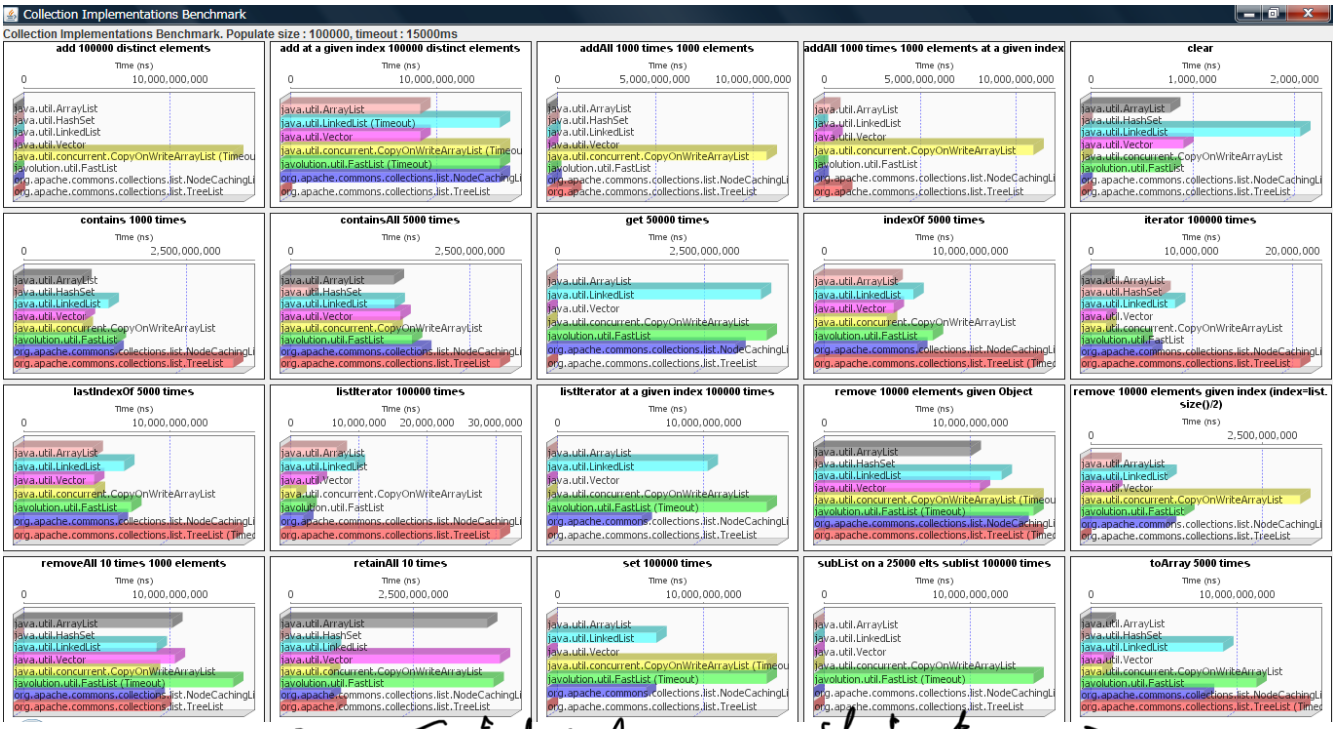
Nachdenkzettel: Collections

1. ArrayList oder LinkedList – wann nehmen Sie was?

→ ArrayList: bei vielen get-methoden / Instanzfragen

→ LinkedList: Wenn man noch mittendrin etwas ändern möchte oder dort etwas einfügen will

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



→ CopyOnWriteArrayList ist sehr oft Timed out

→ Die Zeit wird gemessen in den Collections, also wie lange sie brauchen bei bestimmten Anwendungen

3. Wieso ist CopyOnWriteArrayList scheinbar so langsam?

→ Weil die CopyOnWriteArrayList macht jedes Mal eine Kopie von einem schreibvorgang

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???)

→ z.B. arraylist lässt nur ein Thread zugreifen, während alle anderen warten müssen bevor sie darauf zugreifen können, somit gehen Daten nicht verloren

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 Hauptproblem ist, dass man mit list.remove die Liste direkt ändert, man muss es zu itr.remove ändern

→ Wir würden ansprechen dürfen mit Iterator vorbeigehen! merkwürdig

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

→ Nur wenn wir remove verwenden, wird die Referenz entfernt ansonsten bleibt es dort

→ get() verkommt Kopie der Referenz, die Referenz bleibt drin, erst wenn wir remove machen verschwindet die Kopie des Objektes

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!

- Wir iterieren über eine Collection, er iteriert immer weiter
- Single Thread, es bringt aber nicht viel, egal wie viele Cores man hat, da er nur eins verwendet
- Man kann den Code ändern, um alle 8 Cores zu verwenden, indem man die Liste aufteilt und auf die Cores aufteilt, am Ende wird das Ergebnis wieder zusammengeführt