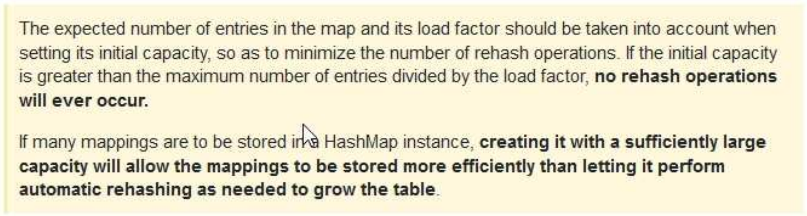
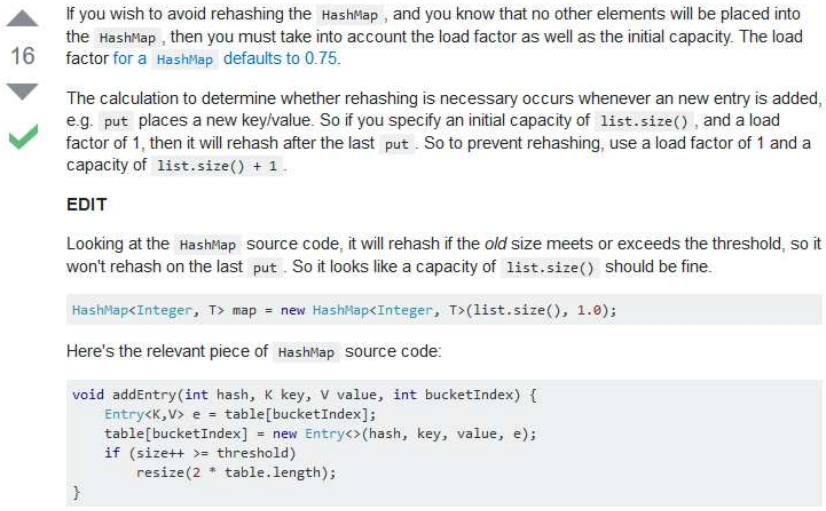
LI3 1516 - NOTA SOBRE OPTIMIZAÇÃO DO USO DE MAPS

As duas notas seguintes foram extraídas da documentação Java sobre Maps em geral.
A sua aplicação no TP de LI3-Java poderá melhorar a eficiência da inserção dos milhões de pares chave-valor a inserir nos maps.



The expected number of entries in the map and its load factor should be taken into account when setting its initial capacity, so as to minimize the number of rehash operations. If the initial capacity is greater than the maximum number of entries divided by the load factor, **no rehash operations will ever occur.**

If many mappings are to be stored in a HashMap instance, **creating it with a sufficiently large capacity will allow the mappings to be stored more efficiently than letting it perform automatic rehashing as needed to grow the table**.



16

If you wish to avoid rehashing the `HashMap`, and you know that no other elements will be placed into the `HashMap`, then you must take into account the load factor as well as the initial capacity. The load factor for a `HashMap` defaults to 0.75.

The calculation to determine whether rehashing is necessary occurs whenever an new entry is added, e.g. `put` places a new key/value. So if you specify an initial capacity of `list.size()`, and a load factor of 1, then it will rehash after the last `put`. So to prevent rehashing, use a load factor of 1 and a capacity of `list.size() + 1`.

EDIT

Looking at the `HashMap` source code, it will rehash if the *old* size meets or exceeds the threshold, so it won't rehash on the last `put`. So it looks like a capacity of `list.size()` should be fine.

```
HashMap<Integer, T> map = new HashMap<Integer, T>(list.size(), 1.0);
```

Here's the relevant piece of `HashMap` source code:

```
void addEntry(int hash, K key, V value, int bucketIndex) {
    Entry<K,V> e = table[bucketIndex];
    table[bucketIndex] = new Entry<>(hash, key, value, e);
    if (size++ >= threshold)
        resize(2 * table.length);
}
```

F. Mário Martins 31/05/2016