Vorgehensweise und Entscheidungsfindung

Für die Lösung der Aufgabe haben wir zwei Klassen entworfen.

class ListScenario (main-Methode)
In der Klasse ListScenario haben wir eine globale Variable ITERATIONS, um die Anzahl der Testdurchläufe zu definieren. Weiterhin gibt es die Methode doBenchmark(). In der main-Methode rufen wir diese Methode mit den drei Listen aus java.util und unseren eigenen Implementierungen auf.
Als Ausgabe erhalten wir die gewünschten Informationen (Zeit, Speichernutzung).
class ListeAdapter
Damit die Methode "doBenchmark()" in der Klasse ListScenario auch mit den Listen-Implementierungen des JDK funktionieren, wurde der ListeAdapter entworfen. Er implementiert unser Interface "Liste" aus a01/a02 und kapselt eine java.util.List.

Beispielhafte Ausgabe auf einem Testsystem (liegt auch als ausgabe.txt dem package bei):

```
*****************************************
Test für Liste: java.util.LinkedList
add(Element): Zeit: 14ms , Speicher: 2191696 byte
get(pos): Zeit: 1779ms
remove("abc"): Zeit: 0ms
remove(pos): Zeit: 12ms
*****************************************
Test für Liste: java.util.ArrayList
add(Element): Zeit: 10ms , Speicher: 993752 byte
get(pos): Zeit: 9ms
remove("abc"): Zeit: 0ms
remove(pos): Zeit: 2723ms
*****************************************
Test für Liste: java.util.concurrent.CopyOnWriteArrayList
add(Element): Zeit: 3005ms , Speicher: 411128568 byte
get(pos): Zeit: 5ms
remove("abc"): Zeit: 3ms
remove(pos): Zeit: 10246ms
*****************************************
Test für Liste: a01.LinkedList
add(Element): Zeit: 10ms , Speicher: 2424560 byte
get(pos): Zeit: 2043ms
remove("abc"): Zeit: 7ms
remove(pos): Zeit: 11ms
*****************************************
Test für Liste: a02.ArrayList
add(Element): Zeit: 6ms , Speicher: 1708752 byte
get(pos): Zeit: 4ms
remove("abc"): Zeit: 1437ms
remove(pos): Zeit: 1397ms
```

Martin Slowikowski
Matrikelnummer: 199 91 66

Jan-Tristan Rudat
Matrikelnummer: 200 78 52

Erklärungen

- Beide LinkedList benötigen mehr Speicher als die ArrayList-Varianten, da hier pro Element jeweils noch "prev" und "next" gespeichert werden

- CopyOnWriteArrayList braucht so viel Speicher, da das Array bei jedem Schreibzugriff geclont wird. Ein Durchlauf des Garbage-Collectors würde hier natürlich wieder viel Speicher freigeben.

- Die ArrayListen sind bei get-Zugriffen wesentlich schneller da direkte Zugriffe auf die Felder erfolgen können. Die LinkedListen müssen hierbei immer maximal die Hälfte der Liste durchlaufen

- Das Entfernen bei den LinkedListen ist wesentlich schneller da kein umkopieren der nachfolgenden Elemente erfolgen muss, sondern lediglich die Werte "next" und "prev" des vorhergehenden und des nachfolgenden Elements geändert werden müssen

- Die extremen Unterschiede bei der Methode "remove("abc")" zwischen java.util.LinkedList und a01.LinkedList sowie java.util.ArrayList und a02.ArrayList kommen daher, dass die Implementierung der JVM lediglich das erste gefundene Element entfernt. Unsere Implementierung entfernt hingegen alle Elemente die mit dem angegebenen übereinstimmen.

Martin Slowikowski
Matrikelnummer: 199 91 66

Jan-Tristan Rudat
Matrikelnummer: 200 78 52