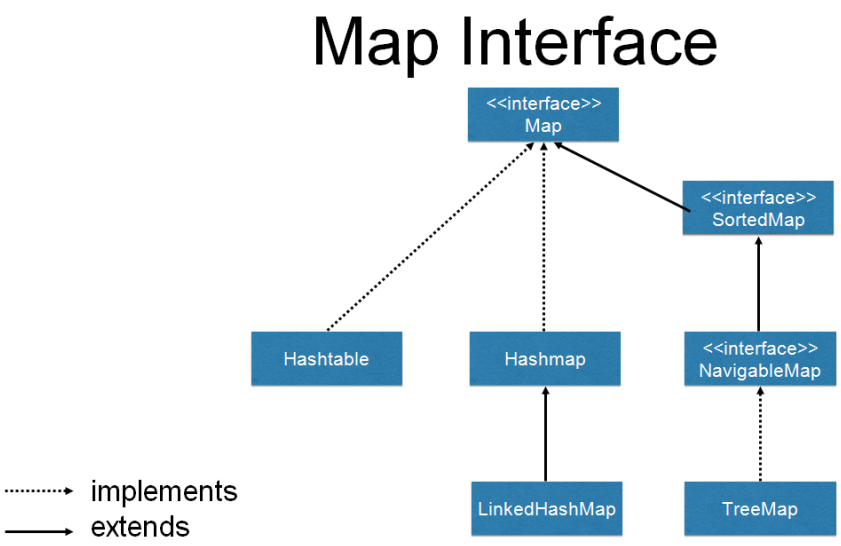
Dia 25 - Map



Conteúdo

Interface Map<K,V>

Parâmetros de tipo:

K - o tipo de chaves mantidas por este mapa

V - o tipo de valores mapeados

Todas as subinterfaces conhecidas:

Vinculações, SimultaneamrentMap<K,V>, ConcurrentNavigableMap<K,V>, LogicalMessageContext, MessageContext, NavigableMap<K,V>, SOAPMessageContext, SortedMap<K,V>

Todas as classes de implementação conhecidas:

AbstractMap, Atributos, AuthProvider, ConcurrentHashMap, ConcurrentSkipListMap, EnumMap, HashMap, Hashtable, IdentityHashMap, LinkedHashMap, PrinterStateReasons, Properties, Provider, RenderingHints, SimpleBindings, TabularDataSupport, TreeMap, UIDefaults, WeakHashMapap

`public interface` **`Map<K,V>`**
Um objeto que mapeia as chaves dos valores. Um mapa não pode conter chaves duplicadas; cada chave pode mapear para no máximo um valor.

Esta interface toma o lugar da classe `Dicionário,` que era uma classe totalmente abstrata em vez de uma interface.

Dia 25 - Map

A interface `Mapa` fornece três *visualizações*de coleção , que permitem que o conteúdo de um mapa seja visto como um conjunto de chaves, coleta de valores ou conjunto de mapeamentos de valor-chave. A *ordem* de um mapa é definida como a ordem em que os iteradores da coleção do mapa retornam seus elementos. Algumas implementações de mapas, como a classe `TreeMap`, fazem garantias específicas quanto à sua ordem; outros, como a classe `HashMap`, não.

Nota: deve ser exercido um grande cuidado se objetos mutáveis forem usados como teclas de mapa. O comportamento de um mapa não é especificado se o valor de um objeto é alterado de uma maneira que afeta `comparações iguais` enquanto o objeto é uma chave no mapa. Um caso especial desta proibição é que não é permitido que um mapa se contenha como uma chave. Embora seja permitido que um mapa se contenha como um valor, é aconselhável extrema cautela: os `métodos iguais` e `hashCode` não são mais bem definidos em tal mapa.

Todas as classes de implementação de mapas de uso geral devem fornecer dois construtores "padrão": um construtor vazio (sem argumentos) que cria um mapa vazio, e um construtor com um único argumento do tipo `Mapa`, que cria um novo mapa com os mesmos mapeamentos de valor-chave que seu argumento. Com efeito, este último construtor permite que o usuário copie qualquer mapa, produzindo um mapa equivalente da classe desejada. Não há como fazer cumprir essa recomendação (pois as interfaces não podem conter construtores), mas todas as implementações de mapas de uso geral no JDK cumprem.

Os métodos "destrutivos" contidos nesta interface, ou seja, os métodos que modificam o mapa em que operam, são especificados para lançar `o UnupportedOperationException` se este mapa não suportar a operação. Se este for o caso, esses métodos podem, mas não são necessários, lançar uma `Exploração Sem SuporteException` se a invocação não tiver efeito no mapa. Por exemplo, invocar o método `putAll (Map)` em um mapa inmodificável pode, mas não é necessário, lançar a exceção se o mapa cujos mapeamentos devem ser "sobrepostos" estiver vazio.

Algumas implementações de mapas têm restrições nas chaves e valores que podem conter. Por exemplo, algumas implementações proíbem chaves e valores nulos, e algumas têm restrições nos tipos de suas chaves. Tentar inserir uma chave ou valor inelegível lança uma exceção não verificada,
tipicamente `NullPointerException` ou `ClassCastException`. Tentar consultar a presença de uma chave ou valor inelegível pode lançar uma exceção, ou pode simplesmente retornar falso; algumas implementações irão exibir o comportamento anterior e algumas irão expor esta última. De forma mais geral, tentar uma operação em uma chave ou valor inelegível cuja conclusão não resultaria na inserção de um elemento inelegível no mapa pode lançar uma exceção ou pode ter sucesso, na opção da implementação. Tais exceções são marcadas como "opcionais" na especificação para esta interface.

Muitos métodos nas interfaces de Framework de Coleções são definidos em termos do método `de iguais.` Por exemplo, a especificação para o método `contémKey (tecla Objeto)` diz: "retorna `verdadeiro` se e somente se este mapa contiver um mapeamento para uma chave `k` tal que `(key==null ? k===null : chave.equals(k))`." Esta especificação *não* deve ser interpretada para implicar que invocar `Map.containsKey` com uma `chave` de argumento não-nulo fará com que `key.equals(k)` seja invocado para qualquer chave `k`. As implementações são livres para implementar otimizações pelas quais a invocação `igual` é evitada, por exemplo, comparando primeiro os códigos de hash das duas teclas. (A especificação `Object.hashCode()` garante que dois objetos com códigos hash desiguais não podem ser iguais.) De forma mais geral, as implementações das várias interfaces do Framework de Coleções são livres para aproveitar o comportamento especificado dos métodos de `objeto` subjacentes onde o implementador julgar apropriado.

Dia 25 - Map

Algumas operações de mapa que realizam travessias recursivas do mapa podem falhar com exceção de instâncias auto-referenciais onde o mapa se contém direta ou indiretamente. Isso inclui o , e métodos. As implementações podem lidar opcionalmente com o cenário auto-referencial, porém a maioria das implementações atuais não o fazem. `clone()` `equals()` `hashCode()` `toString()`

Vídeo

https://www.youtube.com/watch?v=0D2G0xcz8_U&ab_channel=DevDojo

Exercício

1) Dado:

12. public class AccountManager {

13. private Map accountTotals = new HashMap ();

14. fundo privado de aposentadoria;

15

16. public int getBalance (String accountName) {

17. Número inteiro total = (Número inteiro) accountTotals.get (accountName);

18. if (total == null)

19. total = Integer.valueOf (0);

20. retornar total.intValue ();

21.}

23. public void setBalance (String accountName, int amount) {

24. accountTotals.put (accountName, Integer.valueOf (amount));

25.}

26.}

Esta classe deve ser atualizada para fazer uso de tipos genéricos apropriados, sem mudanças no comportamento

(para melhor ou pior). Qual dessas etapas pode ser executada? (Escolha três.)

A. Substitua a linha 13 por

private Map <String, int> accountTotals = new HashMap <String, int> ();

B. Substitua a linha 13 por

private Map <String, Integer> accountTotals = new HashMap <String, Integer> ();

C. Substitua a linha 13 por

Mapa privado <String <Integer>> accountTotals = new HashMap <String <Integer>> ();

D. Substitua as linhas 17-20 por

Dia 25 - Map

```
int total = accountTotals.get (accountName);
if (total == null)
total = 0;
Retorno total;
```

2) Dado:

```
3. import java.util. *;
4. classe pública Magellan {
5. public static void main (String [] args) {
6. TreeMap <String, String> myMap = novo TreeMap <String, String> ();
7. myMap.put ("a", "apple"); myMap.put ("d", "data");
8. myMap.put ("f", "fig"); myMap.put ("p", "pêra");
9. System.out.println ("1º depois da manga:" + // sop 1
10. myMap.higherKey ("f"));
11. System.out.println ("1º depois da manga:" + // sop 2
12. myMap.ceilingKey ("f"));
13. System.out.println ("1º depois da manga:" + // sop 3
14. myMap.floorKey ("f"));
15. SortedMap <String, String> sub = new TreeMap <String, String> ();
16. sub = myMap.tailMap ("f");
17. System.out.println ("1º depois da manga:" + // sop 4
18. sub.firstKey ());
19.}
20.}
```

Qual das instruções System.out.println produzirá a saída primeiro após manga: p?

(Escolha todas as opções aplicáveis.)

A. sop 1

B. sop 2

C. sop 3

D. sop 4

Dia 25 - Map

3) Dado:

3. import java.util. *;

4. public class GeoCache {

5. public static void main (String [] args) {

6. String [] s = {"map", "pen", "marble", "key"};

7. Otelo o = novo Otelo ();

8. Arrays.sort (s, o);

Respostas do autoteste 659

660 Capítulo 7: Genéricos e coleções

9. para (String s2: s) System.out.print (s2 + "");

10. System.out.println (Arrays.binarySearch (s, "map"));

11.}

12. classe estática Othello implementa Comparator <String> {

13. public int compare (String a, String b) {return b.compareTo (a); }

14.}

15.}

Quais são verdadeiras? (Escolha todas as opções aplicáveis.)

A. A compilação falha

B. A saída conterá 1

C. A saída conterá um 2

D. A saída conterá um -1

PROVA

---

2) Dado:

3. import java.util. *;

4. classe pública Magellan {

5. public static void main (String [] args) {

6. TreeMap <String, String> myMap = novo TreeMap <String, String> ();

7. myMap.put ("a", "apple"); myMap.put ("d", "data");

8. myMap.put ("f", "fig"); myMap.put ("p", "pêra");

9. System.out.println ("1º depois da manga:" + // sop 1

10. myMap.higherKey ("f"));

11. System.out.println ("1º depois da manga:" + // sop 2

12. myMap.ceilingKey ("f"));

13. System.out.println ("1º depois da manga:" + // sop 3

14. myMap.floorKey ("f"));

15. SortedMap <String, String> sub = new TreeMap <String, String> ();

16. sub = myMap.tailMap ("f");

17. System.out.println ("1º depois da manga:" + // sop 4

18. sub.firstKey ());

19.}

20.}

Qual das instruções System.out.println produzirá a saída primeiro após manga: p?

(Escolha todas as opções aplicáveis.)

A. sop 1

B. sop 2

C. sop 3

D. sop 4

Dado:

3. import java.util. *;

4. public class GeoCache {

5. public static void main (String [] args) {

6. String [] s = {"map", "pen", "marble", "key"};

7. Otelo o = novo Otelo ();

8. Arrays.sort (s, o);

Respostas do autoteste 659

660 Capítulo 7: Genéricos e coleções

9. para (String s2: s) System.out.print (s2 + "");

10. System.out.println (Arrays.binarySearch (s, "map"));

11.}

12. classe estática Othello implementa Comparator <String> {

13. public int compare (String a, String b) {return b.compareTo (a); }

Dia 25 - Map

14.}

15.}

Quais são verdadeiras? (Escolha todas as opções aplicáveis.)

A. A compilação falha

B. A saída conterá 1

C. A saída conterá "chave do mapa de caneta"

D. A saída conterá um -1