TREEMAP

GRUPO 6:

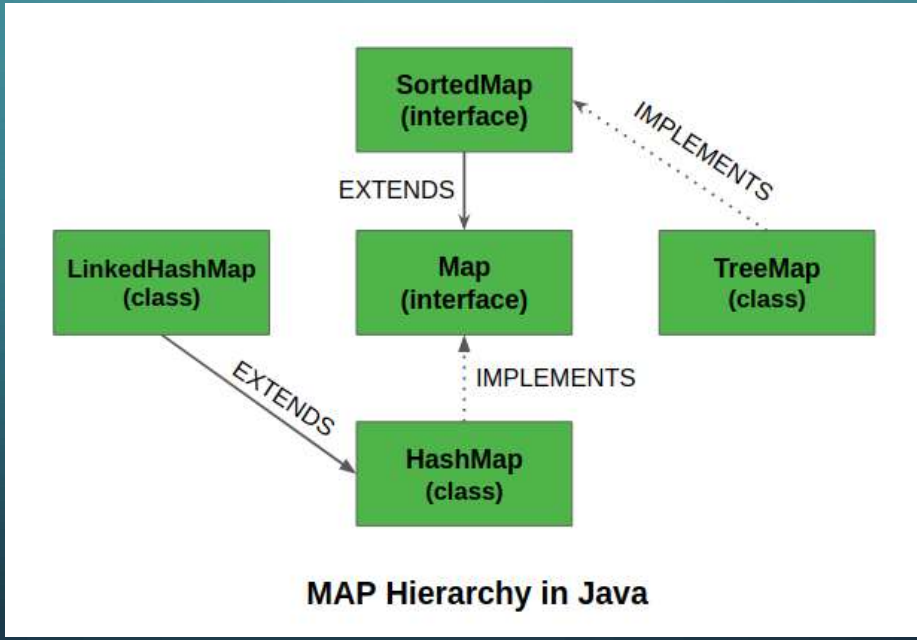
CASSIANO DUARTE

LUCAS IMROTH

WELLERSON BARAUNA

O QUE É O TREEMAP ?

O TreeMap é uma forma de buscar e organizar dados em uma lista, permitindo que o usuário localize informações rapidamente com base em chaves únicas.



MAP Hierarchy in Java

PRINCIPAIS CARACTERÍSTICAS

Ordenação automática	• Ordenação baseado em chave: mantidos em ordem crescente.
Eficiência em operação	• As operações realizadas possuem complexidade logarítmica.
Chaves únicas	• Cada chave dentro de um TreeMap deve ser única.
Arvore rubro-negra	• Arvore binária de busca balanceada.

USO ADEQUADO

Ordenação:	• Para cenários de armazenamento em uma ordem especifica.
Recuperação eficiente	• Recuperação de subconjuntos de dados em um intervalo de chaves.
Eficiência de tempo não critica	• Complexidade de tempo O(log(n))

OPERAÇÕES E COMPLEXIDADE TEMPORAL

O(Log n)

- Inserção (put): Adiciona um par chave-valor.
- Remoção(remove): Remove um elemento.
- Busca(get): Recupera um valor associado a uma chave especifica.

O(1)

- Obter o menor elemento(firstkey): retorna a menor chave.
- Obter o maior elemento (lastKey): Retorna a maior chave.

IMPLEMENTAÇÕES

Implementação	Características
`java.util.TreeMap`	Implementação padrão
`com.google.guava.collect.Maps.newTreeMap()`	Usa uma árvore vermelha-preta, oferece recursos adicionais como comparador personalizado
`org.apache.commons.collections4.map.Treemap`	Usa uma árvore vermelha-preta, oferece recursos adicionais como comparador personalizado e serialização

EXEMPLO DE CÓDIGO

```java
import java.util.Map;
import java.util.TreeMap;

public class Main {
    public static void main(String[] args) {
        // Criando um TreeMap
        TreeMap<Integer, String> treeMap = new TreeMap<>();

        // Adicionando elementos ao TreeMap
        treeMap.put(1, "Valor1");
        treeMap.put(2, "Valor2");
        treeMap.put(3, "Valor3");
```

```
Conteúdo inicial do TreeMap:
Chave: 1, Valor: Valor1
Chave: 2, Valor: Valor2
Chave: 3, Valor: Valor3
```

EXEMPLO DE CÓDIGO

o

```
Conteúdo inicial do TreeMap:
Chave: 1, Valor: Valor1
Chave: 2, Valor: Valor2
Chave: 3, Valor: Valor3
```

```java

    // Imprimindo todos os valores do TreeMap
    System.out.println("Conteúdo inicial do TreeMap:");

    for (Integer key : treeMap.keySet())
        System.out.println("Chave: " + key + ", Valor: " + treeMap.get(key));
    /*Este trecho usa o método keySet() para obter um conjunto de todas
     as chaves no TreeMap. Em seguida, ele itera sobre essas chaves. Para
      cada chave, ele usa treeMap.get(key) para obter o valor correspondente.*/

```

```java
for (Map.Entry<Integer,String> entry : treeMap.entrySet())
    System.out.println("Chave: " + entry.getKey() + ", Valor: " + entry.getValue());
/*Este trecho usa o método entrySet() para obter um conjunto de todas as
 entradas (pares chave-valor) no TreeMap. Em seguida, ele itera sobre
 essas entradas. Para cada entrada, ele usa entry.getKey() para obter
 a chave e entry.getValue() para obter o valor.*/

```

EXEMPLO DE CÓDIGO

```java
        // Obtendo um valor pela chave
        String valor = treeMap.get(2);
        // pega o valor da chave especificada e retorna null se não existir;

        String valor2 = treeMap.getOrDefault( key: 5,  defaultValue: "valor padrao");
        // pega o valor da chave especificada, se nao existir retorna o valor padrao passado

        String valor3 = treeMap.computeIfAbsent( key: 4, k -> "Valor para " + k);
        /* pega o valor e se ele não existir faz uma função lambda que imprime
         a chave, existem outras maneiras de fazer a mesma coisa mas menos eficientes.*/

        System.out.println("\nValor obtido pela chave 2: " + valor3);
```

```
Valor obtido pela chave 2: Valor2
```

```
Valor obtido pela chave 5: valor padrao
```

```
Valor obtido pela chave 4: Valor para 4
```

```
Chave: 1, Valor: Valor1
Chave: 3, Valor: Valor3
Chave: 4, Valor: Valor para 4
```

EXEMPLO DE CÓDIGO

```java
        // Removendo um elemento do TreeMap
        treeMap.remove( key: 2);

        // Imprimindo todos os valores do TreeMap após a remoção
        System.out.println("\nConteúdo do TreeMap após a remoção:");

        for (Map.Entry<Integer,String> entry : treeMap.entrySet())
            System.out.println("Chave: " + entry.getKey() + ", Valor: " + entry.getValue());
        /*Este trecho usa o método entrySet() para obter um conjunto de todas as
         entradas (pares chave-valor) no TreeMap. Em seguida, ele itera sobre
         essas entradas. Para cada entrada, ele usa entry.getKey() para obter
         a chave e entry.getValue() para obter o valor.*/

        // Removendo todos os elementos do TreeMap
        treeMap.clear();

        // Verificando se o TreeMap está vazio
        if (treeMap.isEmpty())
            System.out.println("\nO TreeMap está vazio após a remoção de todos os elementos.");
```

```
Conteúdo inicial do TreeMap:
Chave: 1, Valor: Valor1
Chave: 2, Valor: Valor2
Chave: 3, Valor: Valor3

Valor obtido pela chave 2: Valor2

Conteúdo do TreeMap após a remoção:
Chave: 1, Valor: Valor1
Chave: 3, Valor: Valor3

O TreeMap está vazio após a remoção de todos os elementos.

Process finished with exit code 0
```

EXEMPLO DE CÓDIGO

```java
import java.util.Map;
import java.util.TreeMap;

public class Main {
    public static void main(String[] args) {
        // Criando um TreeMap
        TreeMap<Integer, String> treeMap = new TreeMap<>();

        // Adicionando elementos ao TreeMap
        treeMap.put(1, "Valor1");
        treeMap.put(2, "Valor2");
        treeMap.put(3, "Valor3");


        // Imprimindo todos os valores do TreeMap
        System.out.println("Conteúdo inicial do TreeMap:");

        for (Integer key : treeMap.keySet())
            System.out.println("Chave: " + key + ", Valor: " + treeMap.get(key));
        /*Este trecho usa o método keySet() para obter um conjunto de todas
         as chaves no TreeMap. Em seguida, ele itera sobre essas chaves. Para
          cada chave, ele usa treeMap.get(key) para obter o valor correspondente.*/

        // Obtendo um valor pela chave
        String valor = treeMap.get(2);
        // pega o valor da chave especificada e retorna null se não existir;

        String valor2 = treeMap.getOrDefault( key: 5,  defaultValue: "valor padrao");
        // pega o valor da chave especificada, se nao existir retorna o valor padrao passado

        String valor3 = treeMap.computeIfAbsent( key: 4, k -> "Valor para " + k);
        /* pega o valor e se ele não existir faz uma função lambda que imprime
         a chave, existem outras maneiras de fazer a mesma coisa mas menos eficientes.*/

        System.out.println("\nValor obtido pela chave 4: " + valor3);


        // Removendo um elemento do TreeMap
        treeMap.remove( key: 2);

        // Imprimindo todos os valores do TreeMap após a remoção
        System.out.println("\nConteúdo do TreeMap após a remoção:");

        for (Map.Entry<Integer,String> entry : treeMap.entrySet())
            System.out.println("Chave: " + entry.getKey() + ", Valor: " + entry.getValue());
        /*Este trecho usa o método entrySet() para obter um conjunto de todas as
         entradas (pares chave-valor) no TreeMap. Em seguida, ele itera sobre
         essas entradas. Para cada entrada, ele usa entry.getKey() para obter
         a chave e entry.getValue() para obter o valor.*/

        // Removendo todos os elementos do TreeMap
        treeMap.clear();

        // Verificando se o TreeMap está vazio
        if (treeMap.isEmpty())
            System.out.println("\nO TreeMap está vazio após a remoção de todos os elementos.");
    }
}
```

BOAS MANEIRAS

- Escolha adequada de chaves
- Evitar operações desnecessárias de cópia
- Atenção á complexibilidade do tempo
- Evitar modificar as chaves após a inserção.