Nachdenkzettel: Collections

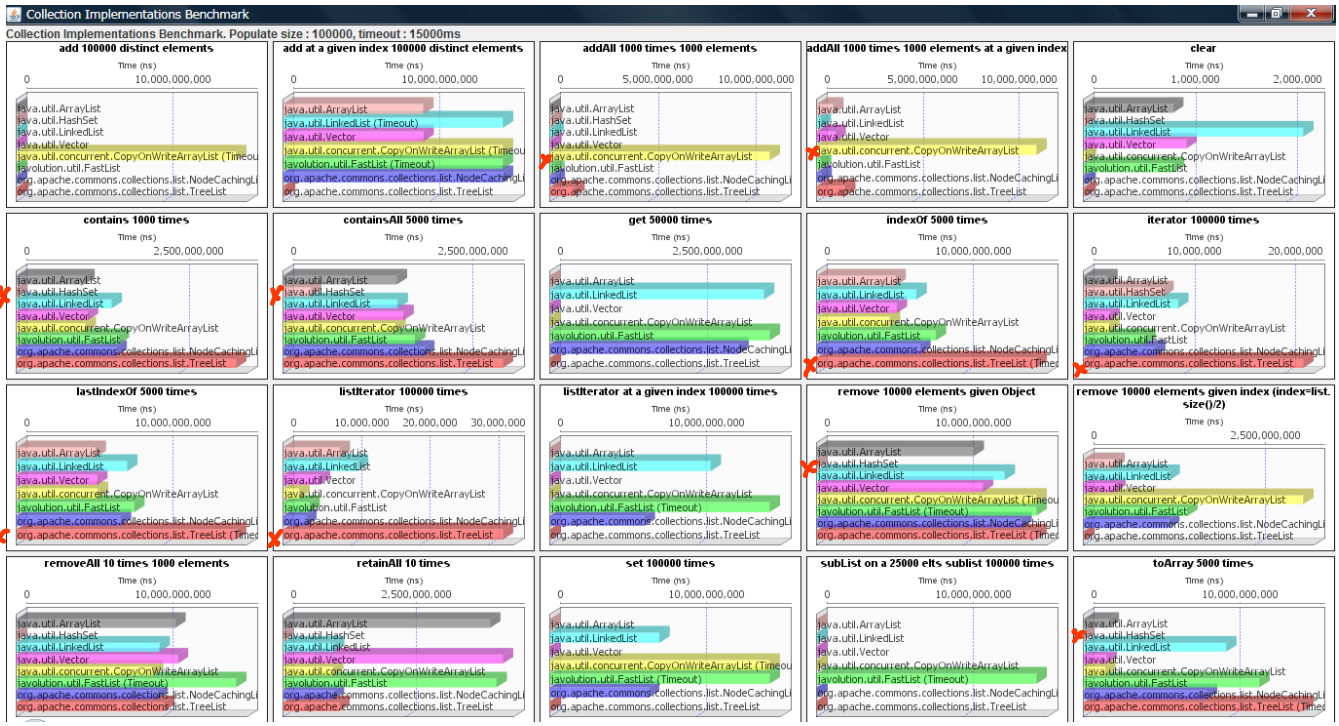
1. ArrayList oder LinkedList – wann nehmen Sie was?

↳ More efficient in cases in which the application demands storing data and accessing it.

Linked List:

More efficient when the application demands manipulation of the stored data.

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



- Adding many elements to CopyOnWriteArrayLists takes significantly longer than the other List types
- contains is fastest in hash sets, same is true for remove
- iterator and indexOf... takes very long in TreeLists
- hashlists are the fastest type to be converted to an array

3. Wieso ist CopyOnWriteArrayList scheinbar so langsam?

It makes a copy of the underlying array whenever it is modified.

↝ meant for synchronized data manipulation

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???)

use Collections.SynchronizedCollection() or CopyOnWriteArrayList

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?)

By removing a list item, the cursor is shifted which could lead to unexpected behavior.

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

get() returns the value at the given index. remove() does the same, but additionally removes the value from the list after returning it and the garbage collector will remove it.

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

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

War der Laptop eine gute Investition? ~ no, as the code above only uses one of the cores to run, so the other cores are unnecessary for that cause.

Für die Mutigen: mal nach map/reduce googeln!