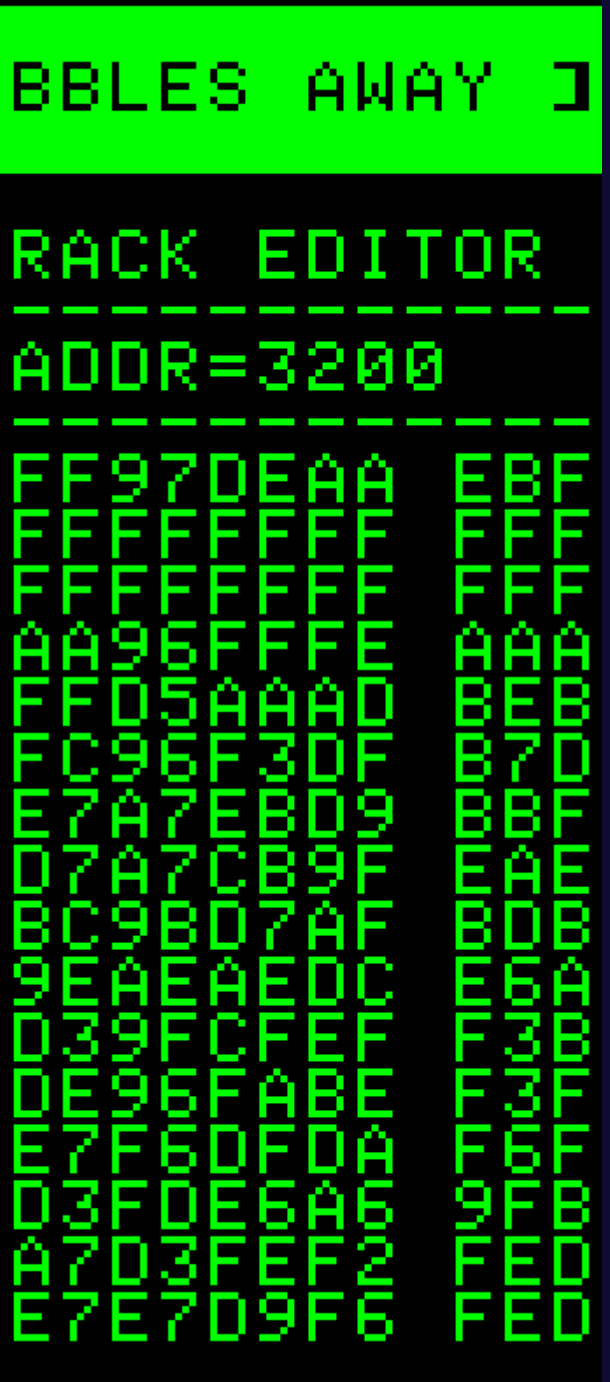
HashSet em Java

Made with Gamma



O que é HashSet?

1 Implementação da Interface Set

HashSet é uma implementação da interface Set que faz uso de uma tabela hash para armazenar os elementos. Ele garante que nenhum elemento duplicado seja armazenado e, ao mesmo tempo, não promete manter qualquer ordem entre os elementos.

2 Eficiência em Operações

Isso o torna excepcionalmente útil para operações de busca, inserção e remoção, que podem ser realizadas de maneira eficiente com complexidade temporal aproximada de O(1).

3 Permite Valor Null

HashSet permite a inserção de um valor null. É importante notar, porém, que ele não é sincronizado, o que significa que múltiplas threads que modificam um HashSet devem ser sincronizadas externamente.

Made with Gamma

Principais Características e Uso Adequado

Características Principais

Entre suas principais características, destaca-se a performance consistente para as operações de adicionar, remover e verificar a existência de um elemento, todas com complexidade temporal aproximada de O(1). HashSet também permite a inserção de um valor null.

Uso Adequado

O uso adequado do HashSet se dá em cenários onde a identificação de elementos únicos é crítica e a ordem de iteração não é relevante. É uma escolha excelente quando a eficiência e a exclusão de duplicatas são prioridades.

Made with Gamma

Implementações e Operações

1

Tabela Hash Interna

HashSet é baseado na tabela hash do HashMap, usando-a internamente para armazenar seus elementos. As operações comuns incluem add(), remove(), contains(), e size(), todas geralmente com uma complexidade temporal de O(1).

2

Função de Hash

A eficiência do HashSet depende significativamente da qualidade da função de hash utilizada para distribuir os elementos entre os "buckets" da tabela hash. Uma função de hash ideal distribui os elementos uniformemente, minimizando colisões e mantendo a complexidade temporal das operações em O(1).

3

Tratamento de Colisões

Para lidar com colisões, o HashSet usa técnicas como encadeamento separado ou endereçamento aberto para garantir que todos os elementos sejam armazenados corretamente e as operações sejam executadas eficientemente.

Made with Gamma

Complexidade Temporal e Interna

Conceito	Definição
Operações O(1)	Operações como adicionar, remover e verificar a existência de elementos têm um tempo de execução constante, não importa quantos elementos o conjunto contenha.
Bucket	Uma unidade de armazenamento onde os elementos são agrupados com base no valor de sua função de hash.
Colisões	Ocorre quando dois ou mais elementos são mapeados para o mesmo "bucket" na tabela hash.

Made with Gamma

Técnicas de Tratamento de Colisões

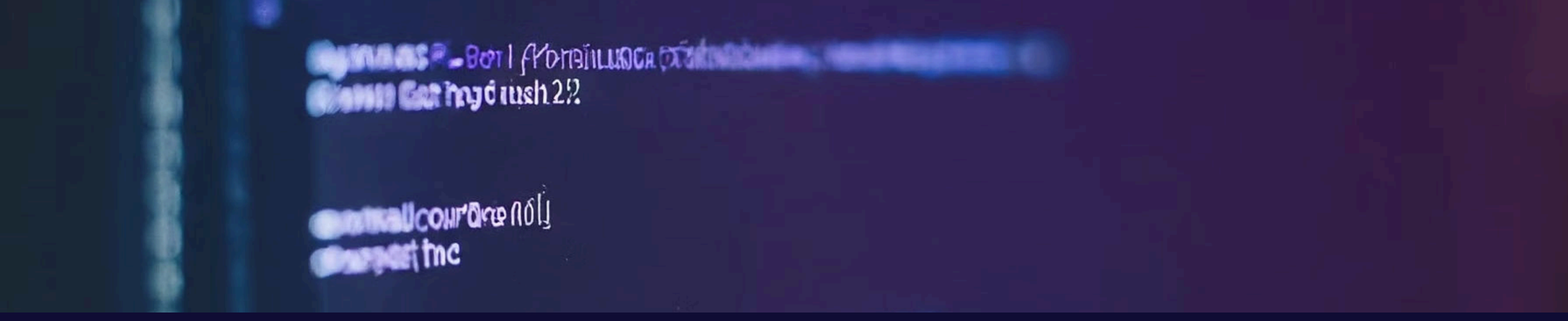
Encadeamento Separado

Cada "bucket" na tabela hash é uma lista ligada. Quando ocorre uma colisão, o elemento é adicionado à lista correspondente ao "bucket" afetado.

Endereçamento Aberto

Quando ocorre uma colisão, o algoritmo de hashing tenta encontrar o próximo "slot" vazio na tabela hash para armazenar o elemento, usando diferentes estratégias de sondagem até encontrar um "slot" disponível.

Made with Gamma



Exemplos de Código

```
Add()
HashSet mySet = new HashSet<>();
conjunto.add("Java");
conjunto.add("Python");
conjunto.add("Java"); // Tentativa de adicionar duplicata
System.out.println(conjunto); // Saída: [Java, Python]

Remove()
HashSet<String> conjunto = new HashSet<>();
conjunto.add("Maçã");
conjunto.add("Banana");
conjunto.add("Laranja");

conjunto.remove("Banana"); // Remove a string "Banana" do conjunto
System.out.println(conjunto); // Saída: [Maçã, Laranja]

Contains()
HashSet<String> conjunto = new HashSet<>();
conjunto.add("Relógio");
conjunto.add("Celular");
conjunto.add("TV");

if (conjunto.contains("TV")) {
    System.out.println("O conjunto contém a string 'TV'.");
} else {
    System.out.println("O conjunto não contém a string 'TV'.");
}


Size()
HashSet<String> conjunto = new HashSet<>();
conjunto.add("Maçã");
conjunto.add("Banana");
conjunto.add("Laranja");

int tamanho = conjunto.size();
System.out.println("Tamanho do conjunto: " + tamanho);
```

Este exemplo demonstra a natureza única do HashSet e como as duplicatas são automaticamente ignoradas. Ele cria um novo HashSet, adiciona dois elementos, tenta adicionar um elemento duplicado (que é ignorado) e, em seguida, imprime o conjunto resultante.

Made with Gamma

Operações Comuns no HashSet

Adicionar Elemento

Use add() para adicionar um elemento ao HashSet. Se o elemento já existir, ele não será adicionado novamente.

Remover Elemento

Use remove() para excluir um elemento. Se o elemento não estiver presente, não ocorrerá nenhuma ação.

Verificar Existência

contains() verifica se um elemento está presente, retornando true se sim e false se não.

Obter Tamanho

size() fornece o número total de elementos no HashSet.

Made with Gamma

Boas Práticas e Dicas

1 Implementar hashCode() e equals()

Ao trabalhar com HashSet, implementar corretamente hashCode() e equals() é crucial para otimizar o desempenho.

2 Considerar Capacidade Inicial

Considerar a capacidade inicial e o fator de carga ao instanciar um HashSet pode também melhorar a eficiência.

3 Sincronização Externa

Como HashSet não é sincronizado, múltiplas threads que modificam um HashSet devem ser sincronizadas externamente para evitar condições de corrida.

Made with Gamma

Conclusão

HashSet é uma ferramenta poderosa do Java Collections Framework, ideal para gerenciar conjuntos de elementos únicos. Compreender suas características, operações e práticas recomendadas é essencial para maximizar a eficácia dessa coleção em seus projetos de software.

Com sua eficiência comprovada em operações de busca, inserção e remoção, o HashSet se destaca como uma escolha excelente quando a identificação de elementos únicos é uma prioridade. Ao dominar os conceitos fundamentais e as técnicas avançadas apresentadas hoje, você estará bem equipado para tirar o máximo proveito dessa estrutura de dados versátil e poderosa.

Made with Gamma