Nachdenkzettel: Collections

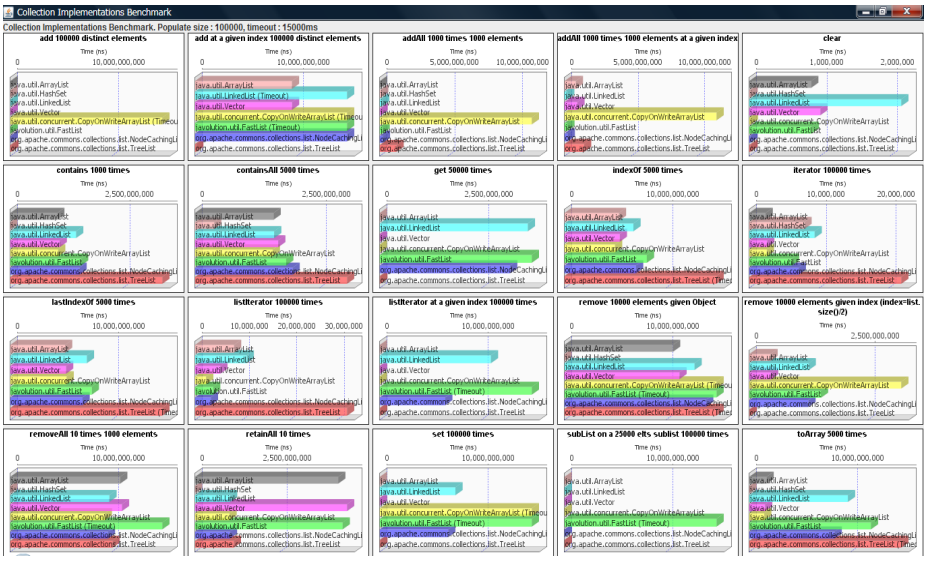
Pia Schilling, Sara Tietze, Merve Özdemir

1. ArrayList oder LinkedList – wann nehmen Sie was?

Linked List	Array List
wenn man eher Zugriffe am Anfang oder Ende der Liste hat	ArrayList wenn Zugriffe schnell über Index erfolgen soll (auch zwischendrin)
schneller wenn man Elemente mittendrin einfügen will bzw. die Liste oft bearbeitet	wenn man schon genau weiß, was in die Liste soll ist eine Array List besser, bzw. wenn man nichts mehr einfügen will
benötigt mehr Speicher	Effizient in der Speichernutzung
langsamer	schneller

2. Interpretieren Sie die Benchmarkdaten von: http://java.dzone.com/articles/java-collection-performance. Fällt etwas auf?

- **CopyOnWriteArrayList** ist vor allem beim Hinzufügen und Löschen von Elementen auffallend langsam
- **ArrayList** und **HashSet** meist sehr schnell (insbesondere add und get)
- **TreeList** ist auch oft langsamer als die anderen Listen
- **FastList** hat oft einen Timeout → wohl doch nicht so schnell



3. Wieso ist CopyOnWriteArrayList scheinbar so langsam?

Weil die Liste jedes mal wenn etwas verändert wird komplett kopiert wird. Änderungen werden also nicht an der Originalliste vorgenommen sondern an einer Kopie davon. Diese Kopie muss erst gemacht werden was Zeit kostet.

4. Wie erzeugen Sie eine thread-safe Collection (die sicher bei Nebenläufigkeit ist) (WAS?? die Arraylists, Linkedlists, Maps etc. sind NICHT sicher bei multithreading??? Wer macht denn so einen Mist???)

Hashtables benutzen

5. Achtung Falle!

List|<Integer> list = new ArrayList<Integer>;

```
Iterator<Integer> itr = list.iterator();
while(itr.hasNext()) {
   int i = itr.next();
   if (i > 5) { // filter all ints bigger than 5
      list.remove();
   }
}
```

Falls es nicht klickt: einfach ausprobieren...
Macht das Verhalten von Java hier Sinn?
Gibt es etwas ähnliches bei Datenbanken? (Stichwort: Cursor. Ist der ähnlich zu Iterator?)

Das Verhalten von Java macht Sinn, weil bei list.remove() nichts in der Klammer steht. Somit weiß es nicht was removed/gelöscht werden soll. Das “list” sollte durch ”itr“ ersetzt werden, dann wird das Element gelöscht, auf welches der Iterator zu diesem Zeitpunkt zeigt.
Der Cursor in Datenbanken zeigt, wie der Iterator die aktuelle Position.

```java
public class Main {
    public static void main(String[] args) {

        List<Integer> list = new ArrayList<>();
        list.add(1);
        list.add(2);
        list.add(6);

        System.out.println(list);

        Iterator<Integer> itr = list.iterator();
        while(itr.hasNext()) {
            int i = itr.next();
            if (i > 5) { // filter all ints bigger than 5
                itr.remove();
            }
        }
        System.out.println(list);
    }
```

```
/Library/Jav
[1, 2, 6]
[1, 2]
```

6. Nochmal Achtung Falle: What is the difference between get() and remove() with respect to Garbage Collection?

get() liefert das Element an der Stelle die eingegeben wird zurück, während remove() das Element an der Stelle entfernt.

7. Ihr neuer Laptop hat jetzt 8 cores! Ihr Code für die Verarbeitung der Elemente einer Collection sieht so aus:

War der Laptop eine gute Investition?

```java
Iterator<Integer> itr = list.iterator();
while(itr.hasNext()) {
  int i = itr.next();
  //do something with i….
}
```

Es wird immer noch nur ein Core benutzt. Damit der Laptop eine gute Investition war, müsste man multi-threaden.

Für die Mutigen: mal nach map/reduce googeln!