COLEÇÕES — MAP

ACH 2003 — COMPUTAÇÃO ORIENTADA A OBJETOS

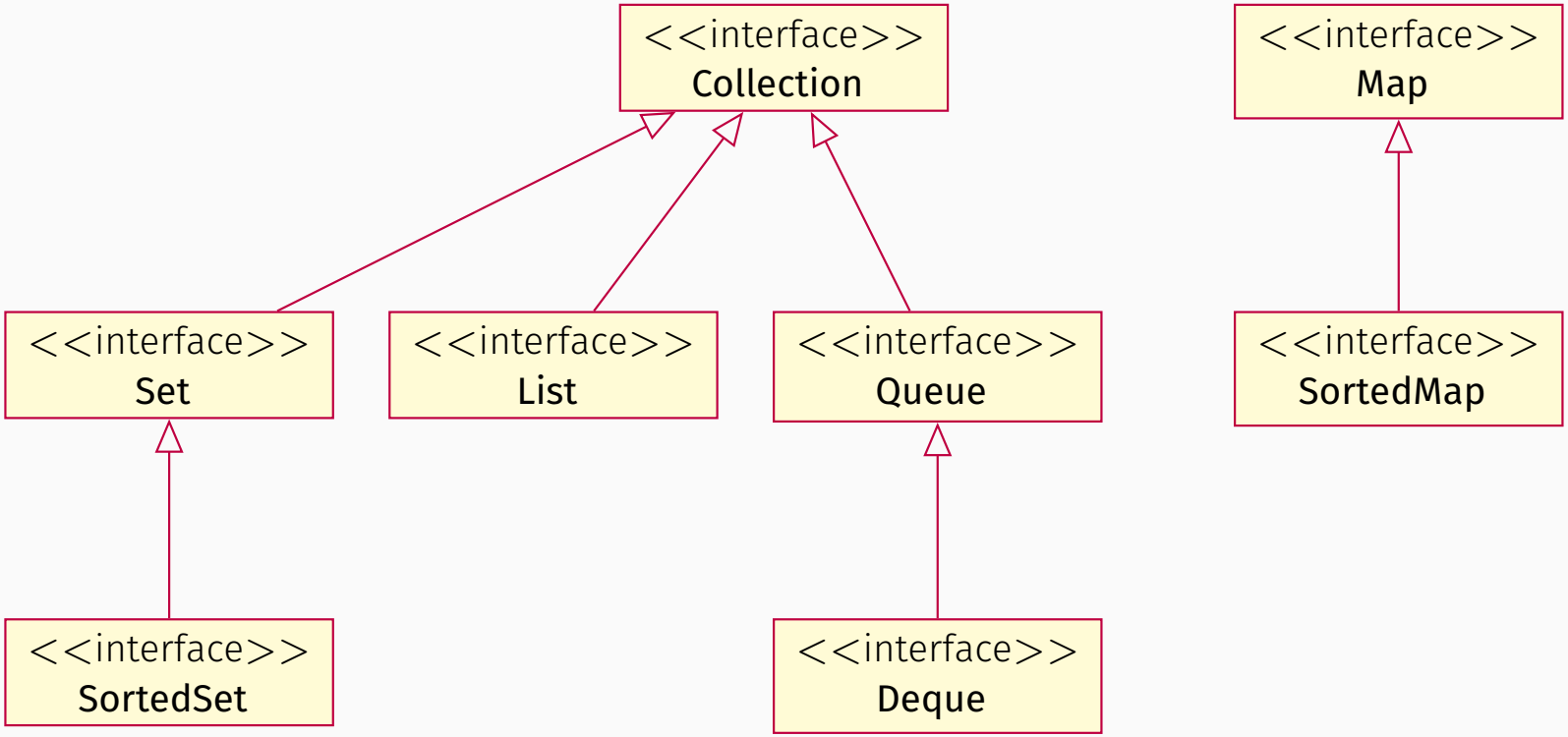
Daniel Cordeiro

18 de março de 2016

Escola de Artes, Ciências e Humanidades | EACH | USP

AVISO

- Não haverá aula nos dias 23 e 25 de março *semana santa* e dias 30 de março e 1º de abril



Principais interfaces de coleções

- `Map` não é exatamente uma `Collection`
- Todas são genéricas: `public interface Collection<E>`

A INTERFACE MAP

MAP

- Um objeto `Map` associa uma chave a um valor
- Um `Map` não pode possuir chaves duplicadas (ou seja, uma chave está associada a um único valor)
- É o equivalente ao conceito matemático de função

Métodos principais

básicos: `put, get, remove, containsKey`

em massa: `putAll, clear`

visões como coleção: `keySet, entrySet, values`

IMPLEMENTAÇÕES DE MAP

O arcabouço provê três implementações de propósito geral para a interface `Map<K,V>`:

- HashMap
- TreeMap
- LinkedHashMap

OPERAÇÕES BÁSICAS

put(K key, V value) associa o valor especificado (*value*) à chave especificada (*key*)

get(Object key) devolve o valor associado à chave `key` ou `null` se não houver nada associado a essa chave

containsKey(Object key) devolve `true` se a instância contiver algum mapeamento para essa chave

containsValue(Object value) devolve `true` se o mapa possuir uma ou mais chaves que apontam para o valor especificado

size() devolve o número de mapeamentos chave–valor dessa instância

isEmpty() devolve `true` se esse mapa não tiver nenhum par chave–valor

EXEMPLO

O que esse método faz?

```
import java.util.*;

public class Freq {
  public static void main(String[] args) {
      Map<String, Integer> m = new HashMap<String, Integer>();

      for (String a : args) {
        Integer freq = m.get(a);
        m.put(a, (freq == null) ? 1 : freq + 1);
      }

      System.out.println(m.size() + " palavras distintas:");
      System.out.println(m);
    }
}
```

O que esse método faz?

```
import java.util.*;

public class Freq {
  public static void main(String[] args) {
      Map<String, Integer> m = new HashMap<String, Integer>();

      for (String a : args) {
        Integer freq = m.get(a);
        m.put(a, (freq == null) ? 1 : freq + 1);
      }

      System.out.println(m.size() + " palavras distintas:");
      System.out.println(m);
    }
}
```

```
$ java Freq if it is to be it is up to me to delegate

8 palavras distintas:
{to=3, delegate=1, be=1, it=2, up=1, if=1, me=1, is=2}
```

EXEMPLO

E se eu quisesse imprimir as palavras em ordem alfabética?

EXEMPLO

E se eu quisesse imprimir as palavras em ordem alfabética?

`HashMap` → `TreeMap`

```
$ java Freq if it is to be it is up to me to delegate

8 palavras distintas:
{be=1, delegate=1, if=1, is=2, it=2, me=1, to=3, up=1}
```

EXEMPLO

E se eu quisesse imprimir as palavras na ordem em que foram inseridas?

EXEMPLO

E se eu quisesse imprimir as palavras na ordem em que foram inseridas?

`HashMap` → `LinkedHashMap`

```
$ java Freq if it is to be it is up to me to delegate

8 palavras distintas:
{if=1, it=2, is=2, to=3, be=1, up=1, me=1, delegate=1}
```

Mais uma amostra da simplicidade e poder de um arcabouço baseado em interfaces.

MAP: OPERAÇÕES EM MASSA

- `clear`: remove todos os mapeamentos
- `putAll`: análogo a `Collection.addAll`

MAP: OPERAÇÕES EM MASSA

- `clear`: remove todos os mapeamentos
- `putAll`: análogo a `Collection.addAll`

Exemplo:

Suponha que você use um `Map` para implementar uma coleção de atributos–valores. O método `putAll`, em conjunto com o construtor de conversão, permite uma implementação interessante para uma coleção de atributos com valores padrão (*default*). Pense num conjunto de atributos de configuração de uma arquivo `.ini`, por exemplo.

```
static <K, V> Map<K, V> newAttributeMap(Map<K, V>defaults, Map<K, V> overrides)
    Map<K, V> result = new HashMap<K, V>(defaults);
    result.putAll(overrides);
    return result;
}
```

VISÕES COMO COLEÇÕES

Os métodos de visão como coleção permitem visualizar um `Map` como uma `Collection` de três modos:

keySet() um `Set` com as chaves contidas no mapa

values() uma `Collection` com os valores contidos no mapa[1]

entrySet um `Set` com os pares chave–valor contidos no mapa; os pares são do tipo `Map.Entry`

Essas visões

fornecem a única forma de iteração em um `Map`.

[1] Por que esse método devolve uma `Collection` e não um `Set` como o primeiro?

for-each

```
for (KeyType key : m.keySet())
    System.out.println(key);
```

usando um iterator

```
// Filtra um map baseado em alguma propriedade de suas chaves
for (Iterator<Type> it = m.keySet().iterator(); it.hasNext(); )
    if (it.next().isBogus())
        it.remove();
```

iteração pelos pares chave–valor

```
for (Map.Entry<KeyType, ValType> e : m.entrySet())
    System.out.println(e.getKey() + ": " + e.getValue());
```

MAIS EXEMPLOS

Usar as operações de `Collection` nas visões permite fazer muita coisa interessante.

```
// verifica se as duas instâncias possuem os mesmos
// pares chave-valor
if (m1.entrySet().containsAll(m2.entrySet())) {
    ...
}

// verifica se os dois possuem mapeamentos
// para as mesmas chaves
if (m1.keySet().equals(m2.keySet())) {
    ...
}
```

MAIS EXEMPLOS

Para validar se os atributos lidos de um arquivo de configuração (`attrMap`) possuem todos os valores obrigatórios (`requiredAttrs`) e só possuem atributos permitidos (`permittedAttrs`), você poderia usar algo como:

```
static <K, V> boolean validate(Map<K, V> attrMap,
                               Set<K> requiredAttrs, Set<K>permittedAttrs) {
    boolean valid = true;
    Set<K> attrs = attrMap.keySet();

    if (! attrs.containsAll(requiredAttrs)) {
        Set<K> missing = new HashSet<K>(requiredAttrs);
        missing.removeAll(attrs);
        System.out.println("Missing attributes: " + missing);
        valid = false;
    }
    if (! permittedAttrs.containsAll(attrs)) {
        Set<K> illegal = new HashSet<K>(attrs);
        illegal.removeAll(permittedAttrs);
        System.out.println("Illegal attributes: " + illegal);
        valid = false;
    }
    return valid; }
```

IMPLEMENTAÇÕES DE COLLECTION

Interface	Hash Table	Resizable Array	Balanced Tree	Linked List	Hash Table+Linked List
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Deque		ArrayDeque		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap

Exercício:

Projetar um programa que receba um conjunto de palavras e imprima quais são anagramas.

Anagrama

Palavra ou frase feita com as letras de outra (ex.: as palavras asco, caos, cosa, saco, soca são anagramas de caso).

BIBLIOGRAFIA

- The Java™ Tutorials – Collections: `https://docs.oracle.com/javase/tutorial/collections/`