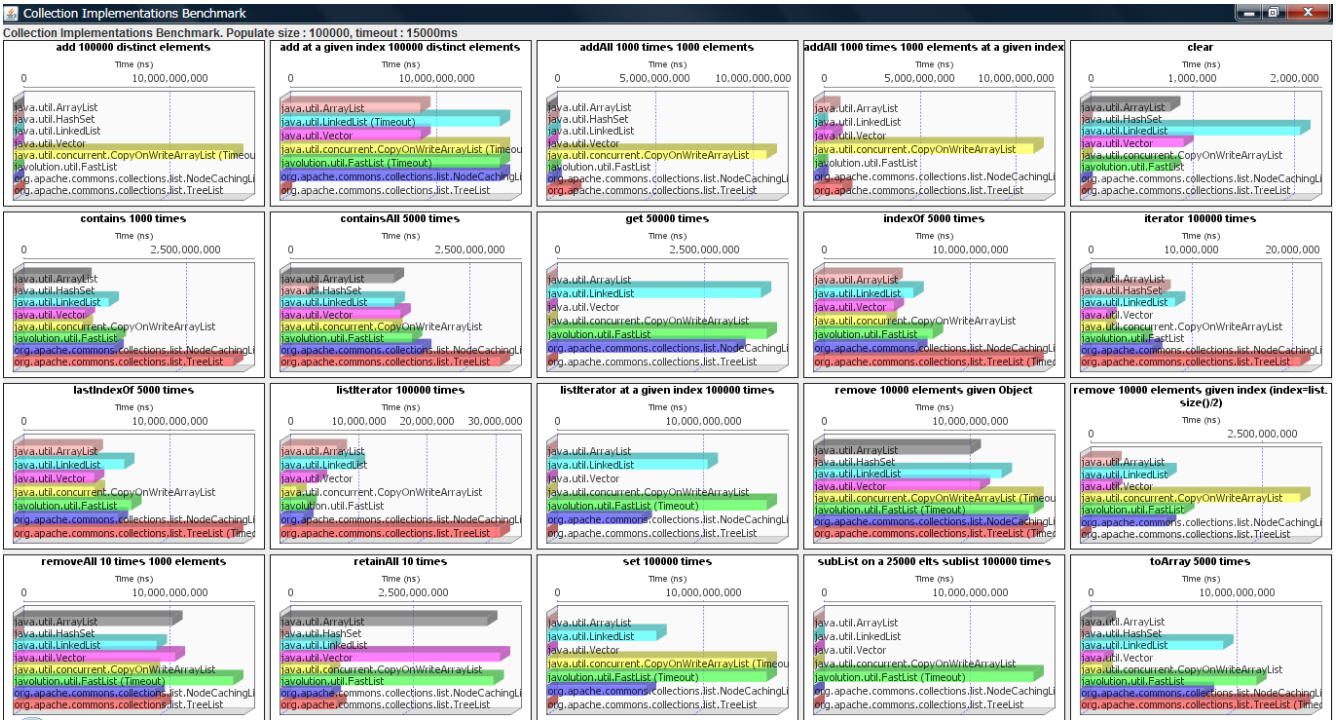
Nachdenkzettel: Collections

1. ArrayList oder LinkedList – wann nehmen Sie was?

LinkedList: besser beim schreiben weil nur Referenz zur vorherigen und nachfolgenden Element geändert

ArrayList: besser beim lesen weil Index direkt zugreifbar

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



Beim schreiben ist CopyOnWriteArrayList sehr langsam dafür verhält es sich beim lesen ähnlich wie die ArrayList. Somit immer den Anwendungsfall beachten.

3. Wieso ist CopyOnWriteArrayList scheinbar so langsam?

Weil immer eine Kopie vom Inhalt erstellt und dann gemerged wird.

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???)

Die CopyOnWriteArrayList ist im diesem Fall besser, da man nicht den aktuellen Inhalt sondern eine Kopie bearbeitet.

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?)

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

Get liefert eine Kopie der aktuellen Referenz.
Remove löscht eine Referenz was den Garbagecollector den Weg ebenet.

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!

Nicht sinnvoll weil singlethreaded code.