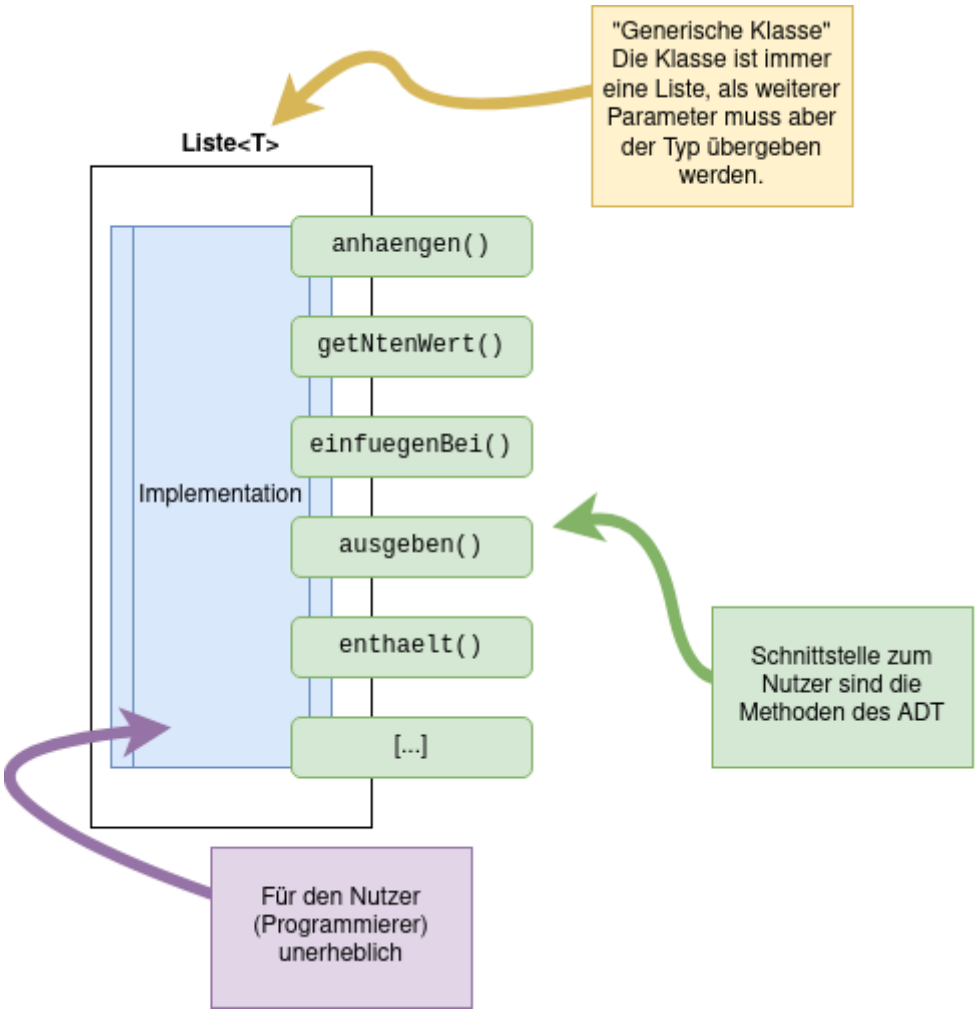
Benchmarking verschiedener Listenimplementationen

Die Verkettete Liste ist ein Beispiel für einen **Abstrakten Datentyp (ADT)**. Eine Liste enthält beliebig viele Werte eines Typs T, z.B. Integer-Zahlen. Zudem sind *Operationen* definiert, die mit der Liste durchgeführt werden können, z.B. `anhaengen(val: T)`. In der objektorientierten Programmierung entspricht das den Methoden.

Die Namen der Operationen und ihre Signatur (also Parameter und Rückgabewert) können in Java durch eine **abstrakte Oberklasse** oder ein **Interface** festgelegt werden – dies ist die ***syntaktische*** Beschreibung.

Allerdings sagen die Methoden und ihre Signaturen nichts über das ***Verhalten*** des ADTs aus, das ist die ***Semantik***. Wir müssen also beschreiben, was die Methode `anhaengen(val: T)` tut. Prinzipiell könnte die Methode `anhaengen` einfach einen leeren Rumpf haben – syntaktisch wäre sie damit korrekt, da es sich um eine Methode ohne Rückgabewert handelt.

Ein Programmierer, der die Klasse verwendet, stellt sich aber etwas anderes unter "anhängen" vor. Üblicherweise bedeutet "anhängen", dass das neue Element hinter den vorhandenen Werten angefügt wird. Man könnte sich aber auch vorstellen, dass das neue Element vorne angefügt wird oder nur dann, wenn es noch nicht vorhanden ist oder an der richtigen Stelle im Bezug auf eine Sortierung…[1)]



(A1)

Beschreibe das Verhalten der folgenden Methoden im Kontext einer verketteten Liste so eindeutig wie möglich. Was genau sollen diese Methoden machen, was geben Sie zurück, welche Voraussetzungen müssen evtl. erfüllt sein, damit man sie auf die verkettete Liste anwenden kann?

1. `vorneAnfuegen(val: T)`
2. `erstesElement(): T`
3. `letztesElement(): T`
4. `findeErstesVorkommen(val: T): int`

Implementationsvarianten

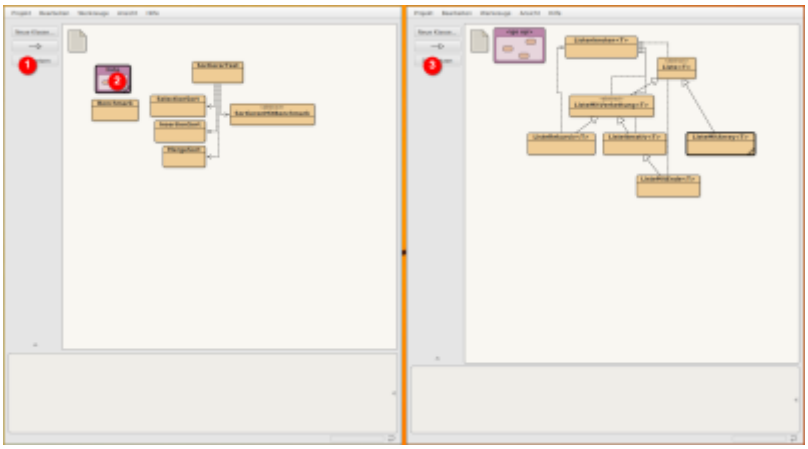
Eine Eigenschaft abstrakter Datentypen (ADTs) ist, dass sie keine Angaben über ihren inneren Aufbau machen, sondern nach außen nur ihre Methoden anbieten und deren Verhalten definieren.

Eine Liste kann man auf verschiedenste Weisen implementieren, wir wollen jetzt testen, ob sich verschiedene Implementationen verschieden verhalten.

Im Repository https://codeberg.org/Info-Unterricht/verkettete-liste-java-benchmark.git findest du verschiedene Implementationen einer verketteten Liste sowie eine Benchmark-Klasse.

(A2)

Checke das Repo aus und übersetze den Code in beiden Verzeichnissen. Übersetze auch die Klassen im Unterordner "Liste", sonst können die Tests nicht durchgeführt werden.



(A3)

Untersuche den Code der Listen-Klassen(n).

- Wo werden die Fähigkeiten der Liste - unabhängig von der konkreten Implementation - definiert?
- Warum ist die Klasse `ListeMitArray` keine Child-Klasse von `ListeMitVerkettung`?
- Warum spielt das keine Rolle für einen Programmierer, der die Klasse `Liste` verwendet?

Benchmarking

Die Benchmark-Klasse führt einige Tests mit den verschiedenen Listenvarianten durch und untersucht, wie viel Speicher und wie viel Zeit die Tests benötigen.

- Speichertests:
  - `speichertestFuellen`: Es werden einige zufällige Werte am Ende der Liste eingefügt.
  - `speichertestFuellenUndEntfernen`: Wie oben, nur werden danach 75% der Werte wieder entfernt.
- Laufzeittests:

- `laufzeittestHintenEinfuegen`: Es werden einige zufällige Werte am Ende der Liste eingefügt.
- `laufzeittestVorneEinfuegen`: Es werden einige zufällige Werte am Anfang der Liste eingefügt.
- `laufzeittestZufaelligEinfuegen`: Es werden einige zufällige Werte an zufälligen Positionen in der Liste eingefügt
- `laufzeittestAufsummieren`: Es werden einige zufällige Werte am Ende der Liste eingefügt. Dann wird mit einer for-Schleife über jedes Element der Liste iteriert und die Summe der Werte bestimmt. Hier wird nur das Iterieren über die Elemente der Liste gemessen.

Die Anzahl der Elemente steigt dabei in Zehnerschritten von 10 bis 1000 an. Die Ausgaben der Testmethoden sehen in etwa wie folgt aus:

```
Anzahl;ListeMitArray;ListeRekursiv;ListeIterativ;ListeMitEnde
10;240;416;416;424
20;440;816;816;824
....
```

(A4)

Führe die Test-Methoden der Klasse Benchmark aus und kopiere die Konsolenausgabe in eine Tabellenkalkulation wie z.B. LibreOffice Calc. Trenne die Ausgabe am Semikolon in Spalten.

Visualisiere mit einem Linien- oder xy-Diagramm, wie sich der Speicherbedarf und die Laufzeit verhalten, wenn die Anzahl der Listenelemente anwächst. Schätze ab (O-Notation), wie sich der Speicher- und Zeitbedarf in Abhängigkeit der Elementanzahl n verhält. Beurteile, wie gut die jeweilige Listenvariante für den jeweiligen Anwendungsfall geeignet ist.

Untersuche den Quelltext der Klassen und finde eine Erklärung für das unterschiedliche Verhalten.

Sortieren geht immer

Ein wichtiger Aspekt abstrakter Datentypen ist, dass sich unterschiedliche Implementierungen nach außen hin logisch gleich verhalten, d.h. ein Programm, das einen bestimmten abstrakten Datentyp verwendet, muss nicht wissen, wie der Typ implementiert wurde.

In der Klasse `SortiererTest` werden verschiedene Sortieralgorithmen (`SelectionSort`, `InsertionSort` und `MergeSort`) auf dem ADT "Liste" durchgeführt. Dabei ist es unerheblich, um welche Listenimplementation es sich handelt – es werden nur Methoden verwendet, die im ADT definiert werden und diese verhalten sich in allen Implementationen gleich.

(A5)

Ersetze in der Methode `testDurchfuehren` den Ausdruck `null` durch den Konstruktor-Aufruf einer Implementation von "Liste" und führe anschließend die Methode `testDurchfuehren` mit Anzahlen zwischen 1.000 und 20.000 aus. Überzeuge dich, dass die Sortierverfahren mit allen Implementationen der Liste funktionieren.

Vergleiche die gemessenen Laufzeiten – wodurch könnten die Unterschiede verursacht sein?

[1)]

Die Bedeutung des Begriffs "anhängen" (englisch „to append“) ist relativ klar. Die Java-Klasse `ArrayList` hat dafür die Methode `add` („hinzufügen“), die sprachlich offen lässt, wo der neue Wert eingefügt werden soll. Hier muss die *Dokumentation* konsultiert werden, die klarstellt, dass der neue Wert ans Ende der Liste kommt.