Collections

LinkedHashMap

Henry Fillvock, Júlio Cordeiro,
Luiz Berger e Thiago Fontanella.

Universidade
POSITIVO

Definição

- a classe LinkedHashMap é um tipo de estrutura de dados em Java que extende HashMap com a capacidade de manter a ordem de inserção dos elementos.

Declaração

```
LinkedHashMap<K, V> linkedHashMap = new LinkedHashMap<K, V>();
```

- K - tipo das chaves do mapa.
- V - tipo dos valores mapeados.

Características

- Utilizado quando há necessidade de trabalhar com a ordem em que os elementos foram inseridos;
- Implementa a interface Map e extende a classe HashMap;
- Contém valores armazenados em cada chave;
- Pode ter uma chave nula e diversos valores nulos;
- Não-sincronizada.

Implementações

Mapeamento de chaves e valores

- Adicionar pares chave-valor com put:

```
linkedHashMap.put("Maçã", 10);
linkedHashMap.put("Banana", 20);
linkedHashMap.put("Laranja", 15);
```

Implementações

Recuperação de valores

- Recuperar valor da chave por get:

```
Integer quantidadeMaçã = linkedHashMap.get("Maçã");
```

Implementações

Iteração de valores

- Iterando sobre as entradas do LinkedHashMap com o método entrySet, que retorna um Set dos mapeamentos contidos no mapa:

```
for (Map.Entry<String, Integer> entry : linkedHashMap.entrySet()) {
    System.out.println("Chave: " + entry.getKey() + ", Valor: " + entry.getValue());
}
```

Implementações

Verificação de existência de chave

- Verificando se uma chave já existe com containsKey:

```
boolean contemBanana = linkedHashMap.containsKey("Banana");
System.out.println("Contém Banana? " + contemBanana);
```

Implementações

Verificação de existência de valor

- Verificando se um valor existe com containsValue:

```
boolean existeValor = linkedHashMap.containsValue(20);
System.out.println("Existe o valor 20? " + existeValor);
```

Implementações

Remoção de elemento

- Removendo um elemento com remove:

```
linkedHashMap.remove("Laranja");
```

Implementações

Obtenção do tamanho

- Obtendo o número de pares chave-valor com o método size:

```
int tamanho = linkedHashMap.size();
```

Complexidade temporal

Operação	Complexidade	*
Inserção	O(1)	O(n) caso seja necessário redimensionar a tabela
Recuperação	O(1)	
Remoção	O(1)	
Verificação de Existência	O(1)	em média
Iteração	O(n)	

Exemplo de uso

```java
import java.util.LinkedHashMap;
import java.util.Map;

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

        // Adicionando elementos
        linkedHashMap.put("Maçã", 5);
        linkedHashMap.put("Banana", 3);
        linkedHashMap.put("Laranja", 7);
        linkedHashMap.put("Morango", 10);

        // Acessando um elemento por chave
        int quantidadeBanana = linkedHashMap.get("Banana");
        System.out.println("Quantidade de Bananas: " + quantidadeBanana);

        // Removendo um elemento
        linkedHashMap.remove("Laranja");

        // Verificando a existência de uma chave
        boolean contemMorango = linkedHashMap.containsKey("Morango");
        System.out.println("Contém Morango? " + contemMorango);

        // Iterando sobre as entradas
        for (Map.Entry<String, Integer> entry : linkedHashMap.entrySet()) {
            System.out.println("Fruta: " + entry.getKey() + ", Quantidade: " + entry.getValue());
        }
    }
}
```

Boas práticas e dicas

- Muito útil em situações onde a ordem dos elementos é importante, como em implementações de listas ordenadas, históricos de ações, entre outros;
- Oferece acesso rápido e eficiente aos elementos por meio de chaves únicas. Especialmente útil para um grande número de elementos;
- Pode ser utilizado para iterar tanto por ordem de inserção quanto de acesso;

Boas práticas e dicas

- Caso não haja necessidade de manter a ordem dos elementos, a aplicação de um LinkedHashMap é redundante, e pode ser evitada para economizar memória.
- Pode ser sincronizado para evitar acesso mútuo ao sincronizar o objeto que o encapsula, ou aplicar um *wrap* utilizando o método Collections.synchronizedMap:

```
Map mapa = Collections.synchronizedMap(new LinkedHashMap(...));
```

Exercícios

> 1. Desenvolva um programa Java para gerenciar a reserva de assentos em um voo. O programa exibirá uma lista numerada de assentos disponíveis (de 1 a 10). Os usuários podem escolher um assento digitando seu número ou sair digitando 0. Ao escolher um assento disponível, ele será marcado como reservado. Utilize um LinkedHashMap para armazenar os assentos, onde a chave é o número do assento e o valor indica se está disponível (true) ou reservado (false):

Exercícios

> 2. Crie um programa Java para simular uma agenda telefônica. Utilize um LinkedHashMap para armazenar os contatos, onde o nome do contato é a chave e o número de telefone é o valor.

Exercícios

> 3. Escreva um programa Java que recebe uma frase como entrada e conta a frequência de cada palavra na frase. Utilize um LinkedHashMap para armazenar as palavras e suas contagens, mantendo a ordem de inserção das palavras.

Obrigado