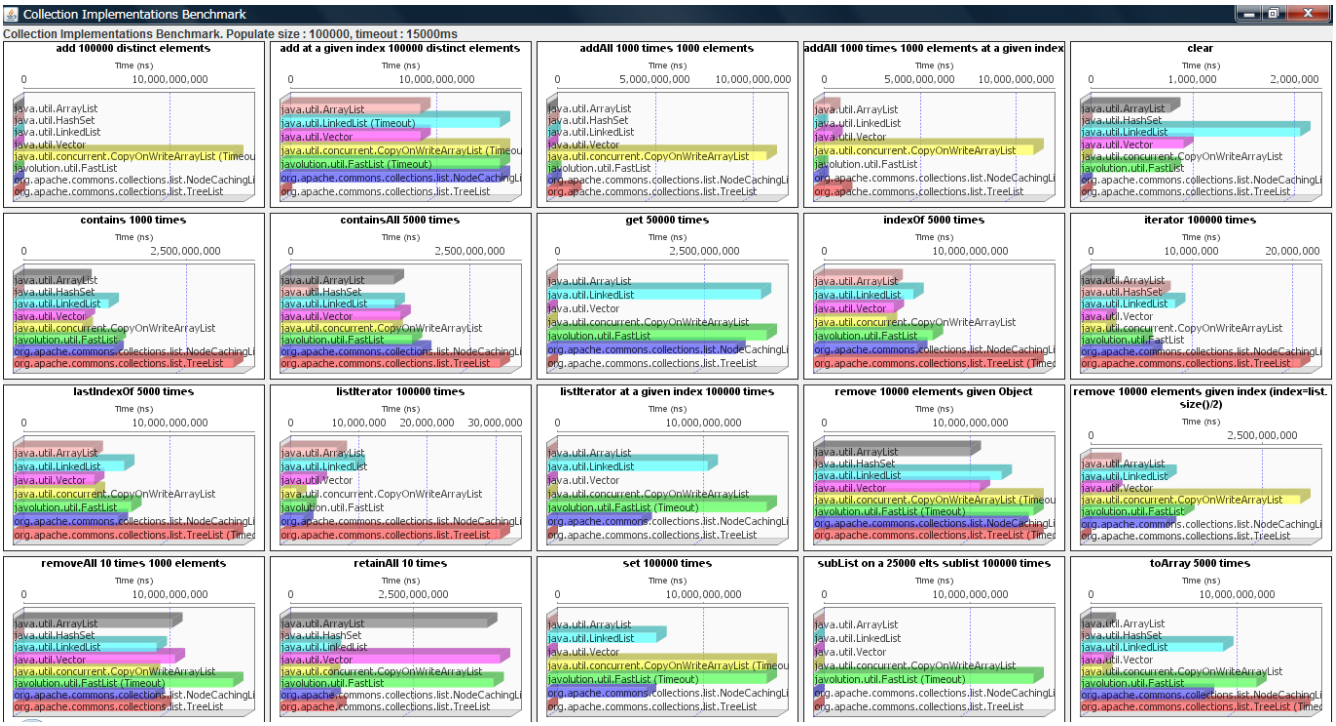
Nachdenkzettel: Collections

1. ArrayList oder LinkedList – wann nehmen Sie was?

Array List: wenn man nach dem Index fragen will, also einfach etwas in der LIste finden muss. Bei vielen gets
Linked List: wenn man viel einfügen muss und oft Einträge an einer bestimmten Stelle dazwischen eingeschoben werden sollen.

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



Array Lists haben eine vergleichsweise gute Performance beim Abrufen.
Bei retain hingegen hat die Linked List eine deutlich bessere Performance
Die Linked List braucht in den meisten Fällen ein kleines bisschen mehr Zeit.
Für viele Punkte gibt es eine optimalere Lösung, aber LinkedList und ArrayList scheinen Allrounder zu sein, da sie in nichts zu negativ auffallen.

3. Wieso ist CopyOnWriteArrayList scheinbar so langsam?

Wenn sie erweitert werden, wird das alte Array kopiert und ein neues Element eingefügt. Wenn sich 2 Threads eine Collection teilen, wird eine komplette Kopie erstellt. "Kopiere erst wenn ich muss und share ansonsten"

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???)

CopyOnWriteArrayList ist eine thread-safe Collection
Unter Collections gibts eine Staticmethode in die man die Collection hinzufügen kann. (Concurrent)

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(i);
   }
}
```

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?)

Man bearbeitet die Collection in der Schleife -> Cocurrent modification Exception

Man ändert die Liste während sie interferiert wird. es ändert sich die Länge die in der Schleife wichtig ist

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

get() macht nur eine Referenz -> wenn aber eine referenz noch außerhalb der Liste liegt würde der GarbageCollector auch nicht zugreifen
remove() löscht

Bei mir hats in der VL gelaggt sorry, konnte nicht alles mitschreiben

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?

Für die Mutigen: mal nach map/reduce googeln!

Wie hängen die Cores mit dem Code zusammen?: Dieser Code nutzt nur einen Thread daher nutzt er die Cores nicht. Daher ist der Laptop für DIESEN Code keine Investition. Man müsste Parallelisieren ->Partitionierung ->8 Collections, indem man map(den Code als Parameter und 1/8 der Collection)