

Inhalt

1. Einleitung	2
2. Testergebnisse	2
Bild	4
3. Fazit	4
4. Allgemein über LinkedList und ArrayList	5
ArrayList	5
Vorteile:	5
Nachteile:	5
LinkedList	5
Vorteile:	5
Nachteile:	5
Fazit:	5
5. String-Verkettung vs StringBuilder	6
String-Verkettung	6
StringBuilder	6
Testergebnisse & Fazit	6
6. StringBuffer vs. StringBuilder	6
StringBuilder (empfohlen)	6
StringBuffer	6
Quelle	7

1. Einleitung

In diesem Test wurden die Methoden der zwei List Klassen (ArrayList und LinkedList) mit Performance getestet. Dafür wurden verschiedene Methoden erstellt, womit man letztendlich messen könnte wie lang jeweils die zwei brauchen. Die Methoden sind für das Hinzufügen, Löschen und Ausgeben der Elemente.

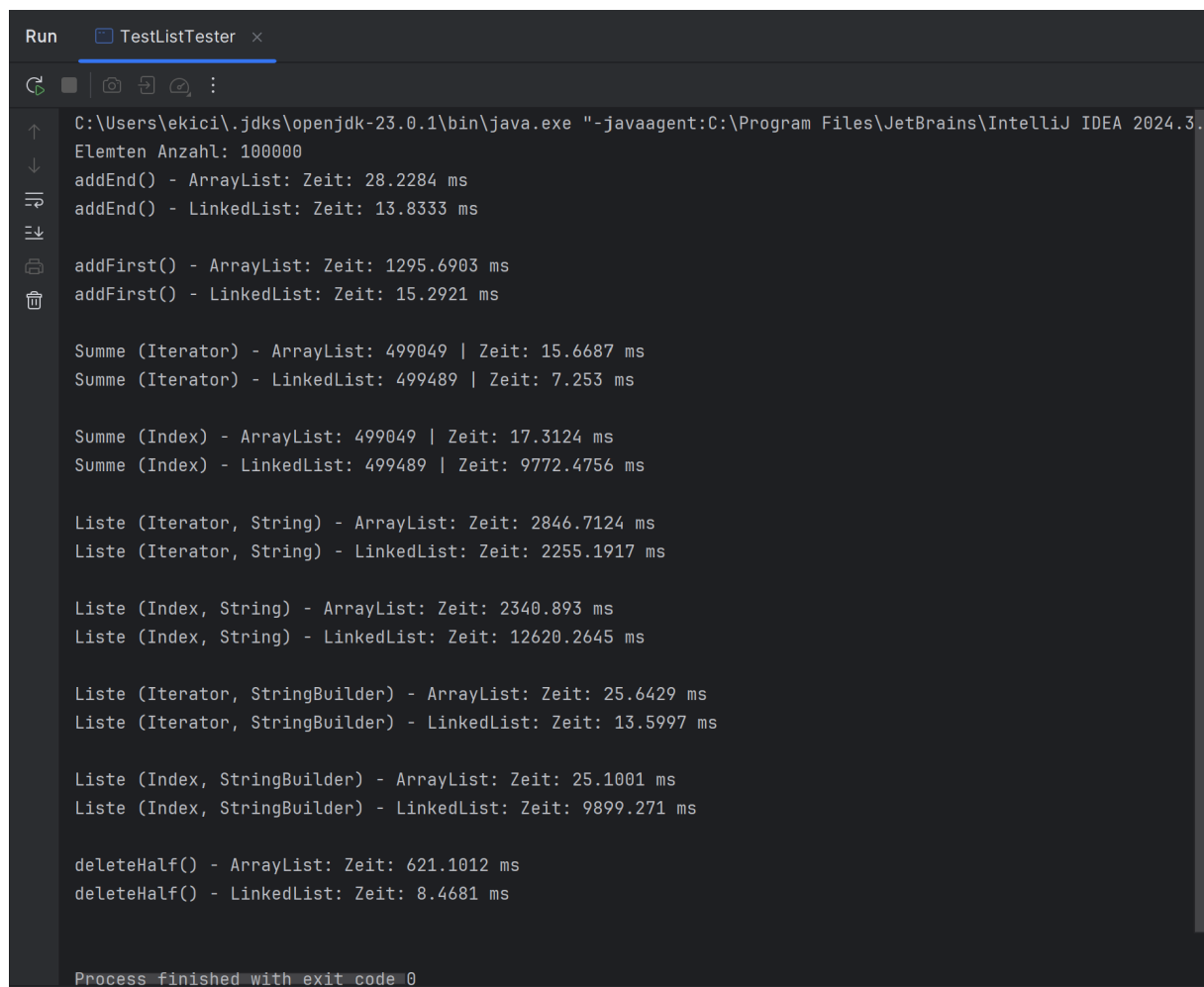
Am Ende des Testes wird ebenso der Unterschied zwischen String Verkettung und StringBuilder erklärt und auch getestet.

2. Testergebnisse

- **addEnd():**
 - ArrayList: 28,2284 ms
 - LinkedList: 13,8333 ms
 - *Interpretation:* Da ArrayList intern ein Array verwendet, kann das Hinzufügen am Ende problematisch sein, wenn das interne Array vergrößert werden muss. LinkedList speichert die Elemente als verkettete Knoten (Node), wodurch das Hinzufügen am Ende schneller ist.
- **addFirst():**
 - ArrayList: 1295,6903 ms
 - LinkedList: 15,2921 ms
 - *Interpretation:* Das Einfügen an den Anfang ist für ArrayList teuer, da alle Elemente verschoben werden müssen. LinkedList kann direkt am Anfang einfügen, was viel effizienter ist.
- **summeIterator():**
 - ArrayList: 15,6687 ms
 - LinkedList: 7,253 ms
 - *Interpretation:* Der Iterator ist für LinkedList effizienter, da er sich gut durch die verkettete Struktur bewegt. ArrayList hat jedoch auch eine schnelle Iteration durch das interne Array.
- **summeIndex():**
 - ArrayList: 17,3124 ms
 - LinkedList: 9772,4756 ms

- *Interpretation:* Der Zugriff per Index ist bei ArrayList effizient, da die Elemente in einem Array gespeichert sind. Bei LinkedList muss jedoch jedes Element von vorne durchlaufen werden, was die Zugriffszeit erheblich verlängert.
- **listIterator():**
 - ArrayList: 2846,7124 ms
 - LinkedList: 2255,1917 ms
 - *Interpretation:* Beide Methoden sind relativ langsam, aber LinkedList hat hier leichte Vorteile aufgrund der natürlichen Struktur.
- **listIndex():**
 - ArrayList: 2340,893 ms
 - LinkedList: 12620,2645 ms
 - *Interpretation:* Der Indexzugriff ist für LinkedList ineffizient, da für jedes Element eine lineare Suche durchgeführt werden muss.
- **listIteratorStringBuilder():**
 - ArrayList: 25,6429 ms
 - LinkedList: 13,5997 ms
 - *Interpretation:* StringBuilder verbessert die Performance beider Strukturen erheblich. LinkedList ist hier etwas effizienter.
- **listIndexStringBuilder():**
 - ArrayList: 25,1001 ms
 - LinkedList: 9899,271 ms
 - *Interpretation:* Ähnlich wie bei listIndex() hat LinkedList durch den ineffizienten Indexzugriff eine sehr hohe Laufzeit.
- **deleteHalf():**
 - ArrayList: 621,1012 ms
 - LinkedList: 8,4681 ms
 - *Interpretation:* Das Löschen von Elementen ist für LinkedList sehr effizient, da nur Zeiger umgehängt werden. Bei ArrayList müssen Elemente verschoben werden, was Zeit kostet.

Bild



```
Run TestListTester x
C:\Users\ekici\.jdk\openjdk-23.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2024.3.
Elemten Anzahl: 100000
addEnd() - ArrayList: Zeit: 28.2284 ms
addEnd() - LinkedList: Zeit: 13.8333 ms

addFirst() - ArrayList: Zeit: 1295.6903 ms
addFirst() - LinkedList: Zeit: 15.2921 ms

Summe (Iterator) - ArrayList: 499049 | Zeit: 15.6687 ms
Summe (Iterator) - LinkedList: 499489 | Zeit: 7.253 ms

Summe (Index) - ArrayList: 499049 | Zeit: 17.3124 ms
Summe (Index) - LinkedList: 499489 | Zeit: 9772.4756 ms

Liste (Iterator, String) - ArrayList: Zeit: 2846.7124 ms
Liste (Iterator, String) - LinkedList: Zeit: 2255.1917 ms

Liste (Index, String) - ArrayList: Zeit: 2340.893 ms
Liste (Index, String) - LinkedList: Zeit: 12620.2645 ms

Liste (Iterator, StringBuilder) - ArrayList: Zeit: 25.6429 ms
Liste (Iterator, StringBuilder) - LinkedList: Zeit: 13.5997 ms

Liste (Index, StringBuilder) - ArrayList: Zeit: 25.1001 ms
Liste (Index, StringBuilder) - LinkedList: Zeit: 9899.271 ms

deleteHalf() - ArrayList: Zeit: 621.1012 ms
deleteHalf() - LinkedList: Zeit: 8.4681 ms

Process finished with exit code 0
```

3. Fazit

- ArrayList ist vorteilhaft für schnelle Indexzugriffe und Iterationen.
- LinkedList ist effizienter beim Einfügen und Löschen von Elementen.
- LinkedList sollte nicht für häufige Indexzugriffe verwendet werden.
- Der Einsatz von StringBuilder kann die Performance beim String-Konkatenieren erheblich verbessern.

4. Allgemein über LinkedList und ArrayList

ArrayList

Vorteile:

- Schneller Direktzugriff (Konstante Zugriffszeit) → Da die ArrayList intern ein Array verwendet, kann sie Elemente über den Index sofort finden, ohne durch die Liste zu laufen.
- Geringerer Speicherverbrauch → Speichert nur die eigentlichen Werte, ohne zusätzliche Verweise auf vorherige und nächste Elemente.

Nachteile:

- Langsames Einfügen und Löschen (Lineare Verschiebung notwendig) → Wenn ein Element in der Mitte eingefügt oder entfernt wird, müssen alle nachfolgenden Elemente verschoben werden.
- Teure Speicheranpassung (Neuallokation erforderlich) → Wenn die Liste wächst und das interne Array zu klein wird, muss ein neues Array erstellt und alle Elemente kopiert werden.

LinkedList

Vorteile:

- Schnelles Einfügen und Löschen (Direktes Anpassen der Verweise) → Kein Verschieben von Elementen, nur die Verkettungen werden angepasst.
- Flexible Speicherverwaltung (Dynamische Größe) → Wächst und schrumpft ohne Speicher-Neuzuweisung.

Nachteile:

- Langsamer Zugriff (Durchlaufen der Liste notwendig) → Um ein bestimmtes Element zu finden, muss man die Liste von Anfang an durchgehen.
- Höherer Speicherverbrauch (Zusätzliche Verweise auf andere Elemente) → Jedes Element speichert zusätzlich Referenzen auf seinen Vorgänger und Nachfolger.

Fazit:

- Verwende ArrayList, wenn du oft auf Elemente zugreifen musst.
- Verwende LinkedList, wenn du viele Elemente einfügst oder löschst.

5. String-Verkettung vs StringBuilder

String-Verkettung

- Problem: Strings sind unveränderlich (immutable). Bei jeder Verkettung (liste = liste + element) wird ein neues String-Objekt erstellt und die alten Werte kopiert.
- Folge: Sehr langsam und hoher Speicherverbrauch, besonders bei großen Listen.

StringBuilder

- Lösung: StringBuilder verändert den String direkt, ohne neue Objekte zu erzeugen.
- Vorteil: Deutlich schneller und speichereffizienter, da er intern ein veränderbares Array verwendet.

Testergebnisse & Fazit

- Mit String dauert das Auflisten sehr lange, da viele unnötige Objekte erstellt werden.
- Mit StringBuilder geht es wesentlich schneller, weil er direkt am Speicher arbeitet.
- Immer StringBuilder nutzen, wenn viele String-Operationen durchgeführt werden!

6. StringBuffer vs. StringBuilder

Beide Klassen sind mutable (veränderbar) und effizienter als die normale String-Verkettung (+).

StringBuilder (empfohlen)

- Schnell → Keine Synchronisation (nicht thread-safe).
- Für Single-Thread-Anwendungen optimal (z. B. unsere Tests mit listIteratorStringBuilder).

StringBuffer

- Thread-sicher, weil synchronisiert.
- Langsamer als StringBuilder, da jede Methode synchronisiert ist.

Quelle

- [1] „When to use LinkedList over ArrayList in Java_ - Stack Overflow“.
<https://stackoverflow.com/questions/322715/when-to-use-linkedlist-over-arraylist-in-java>
- [2] „ArrayList vs LinkedList in Java - GeeksforGeeks“.
<https://www.geeksforgeeks.org/arraylist-vs-linkedlist-java/>
- [3] „ArrayList vs LinkedList - Tpoint Tech“.
<https://www.tpointtech.com/difference-between-arraylist-and-linkedlist>
- [1] „Java String, StringBuffer und StringBuilder im Vergleich“.
<https://www.centron.de/tutorial/java-string-stringbuffer-und-stringbuilder-im-vergleich/>
- [2] „StringBuilder vs. Stringverkettung ☞_ Java - Hilfe _ Java-Forum.org“.
<https://www.java-forum.org/thema/stringbuilder-vs-stringverkettung.50804/>