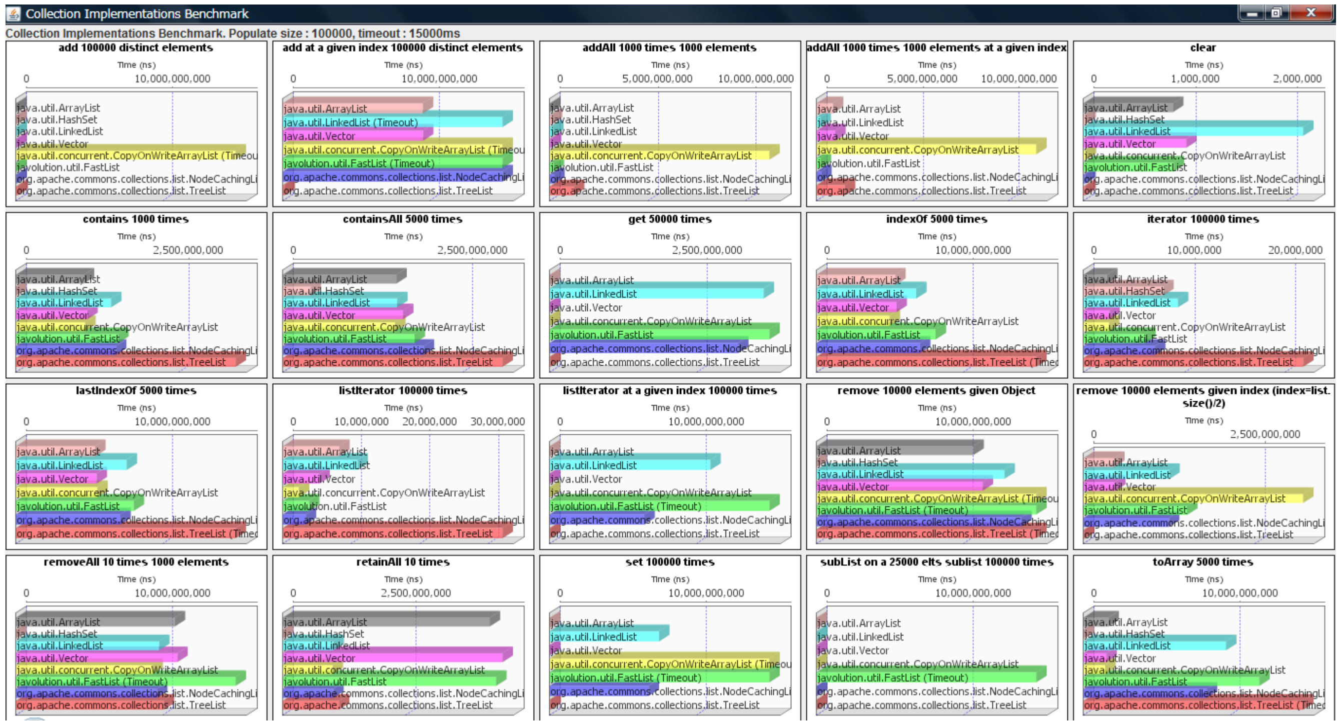
Nachdenkzettel: Collections

1. ArrayList oder LinkedList – wann nehmen Sie was?

You’d use an ArrayList if you want to store and access data at a fast pace. However, if you’d want to manipulate already saved data as fast as possible, LinkedList is the right tool for this. Also, if you need a List, that’s also a queue you want to use a LinkedList as it implements both interfaces.

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



- CopyOnWriteArrayList is very slow according to these benchmarks.
- ArrayList isn’t really slow at any usecase, but oftentimes isn’t the fastest Collection.
- HashSet is pretty fast compared to the Lists on contains()

3. Wieso ist CopyOnWriteArrayList scheinbar so langsam?

A CopyOnWriteArrayList makes a new copy of the underlying array every time a modification is done. This is due to its itended use as a easy means to cope with synchronized data manipulation.

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???)

Collections.synchronizedCollection()

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?)

The above code example can’t be compiled. So, we tried to imagine what it should do and corrected it:

```
List<Integer> list = new ArrayList<Integer>();

System.out.println(list);

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

System.out.println(list);
```

And, the answer is yes. It makes sense. When calling itr.next() the iterator increments by one automatically. The side effect is that it returns the corresponding element in the list.

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

Both methods return the value corresponding to the given index. While, get does nothing else, remove also removes the element from the list. Therefore, the Garbage Collector has to remove it from memory if the object reference isn’t saved somewhere else (i.e. when using the return value (without unboxing (e.g. when using the Integer class))).

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?

The laptop has been a good investment, because now it feels way better to wait for my code to run as long as I have waited before… The implementation above only makes use of one of the eight cores (or if it has SMT-capabilities of one of the sixteen logic core).
So, no – according to normal standards it hasn’t been a good investment.

Für die Mutigen: mal nach map/reduce googeln!

Ok, done!