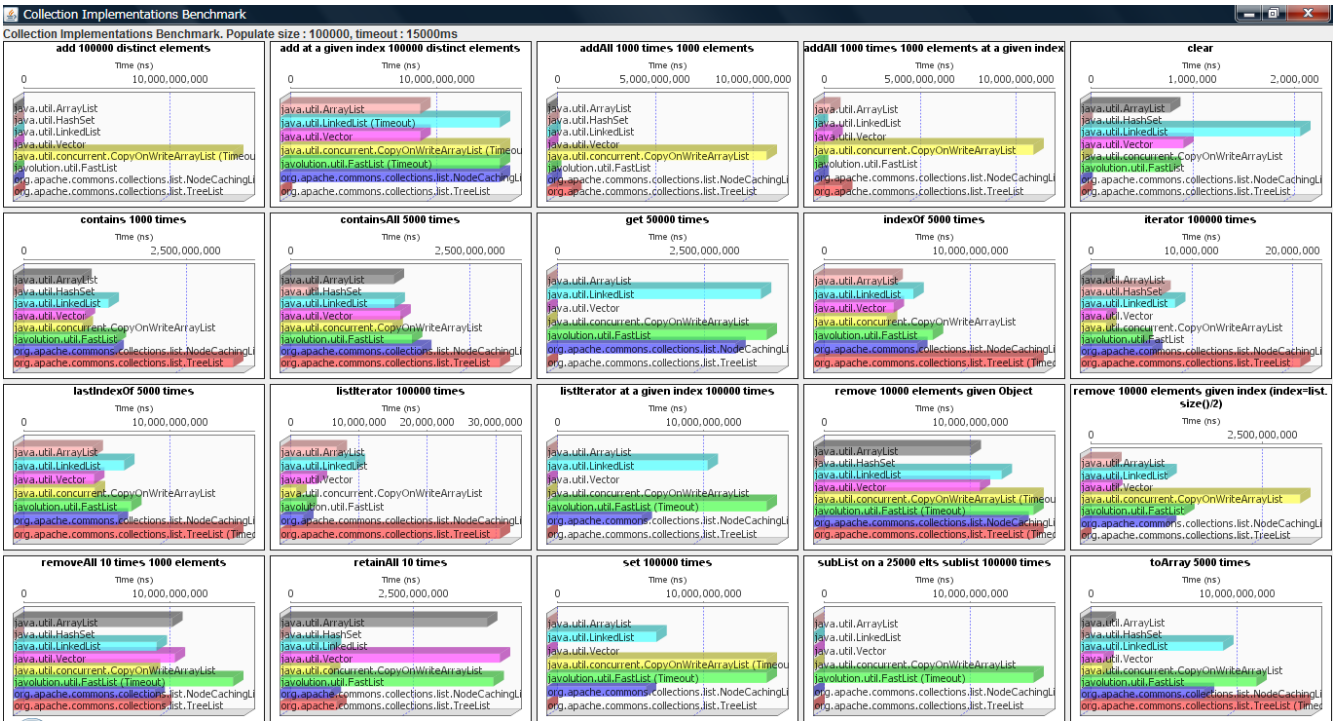
Nachdenkzettel: Collections

1. ArrayList oder LinkedList – wann nehmen Sie was?

LinkedList eignet sich für sequenzielle, aufeinanderfolgende Zugriffe, wohingegen sich ArrayList für zufällige Zugriffe eignet.
Für die LinkedList sind Zugriffe und Änderungen nach Big-Oh Notation O(1), nur der Zugriff auf ein random Index ist O(n). Für die ArrayList sind Zugriffe sowie ein Einfügen am Ende generell O(1), nur anderes Einfügen O(n).

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



CopyOnWriteArrayList ist sehr langsam bei Einfügen von Objekten sehr langsam.

3. Wieso ist CopyOnWriteArrayList scheinbar so langsam?

Um Tread-Safety zu garantieren erstellt das Objekt bei jeder Modifikation eine Kopie des Arrays. Grund ist, dass z.B. in einem anderen Tread ein Iterator auf den ArrayList zugreift, und hierfür das Objekt nicht verändert werden kann. Ein Iterator kann hier jedoch keine Modifikationen durchführen, sondern nur lesen.

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???)

Ich verbiete den Zugriff, sprich werfe eine Exception, auf meine Objekte, sollte schon eine Modifikation am laufen sein.

5. Achtung Falle!

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

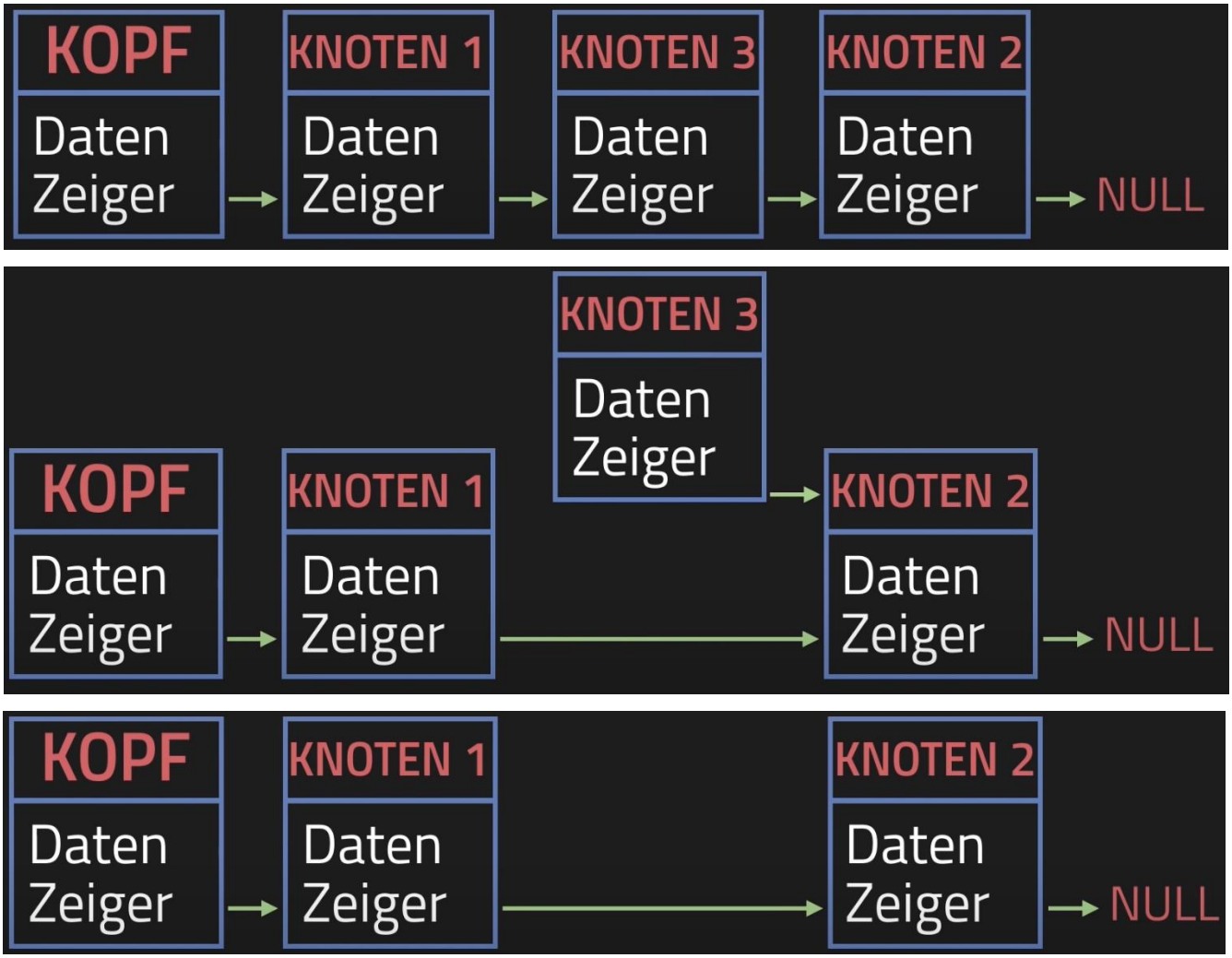
```
Iterator<Integer> itr = list.iterator();
while(itr.hasNext()) {
  int i = itr.next();
  if (i > 5) { // filter all ints bigger than 5
    list.remove();
  }
}
```

Da der Iterater momentan Zugriff auf die ArrayList hat, kann man mit list.remove(n) keine Modifikation an der ArrayList vornehmen und es wird eine ConcurrentModificationException ausgegeben.

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?

Mit get(n) wird ein Element an der Stelle n abgefragt. Dieses ist nach dem Aufruf weiterhin vorhanden. Mit remove(n) wird die Referenz des Elements aufgelöst, weshalb es vom Garbage Collector aufgesammelt wird. Der Prozess läuft wiefolg ab:



Dabei rutschen alle Elemente hinter dem entfernten Element auf und ihr Index wird um 1 kleiner.

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! Die 8 Cores sind ja dafür da, dass sie parallel arbeiten können. Allerdings kann nur ein Iterator gleichzeitig auf die ArrayList zugreifen, weshalb so viele Cores für diesen Fall unnütz sind.
Würde ein anderer Core zur gleichen Zeit auf die ArrayList zugreifen, käme es zu einer ConcurrentModificationException.

Für die Mutigen: mal nach map/reduce googeln!