Concurrent collections - TP3

November 5, 2018

Thomas COUCHOUD
thomas.couchoud@etu.univ-tours.fr
Victor COLEAU
victor.coleau@etu.univ-tours.fr

Chapter 1

Qu'est-ce que c'est ?

1.1 Collection

En Java Collection désigne une interface. Cette dernière à pour but de désigner un groupe d'objets appelés éléments. Cette définition est très générique ce qui laisse les implémentations variées. Certaines peuvent être ordonnées, d'autres non; certaines avec duplicatas, d'autres non.

Dans le JDK on ne trouve pas d'implémentation directe de cette interface, seulement des implémentations d'interfaces plus spécifiques telles que List ou Set.

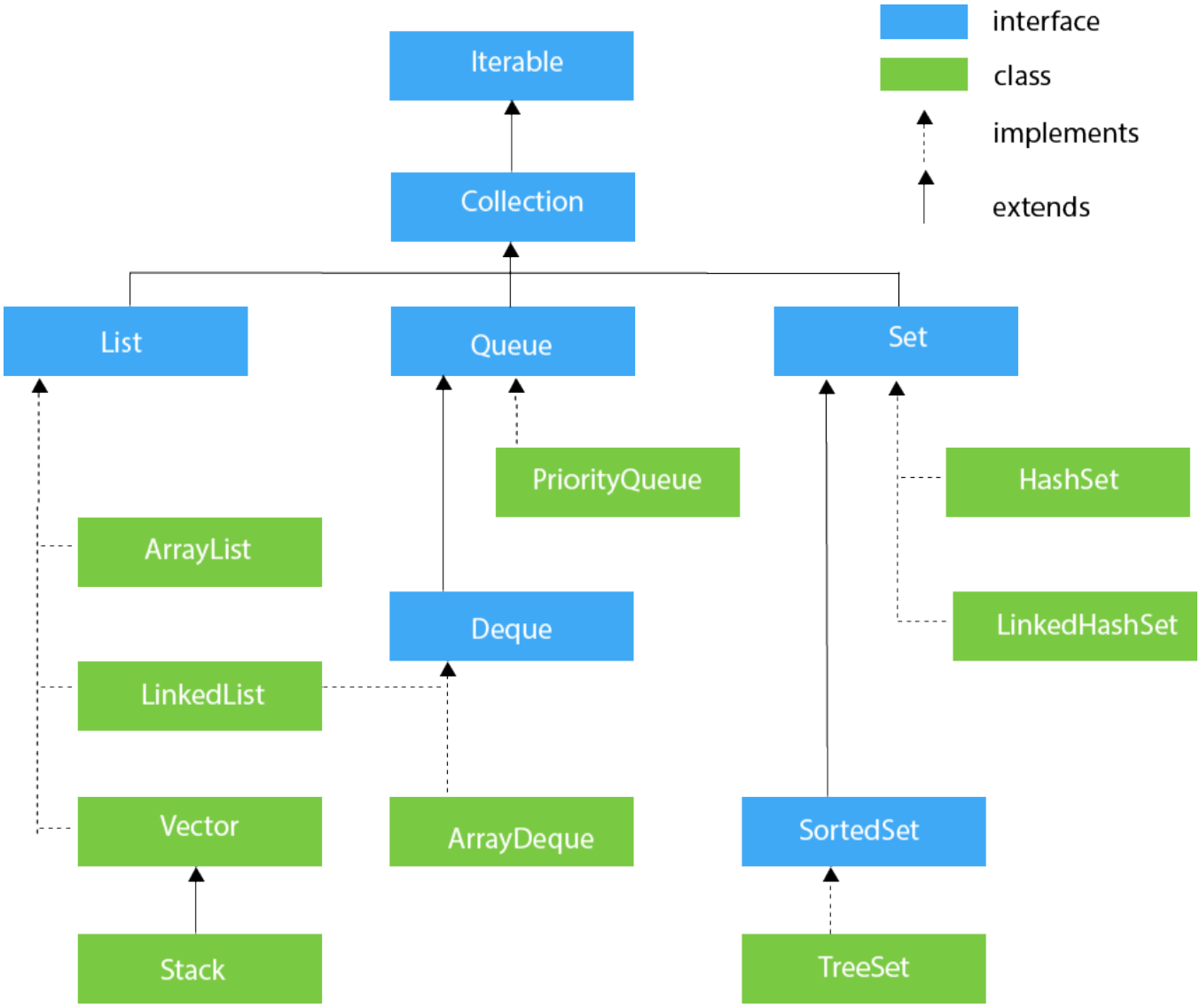

Figure 1.1

Collection définie notamment les méthodes add(element), contains(element) et remove(element). Parlons donc rapidement des 3 grandes extensions de Collection: List, Set, Queue.

- List définie une collection d'objets ordonnée. On peut avoir des éléments dupliqués et il y a du sens de parler de l'élément à l'indice k. De ce fait la méthode remove(index) est aussi définie dans ce cas.

  Exemple: $[1,2,3,4,3,5,4]$

- Queue désigne une queue. Cette dernière n'est pas très différente d'une list: une collection d'éléments ordonnés. Cependant on ne peux manipuler les objets qui se trouvent uniquement aux extrémités (on ajoute d'un coté, et on retire d'un autre). Parler de l'élément à l'indice k n'a donc aucun sens, on le verra quand il est en tête de queue.

  On retrouve les fonctions offer(element), poll et peek.

- Set définie une collection d'objet n'ayant pas d'ordre et pas de duplicata.

1.2 Concurrence

Listing 1.1 – Main.java

```
1  import java.util.*;
2  class Main extends Thread {
3      static ArrayList l = new ArrayList();
4      public void run()
5      {
6          try {
7              Thread.sleep(2000);
8          }
9          catch (InterruptedException e) {
10         }
11         l.add("E4");
12     }
13
14     public static void main(String[] args) throws InterruptedException
15     {
16         l.add("E1");
17         l.add("E2");
18         l.add("E3");
19         new Main().start();
20
21         Iterator i = l.iterator();
22         while (i.hasNext()) {
23             String s = (String)i.next();
24             System.out.println(s);
25             Thread.sleep(6000);
26         }
27         System.out.println(l);
28     }
29 }
```

Si on lance le code ci-dessus, nous allons obtenir une exception java.util.ConcurrentModificationException. En effet, nous avons deux threads qui tentent d'accéder à la même liste en même temps (le while et notre add ligne 11).

Afin de pouvoir réaliser ces opérations soit on gère cela dans notre code pour être sûr qu'une seule opération est réalisée à la fois, ou bien on utilise des implémentations de collections qui gère la concurrence.

Deux façons de procéder reposent sur Collections.synchronizedXxx() et le package java.util.concurrent.

Chapter 2

Comment ça marche ?

2.1 Collections.synchronizedXxx()

2.2 java.util.concurrent

Chapter 3

Quels sont les impacts sur les performances ?