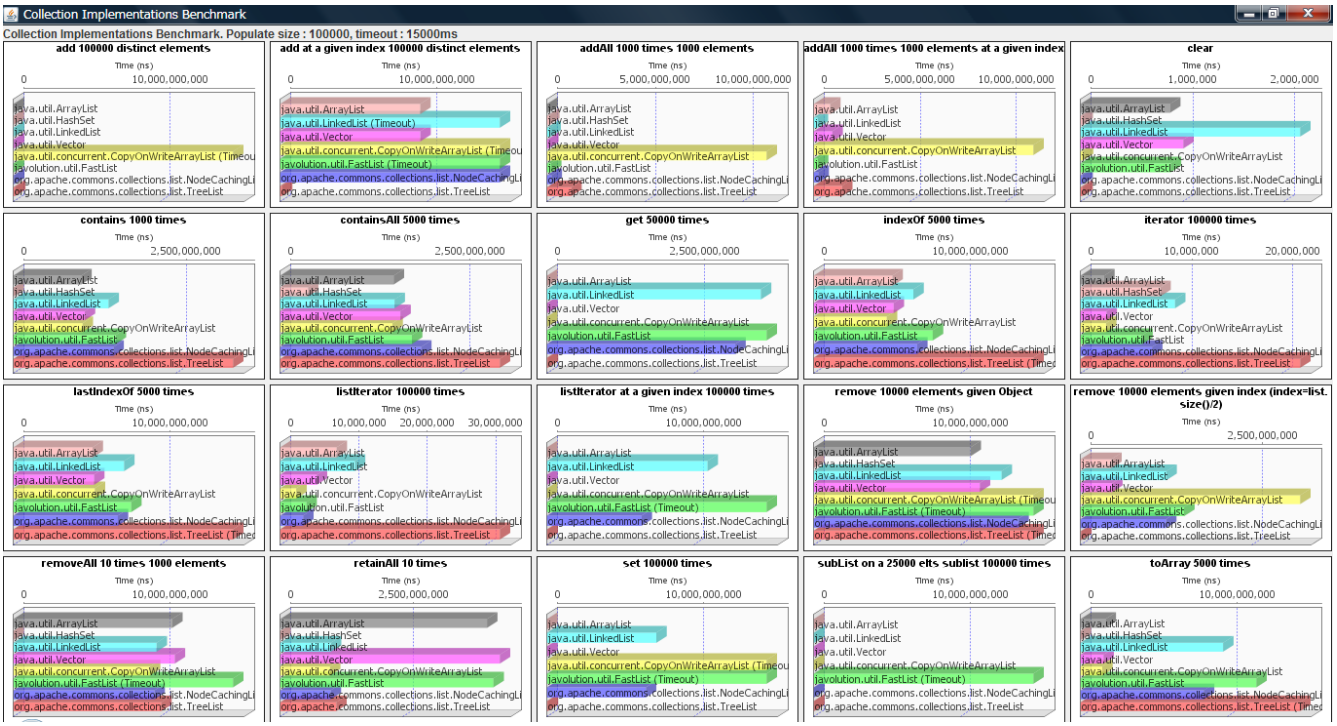
Nachdenkzettel: Collections

1. ArrayList oder LinkedList – wann nehmen Sie was?

Arraylist: schneller Zugriff auf Elemente

LinkedList: schnelle Bearbeitung

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



Je nach Collection brauchen die Methoden unterschiedlich lange für die selbe Aktion

3. Wieso ist CopyOnWriteArrayList scheinbar so langsam?

Die Arraylist wird bei jeder Bearbeitung mit einer Kopie von sich selbst mit der Neuerung ersetzt

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???)

Vector, Hashtable, Properties, Stack

5. Achtung Falle!

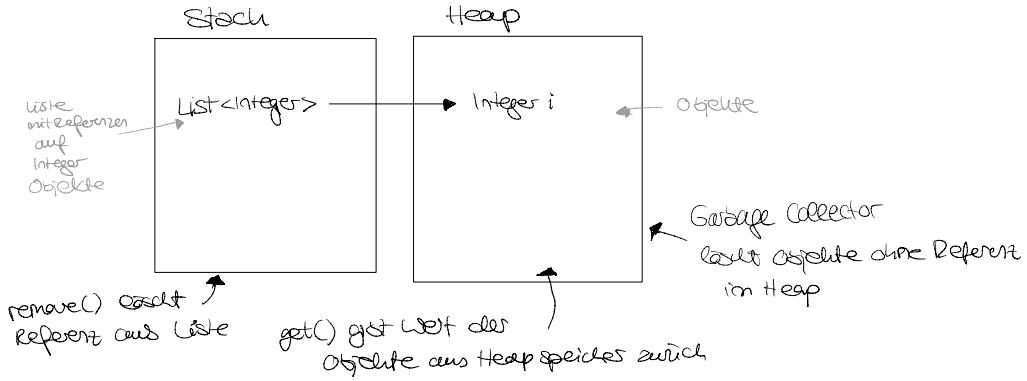
List|<Integer> list = new ArrayList<Integer>;

```
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?)

macht kein Sinn, denn i nimmt den Wert vom Index an & die Elemente nach dem 5. werden gelöscht

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



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? Nein, weil nur 1 core belastet wird

Für die Mutigen: mal nach map/reduce googeln!