2. Arbeitsblatt: Einfach verkettete Listen

Sie haben die Grundstruktur einer einfach verketteten Liste kennengelernt. Der erste Teil dieses Arbeitsblatts soll Sie anleiten, eine Liste zu implementieren, welche die Funktionalitäten des Java-*List*-Interfaces anbietet. Der zweite Teil leitet Sie an, einen *Stack* zu implementieren, der Ihre Listen-Implementation als interne Struktur verwendet.

Importieren Sie dazu das Programmier-Projekt *ch.fhnw.algd2.lists* als Gradle-Projekt, vom Fileserver. Beachten Sie folgendes:

- Die zu implementierenden Klassen enthalten alle eine einfache `main`-Methode am Ende, die sich eignet, um einen ersten Eindruck von der Implementation zu bekommen.
- Überlegen Sie sich die asymptotische Zeitkomplexität für die jeweiligen Operationen bei den einzelnen Teilaufgaben und notieren Sie diese.
- Für jede Teilaufgabe existiert eine eigene Klasse mit Unit-Tests. Das Präfix im Klassennamen, z.B. „*A*_“, entspricht jeweils der Teilaufgabe. Sie können diese Unit-Tests einzeln ausführen, um die entsprechende Teilaufgabe separat zu testen. Es empfiehlt sich jedoch jeweils gleich alle Tests auszuführen (Rechtsklick auf den test-Ordner -> Run as -> JUnit Test). Dies hat den Vorteil, dass nur eine Run-Configuration erstellt werden muss. Zusätzlich sehen Sie später gleich, ob alle vorangehenden Tests nach wie vor erfolgreich sind. Fehlgeschlagene Tests späterer Aufgaben können einfach ignoriert werden.
- Wichtig: Testen Sie jede Teilaufgabe mit dem entsprechenden Unit-Tests und gehen Sie erst zur nächsten, wenn alle Tests erfolgreich durchlaufen werden. Die Tests der nachfolgenden Teilaufgaben hängen von vorherigen ab und erwarten korrektes Verhalten Ihrer Implementierung.

Machen Sie sich mit der Struktur des Templates vertraut, indem Sie die Listen-Grafik des Abschnitts 2.3 in den Unterrichtsunterlagen ergänzen und mit den Namen der Klassen und Variablen beschriften:

- Schreiben Sie zu jedem Viereck, zu welcher Klasse das Objekt gehört
- Beschriften Sie die Pfeile / Referenzen zwischen den Objekten mit den entsprechenden Variablennamen

Das Template verwendet die gleichen Definitionen, wie der unter der Graphik angegebene Source-Code.

1. Implementation einer SinglyLinkedList

A. Implementieren der Add-Methode

Implementieren Sie als erstes die Methode `public boolean add(E e)`. Zeichnen Sie zuerst in der Grafik zu Beginn der Seite 2 (bei `add`) ein, wie die Liste nach dem Einfügen eines weiteren Elements aussieht. Nummerieren Sie zusätzlich, in welcher Reihenfolge Sie welche Schritte ausführen (z.B: 1.) Referenz von *A* nach *B* setzen, 2.) Objekt XY erzeugen usw.). Ob Ihre Implementierung funktioniert sehen Sie, wenn alle Tests der Klasse `A_MyLinkedListTest_Add` erfolgreich sind.

Asymptotische Zeitkomplexitäts-Klasse Ihrer Implementation:

B. Implementieren der Contains-Methode

Überlegen Sie sich anhand der ergänzten Listen-Grafik auf Seite 1 im Skript, wie Sie einen Test programmieren würden, ob die Liste einen bestimmten Wert enthält. Implementieren Sie dann die Methode `public boolean contains(Object o).`

Asymptotische Zeitkomplexitäts-Klasse Ihrer Implementation:

C. Implementieren der Remove-Methode

Zeichnen Sie in die Grafik unter `remove` im Skript auf der Seite 2, welche Änderungen vorgenommen werden müssen, wenn Sie ein Objekt Ihrer Wahl entfernen würden. Implementieren Sie die Methode `public boolean remove(Object o)`. Einer der Unit-Tests funktioniert nur, wenn `add(e)` das Element `e` am Ende der Liste einfügt, wie es der Definition des `List`-Interface entspricht.

Asymptotische Zeitkomplexitäts-Klasse Ihrer Implementation:

D. Implementieren der Methoden Add und Remove über Index-Zugriffe

Das Interface *List* erweitert das Collection-Interface und ergänzt dies u.a. um Methoden, die direkt auf das *n*-te Element in der Liste zugreifen. So können Elemente an frei gewählten Positionen hinzugefügt, entfernt oder ausgelesen werden. Implementieren Sie einige solche Methoden:[1]

- `public E remove(int index)`
- `public void add(int index, E element)`
- `public E get(int index)`

Überlegen Sie sich vor der Implementation, welche Änderungen nötig sind, wenn Sie das neue Element an Position 2 einfügen möchten. Ergänzen Sie die Grafik auf der Seite 2 im Script (unter `add(2, e)`)

Asymptotische Zeitkomplexitäts-Klasse für Index-Operationen:

E. (Optional) Erlauben Sie null-Elemente in Ihrer Liste

Bisher wurde nicht geprüft und definiert, wie sich Ihre Liste verhalten soll, wenn als Element *null* übergeben wird. Passen Sie Ihre Implementierung so an, dass null-Werte erlaubt sind.

2. Stack-Implementation basierend auf Ihrer Liste

Voraussetzung für die Lösung dieser Aufgabe ist, dass in Ihrer Liste die Teilaufgaben A bis D erfolgreich gelöst wurden.

Die Dateien für diese Aufgabe befinden sich im Paket `ch.fhnw.algd2.collections.list.stack`.

Das Interface `IStack` definiert die Methoden, die Ihre Stack-Implementierung zur Verfügung stellen soll. Eine Spezifikation des Verhaltens finden Sie in der Spezifikation der Klasse `java.util.Stack`[2].

Implementieren Sie die Klasse `MyStack` samt der im Interface `IStack` definierten Operationen. Der Stack soll ein Objekt der Klasse `MyLinkedList` verwenden, die Sie bei Aufgabe 1 programmiert haben, um Daten zu speichern. Wenn Sie die Liste geschickt benutzen, sind alle Operationen in kontanter Zeit *O*(1) ausführbar.

In der Stack-Klasse gibt es ebenfalls eine `main`-Methode, mit der sie Ihre Implementierung ausprobieren können und es gibt Unit-Tests in der Klasse `MyStackTest`.

[1] Genaues finden Sie in der Interface Doku: https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/List.html

[2] https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Stack.html