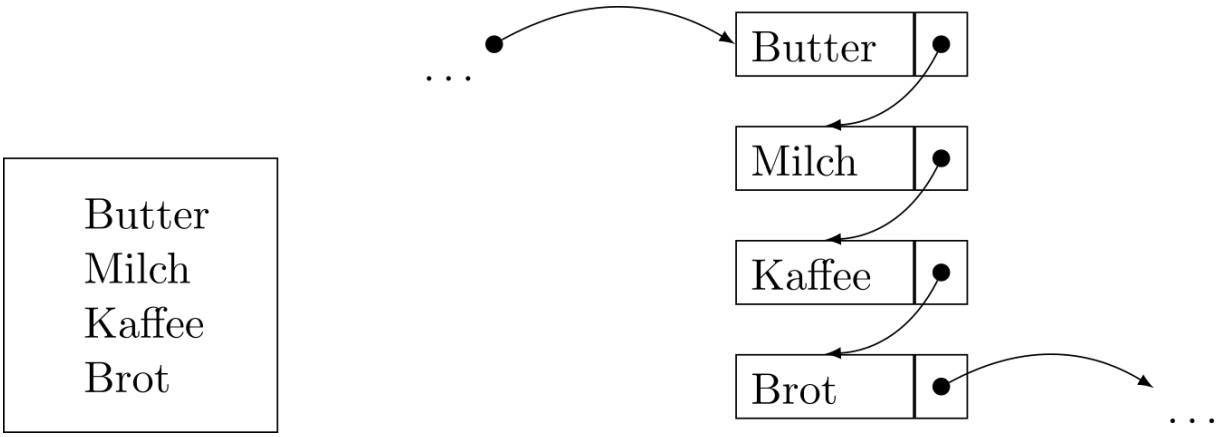
http://schuelerlabor.informatik.rwth-aachen.de

# Die Liste als dynamische Datenstruktur




**Verfasser:** Christian Graf, Tobias Reimes, Vitaly Venediktov; überarbeitet von: Thordis Dusella

**Kurz-Info:**

Informatischer Inhalt: Datenstruktur Liste, Queue und Stack
Jahrgangsstufe: EF oder Q1
Vorwissen: Programmierkenntnisse in Java, Kenntnisse in der Arbeit mit BlueJ

Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

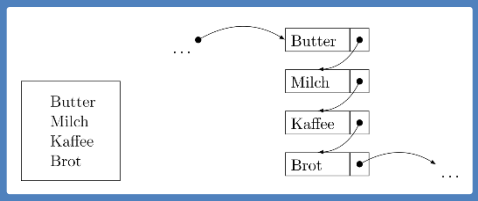
# KURZINFORMATION

**Titel:** Die Liste als dynamische Datenstruktur

**Schulstufe:** Oberstufe

**optimale Jahrgangsstufe:** EF

**geeignete Kursart:** Grundkurs




**Themenbereich:** Datenstrukturen Liste (einfach- und doppelverkettet), Stack, Queue

**Leitideen:** Das Leitprogramm kann als Einstieg in die Arbeit mit Listen als dynamische Datenstruktur genutzt werden.

**EINORDNUNG IN GESETZLICHE RAHMENBEDINGUNGEN**

**Lehrplan NRW:**

**Inhaltsfeld:** Daten und ihre Strukturierung, Algorithmen

**Kompetenzbereiche**: Implementieren, Modellierung

**Vorgaben zum Zentralabitur 2021-2023:** lineare Strukturen (Array bis zweidimensional, Stack, Queue), formale Sprachen (Java)

**Bildungsstandards der GI:**

Verwendung, Modellierung und Implementierung von Operationen auf statischen und dynamischen Datenstrukturen

**Vorkenntnisse:** Programmierkenntnisse in Java, Vertrautheit in der Arbeit mit BlueJ

**Inhaltsbeschreibung:** Die Anordnung und Abspeicherung von Daten in Datenstrukturen ist ein wichtiger Bestandteil der Informatik. Neben statischen Datenstrukturen wie Arrays, gibt es auch dynamische Varianten wie zum Beispiel Listen, Queues und Stacks. Diese werden in diesem Leitprogramm stets mit alltäglichen Beispielen erklärt und schließlich mithilfe von BlueJ in Java implementiert.



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

## INHALT

Kurzinformation ..... 2
Einführung ..... 4
Arbeitsanleitung ..... 5
Kapitel 1 – Einführung und Motivation ..... 7
Kapitel 2 – Die Liste ..... 14
Kapitel 3 – Arten von Listen: Queue und Stack ..... 26
Kapitel 4 – Implementierung von Listen ..... 35
Quellenverzeichnis ..... 44
Abbildungsverzeichnis ..... 44
Mediothek für die Schüler*innen ..... 45
Zusammenfassung der wichtigsten Inhalte ..... 46



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

## EINFÜHRUNG

Wie der Titel schon verrät, setzt du dich in diesem Leitprogramm mit Listen auseinander. Allerdings wirst du diese nicht aus einer alltäglichen, sondern einer informatischen Perspektive betrachten. Dort bezeichnet man eine Liste als eine dynamische Datenstruktur. Vielleicht bist du beim Schreiben eines Programms, zum Beispiel in Java, auch schon einmal auf das Problem gestoßen sein, dass du Daten abspeichern musstet, obwohl du beim Schreiben des Programms noch gar nicht wusstest, wie groß diese sind. Genau diese Flexibilität und weitere Vorteile bieten dir die dynamischen Datenstrukturen.

Dieses Leitprogramm soll dir die Möglichkeit bieten, dynamische Datenstrukturen kennenzulernen. Natürlich gibt es auch noch weitere Datenstrukturen. Hier soll exemplarisch die Liste erarbeitet werden, da diese sehr verbreitet ist und es recht einfach ist, schwierigere Datenstrukturen zu verstehen, wenn man die Liste schon kennt.

Um dich langsam an die Datenstrukturen heranzuführen, wird dir das Leitprogramm im ersten Kapitel zunächst einige Beispiele für Listen vorgeben; sowohl aus dem alltäglichen Gebrauch als auch solche, die schon eher als „informatische Liste“ zu bezeichnen sind. Im zweiten Kapitel wird dann die Datenstruktur „Liste“ recht abstrakt eingeführt. Dies wird allerdings noch nicht in einer Programmiersprache umgesetzt, sondern zunächst auf Papier, da es wichtig ist, die Struktur der Liste zuerst zu verstehen, bevor man diese in einen Programmcode überführt. Du wirst merken, dass es gar nicht so einfach ist, eine Liste formal korrekt zu beschreiben. Bevor du im vierten Kapitel die im zweiten Kapitel eingeführte Liste implementierst, werden im dritten Kapitel noch zwei spezielle Listenarten, nämlich Queue und Stack, vorgestellt.

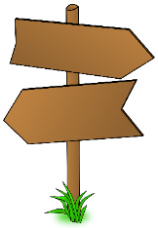
Insgesamt ist zur Bearbeitung des Leitprogramms ein Zeitrahmen von ca. sieben Unterrichtsstunden vorgesehen, wobei das zweite und vierte Kapitel als Schwerpunkte gedacht sind.



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

## ARBEITSANLEITUNG

Hier erhältst du wichtige Informationen darüber, wie du das Leitprogramm bearbeiten sollst. Sollte deine Lehrkraft dir noch zusätzliche Informationen oder Aufträge geben, so beachte diese auch bei deiner Arbeit in den nächsten Stunden. Das Leitprogramm besteht aus vier Kapiteln, die (bis auf kleine Unterschiede) vom Prinzip her gleich aufgebaut sind. Für die Bearbeitung der ersten drei Kapitel benötigst du vorrangig einige Blätter Papier und einen Stift. Erst ab dem vierten Kapitel wirst du einen Computer benötigen, da erst hier mit dem Programmieren begonnen wird. Bitte arbeite das Leitprogramm in deinem individuellen Tempo durch. Gegebenenfalls wird deine Lehrkraft dich auch bitten, zu Hause an dem Leitprogramm weiterzuarbeiten. Im Folgenden findest du eine Erklärung der wichtigsten Symbole und einen kurzen Überblick über den Aufbau der Kapitel.



**Übersicht**

Alle Kapitel beginnen mit einer kurzen Übersicht, in der kurz beschrieben wird, was dich in diesem Kapitel erwartet.

**Lernziel**

Die Lernziele beschreiben, was du nach dem Durcharbeiten des Kapitels und dem Lösen der Aufgaben können wirst.

Nach der Bekanntgabe der Lernziele beginnt direkt die Vermittlung des Stoffes. Die Erklärungstexte werden allerdings immer wieder von Aufgaben unterbrochen. Du findest sie bei den folgenden Symbolen:

**Aufgabe**

Die Aufgaben bearbeitest du alleine, wenn nichts anderes vermerkt ist. Sie dienen dem besseren Verständnis und werden auch nur von dir selbst überprüft. Nach der Bearbeitung kannst du dir die Lösungen zu den Aufgaben bei deiner Lehrkraft abholen, um deine Ergebnisse mit der Musterlösung zu vergleichen. Bearbeite die Aufgaben sehr gründlich, da sie als Vorbereitung für die Lernfortschrittskontrolle dienen. Wenn du diese Aufgaben gelöst hast, kannst du mit der Bearbeitung des Leitprogramms fortfahren.

**Übungsaufgaben am Computer**

Bearbeite diese Aufgaben am Computer. Arbeite erst weiter, wenn dein Programm läuft. Du darfst diese Aufgaben alleine oder mit einem Partner bearbeiten.

**Theorie**

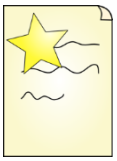
Neben den Aufgaben wird der Text auch immer wieder durch kompakte Erklärungen der Theorie ergänzt.



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

**Lernkontrolle**

Am Ende jedes Kapitels steht die Lernkontrolle. Hier kannst du überprüfen, ob du den Inhalt des Kapitels beherrschst. Auch hier gibt es die Lösungen bei der Lehrkraft, und du kontrollierst deine Ergebnisse wieder selbst.



**Additum**

Diese Aufgaben kannst du bearbeiten, wenn du dein Wissen noch vertiefen oder einfach weitertüfteln willst. Du kannst sie aber auch überspringen. Diese Aufgaben sind extra anspruchsvoller.

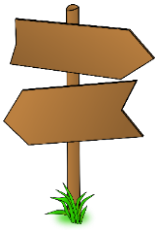
Falls du also den Test bestehst und die Mehrheit der Klasse mit dem Stoff noch nicht durch ist, warten hier zusätzliche Erklärungen und weitere Herausforderungen auf dich.

Solltest du noch Fragen haben oder einen Abschnitt eines Kapitels auch nach der dritten Durchsicht noch nicht verstehen, dann frage eine*n Mitschüler*in um Rat. Bedenke, dass sie dir zwar bei den Aufgaben helfen können, du die Lernfortschrittskontrolle aber alleine machen musst. Also versuche auf jeden Fall, die Aufgaben zu verstehen. Wenn ihr auch zusammen nicht weiterkommt, so fragt bitte eure Lehrkraft.

Wir wünschen dir viel Spaß und Erfolg beim Bearbeiten des Leitprogramms!

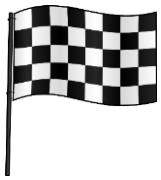


Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).



**Übersicht**

Sicherlich hast du schon viele Beispiele für Listen gesehen, dazu zählen auch simple Beispiele wie Einkaufslisten oder die Liste der Schüler*innen in eurem Klassenbuch. Auf der anderen Seite gibt es viele Beispiele, die auf den ersten Blick nicht so offensichtlich als Listen zu erkennen sind: Dazu zählt das Telefonbuch in einem Smartphone oder der Posteingang eines E-Mail-Accounts. Solche Listen wirst du mithilfe des Leitprogramms genauer betrachten und offensichtliche Unterschiede erarbeiten, um daraus die Ideen der dynamischen Liste zu entwickeln.



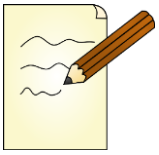
**Lernziel**

Nachdem du dieses Kapitel bearbeitet hast,

- kannst du statische von dynamischen Listen unterscheiden.
- erkennst du Vorteile dynamischer gegenüber statischen Listen.

## 1.1 DIE LISTE

Bevor du dich konkret mit Listen auseinandersetzen wirst, wirst du dir nun ein paar Gedanken über deren Notwendigkeit machen. Es gibt viele Situationen, in denen eine Menge von Informationen zusammengetragen und organisiert werden müssen. Eine von sehr vielen Möglichkeiten, dies zu erreichen, ist die Verwendung einer Liste.



**Aufgabe 1.1**

(a) Überlege dir, welche Listen dir schon in deinem Alltag begegnet sind.
(b) Schreibe auf, was du erwartest, wenn du in einem Lexikon den Begriff „Liste“ nachschlagen würdest.

Wahrscheinlich hast du festgestellt, dass es gar nicht so einfach ist, den Begriff „Liste“ zu definieren. Deshalb möchten wir hier noch mal eine mögliche Definition angeben:

**Definition: Liste**

Liste (von italienisch lista: Leiste, Papierstreifen) bezeichnet ein bestimmtes Verzeichnis oder generell eine Form von Verzeichnisstruktur. Eine Liste ist eine thematische Sammlung bestimmter Objekte, die nach einem bestimmten Prinzip angeordnet sind, beispielsweise alphabetisch oder chronologisch.

Dies beschreibt genau das, was du dir sicherlich intuitiv unter einer Liste vorgestellt hast. Allerdings wirst du im Verlauf des Leitprogramms sehen, dass die genaue Struktur einer Liste innerhalb der Informatik doch ein wenig komplexer ist. Daher wirst du dir in den nächsten beiden Abschnitten erst einmal Beispiele für solche Listen anschauen.



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

## 1.2 DER POSTEINGANG EINES E-MAIL-ACCOUNTS

Zunächst wird der Posteingang eines E-Mail-Accounts betrachtet, den du mit Sicherheit schon einmal gesehen hat. Interessant ist hier jetzt allerdings nicht der konkrete Inhalt, sondern die Struktur, wie E-Mails dargestellt, gespeichert und verwaltet werden. Ein solcher Posteingang könnte zum Beispiel so aussehen:

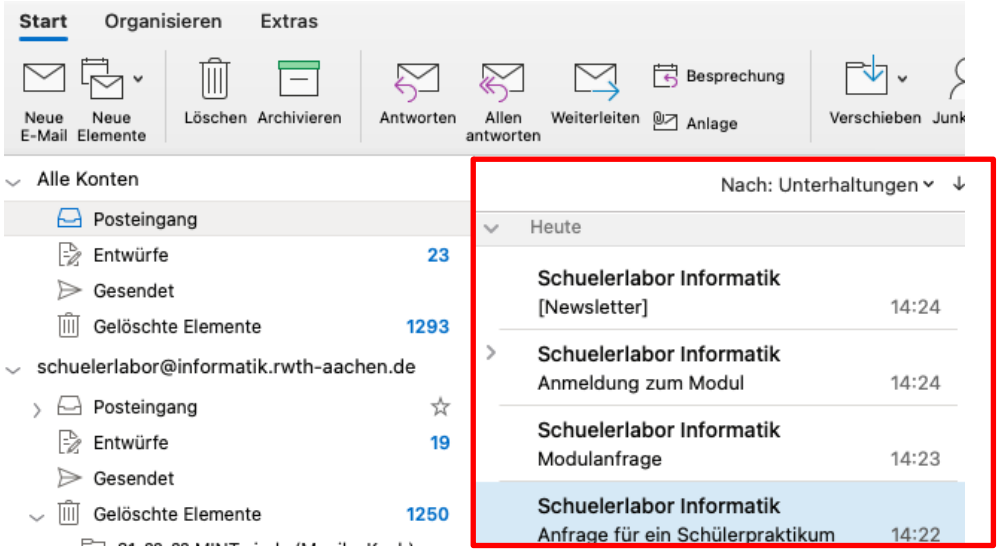
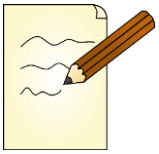



Abbildung 1: Posteingang eines E-Mail-Accounts

Jetzt stellt sich die Frage, warum es sich hier um eine Liste handelt. Wenn du jetzt noch einmal in die Definition der Liste aus Abschnitt 1.1 schaust, stellst du fest, dass die geforderten Merkmale für den Posteingang erfüllt sind: Es handelt sich hier um eine Sammlung thematischer Objekte, in diesem Fall E-Mails. Diese sind untereinander aufgelistet, haben also eine strukturierte Darstellung. Ferner sind die E-Mails in der Regel nach ihrem Ankunftsdatum sortiert. Im Folgenden wirst du dich genauer mit der Funktionsweise auseinandersetzen.



**Aufgabe 1.2**

Schaue dir den Posteingang deines eigenen E-Mail-Account an. Überlege dir, welche Funktionen dein Posteingang besitzt, was du also im Posteingang mit deinen Mails machen kannst. Notiere die einzelnen Funktionen stichpunktartig, damit du sie nachher mit den hier dargestellten vergleichen kannst.

Wie du sicherlich festgestellt hast, besitzt dein Posteingang recht viele Funktionen. Noch zu erwähnen ist, dass hier nur der Posteingang betrachtet wird. Weitere Bestandteile eines E-Mail-Accounts, wie beispielsweise einen Spam-Ordner oder ein Adressbuch, werden nicht näher beachtet. Die genauen Funktionen des Posteingangs sind die folgenden:

- **Ankommen einer neuen Mail:** Wenn du eine neue E-Mail erhältst, dann erscheint diese ganz oben in der Liste. Die neuen E-Mails sind meistens gesondert markiert.
- **Löschen einer Mail:** Wenn du eine E-Mail gelesen und beantwortet hast, dann kannst du diese löschen oder in einen anderen Ordner verschieben. Damit verschwindet diese E-Mail aus dem Posteingang.
- **Betrachten einer Mail:** Wenn du eine E-Mail ausgesucht hast, die du lesen willst, dann klickst du auf diese und kannst sie lesen. Später kannst du diese dann, wenn es nötig ist, beantworten.

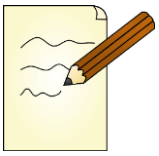


Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

- **Sortieren:** Standardmäßig sind deine E-Mails nach Ankunftsdatum und -uhrzeit sortiert, so dass die aktuelleren über den älteren aufgelistet werden. Du kannst die Liste aber auch anders sortieren lassen, beispielsweise alphabetisch nach Absender*in oder Betreff.
- **Filtern:** Wenn du eine bestimmte, vielleicht schon etwas ältere, E-Mail suchst, dann kann es hilfreich sein, einen Filter anzuwenden. Zum Beispiel kannst du dir so nur die E-Mails von deinem besten Freund anzeigen lassen.
- **Beantworten:** Mit dieser Funktion kannst du der*dem Absender*in einer in deinem Posteingang vorhandenen E-Mail eine Antwort schicken.

## 1.3 DIE SCHÜLER*INNENLISTE IM VERGLEICH

Als Nächstes wird ein weiteres Beispiel betrachtet: die Liste der Schüler*innen in einem Klassenbuch. Wenn die Lehrkraft zu Beginn des Schuljahres die Namen der Schüler*innen in das Buch einträgt, dann macht er oder sie dies in alphabetischer Reihenfolge. Kommt nun während des Schuljahres jemand Neues in die Klasse, so muss diese Person ans Ende der Liste angefügt werden. Ebenso kann es sein, dass ein*e Schüler*in im Verlauf des Schuljahres die Klasse verlässt und so aus der Liste gestrichen werden muss.



**Aufgabe 1.3**

Vergleiche die beiden Beispiele 1.2 und 1.3! Beachte dabei besonders die Punkte

- Sortierung
- freie Plätze
- Einfügen
- Löschen

Es ist natürlich nicht einfach, zwei Listen zu vergleichen, die einerseits in elektronischer und andererseits in handschriftlicher Form vorhanden sind. Dies ist hier allerdings kein Problem, da du nur die Eigenschaften der beiden Listen vergleichst. Welche Liste einfacher zu handhaben oder zu verwalten ist, wird zunächst außenvor gelassen. Beim Vergleich der beiden Beispiele 1.2 und 1.3 stellt man einige Unterschiede zwischen diesen beiden Listen fest. Zunächst sieht man, dass eine Liste der Schüler*innen auf eine bestimmte Anzahl an Schüler*innen begrenzt ist, im Gegensatz zum Posteingang, in dem theoretisch unendlich viele E-Mails abgelegt werden können. Kommen neue Schüler*innen in eine Klasse, können diese auf der Liste nur noch hinter den letzten Eintrag angefügt werden, wodurch die alphabetische Sortierung verloren geht. Wird ein Name auf der Liste durchgestrichen, bleibt dieser Platz in der Liste trotzdem besetzt; es entsteht also eine Art Lücke in der Liste. Selbst wenn man nun mit Tipp-Ex das Feld wieder so herrichten würde, dass man einen neuen Namen dort eintragen könnte, ist es in der Regel unwahrscheinlich, dass ein neu hinzukommender Name von der alphabetischen Sortierung her genau in diese Lücke hineinpassen würde. Beim Posteingang handelt es sich also um eine dynamische, bei der Schüler*innenliste um eine statische Liste. Eine dynamische Liste bietet, wie du sicherlich schon gemerkt hast, ein hohes Maß an Flexibilität, da die Liste jederzeit beliebig verändert werden kann. Was das genau bedeutet und weitere Eigenschaften einer solchen Liste wirst du im nächsten Abschnitt noch genauer kennenlernen. Im Gegensatz dazu bietet die statische Liste diesen Komfort nicht. Sie kann nach der Erstellung nur noch sehr schwer verändert werden. Diese Listen eignen sich gut für Daten, die nur selten verändert werden müssen, wären aber beispielsweise für einen Posteingang vollkommen ungeeignet, da sich dort der Inhalt in der Regel mehrmals täglich ändert.



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

## 1.4 DIE DYNAMISCHE LISTE

Nachdem du im vorangegangenen Abschnitt gesehen hast, was unter einer statischen Liste zu verstehen ist, kann nun klar festgelegt werden, was im weiteren Verlauf unter einer dynamischen Liste verstanden wird.

**Definition: dynamische Liste**

Eine Liste wird als dynamisch bezeichnet, wenn sie folgende Eigenschaften besitzt:

- Die Anzahl der Listenelemente ist nicht beschränkt oder vorher festgelegt.
- Die Listenelemente können beliebig an jeder Stelle der Liste eingefügt oder gelöscht werden.

Es kann in einer dynamischen Liste nicht vorkommen, dass durch das Löschen eines Elements eine Lücke entsteht. Wird ein Element gelöscht, „rutschen“ die Nachfolgeelemente um eine Position auf. Wird ein Element an einer bestimmten Position eingefügt, werden die Elemente dahinter um eine Stelle nach hinten verschoben. Dieser Vorteil bringt allerdings den Nachteil mit sich, dass die Erstellung einer dynamischen Liste recht aufwendig ist. Im Verlauf des Leitprogramms wirst du sehen, dass die Vorteile dieser Struktur trotzdem überwiegen. Wie diese Listen nun konkret aufgebaut und weitere Funktionen erzeugt werden können, soll in den folgenden Kapiteln erläutert werden.

**Aufgabe 1.4**

In dieser Aufgabe werden die beiden Beispiele aus der Übersicht aufgegriffen. Entscheide jeweils für die Einkaufsliste und für das Adressbuch in einem Smartphone, ob es sich um eine statische oder dynamische Liste handelt. Begründe deine Antworten!

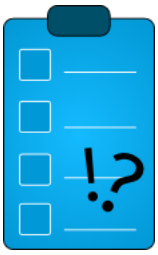
## 1.5 LÖSUNGEN DER AUFGABEN

Alle Aufgaben bearbeitet? Dann kannst du deine Lehrkraft oder das Betreuer*innenteam nach den Musterlösungen fragen, um deine Antworten mit diesen abzugleichen. Solltest du dann noch Fragen haben, dann bespreche diese bitte mit deiner Lehrkraft oder einem Mitglied des Betreuer*innenteams.



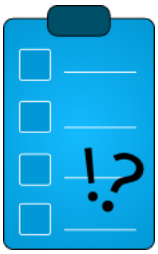
Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

## 1.6 LERNFORTSCHRITTSKONTROLLE



**Lernkontrolle 1.1**

Erläutere, warum die Liste der Schüler*innen aus Kapitel 1.3 keine dynamische Liste ist.



**Lernkontrolle 1.2**

Gib die Definition einer Liste und die Eigenschaften einer dynamischen Liste mit deinen eigenen Worten wieder.

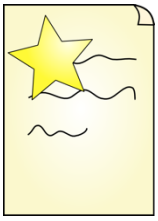


Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

## 1.7 LÖSUNGEN DER LERNFORTSCHRITTSKONTROLLE

Wenn du die Lernfortschrittskontrolle fertig bearbeitet hast, dann bitte deine Lehrkraft oder ein Mitglied des Betreuer*innenteams um die Musterlösungen, damit du deine Antworten mit diesen vergleichen kannst. Bei Fragen kannst du die Lehrkraft oder das Betreuerteam um Hilfe bitten.

## 1.8 ADDITUM

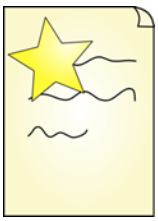


Im Additum des ersten Kapitels wird sich erstmal wieder von der Liste distanziert und auf die Datenkapselung eingegangen.

**Definition: Datenkapselung**

Kapselung bezeichnet ein wichtiges Konzept innerhalb der objektorientierten Programmierung. Unter Datenkapselung versteht man in der Programmierung das Verbergen der genauen Implementierung. Dazu gehört beispielsweise der kontrollierte Zugriff auf Methoden bzw. Attribute von Klassen.

Der direkte Zugriff auf die interne Datenstruktur wird also unterbunden und erfolgt stattdessen nur über definierte Schnittstellen. Die Schnittstellen zum Bearbeiten von Attributen einer Klasse bezeichnet man oft als Getter und Setter.



**Zusatzaufgabe 1.1**

Nachdem du nun einen groben Überblick über die Datenkapselung bekommen hast, sollst du dich im Internet weiter zu dem Thema informieren. Wenn du noch genügend Zeit hast, kannst du auch zum Thema „Information Hiding“ recherchieren.

Wahrscheinlich hast du bei deiner Recherche auch schon einiges über Vor- und Nachteile der Datenkapselung gehört. Trotzdem soll noch einmal auf die Vor- und Nachteile eingegangen werden. Die Vorteile findest du in der folgenden Aufzählung. Es ist nicht schlimm, wenn du mit einigen Begriffen dort nichts anfangen kannst. Wenn du sie verstehen möchtest, kannst du sie einfach googlen.

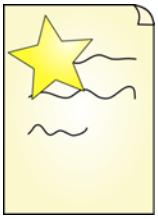
- Da die Implementierung einer Klasse anderen Klassen nicht bekannt ist, kann die Implementierung geändert werden, ohne die Zusammenarbeit mit anderen Klassen zu beeinträchtigen. Ebenso ist es dann nicht nötig, die anderen Klassen zu modifizieren.
- Erhöhte Übersichtlichkeit, da nur die öffentliche Schnittstelle einer Klasse betrachtet werden muss und nicht die genaue Implementierung.
- Beim Zugriff über eine Methode spielt es für die aufrufende Klasse keine Rolle, ob diese Funktion 1:1 im Inneren der Klasse existiert, das Ergebnis einer Berechnung ist oder möglicherweise aus anderen Quellen (zum Beispiel einer Datei oder Datenbank) stammt.
- Deutlich verbesserte Testbarkeit, Stabilität und Änderbarkeit der Software bzw. deren einzelnen Klassen.

Es gibt nur einen gravierenden Nachteil, der dir vermutlich noch nicht aufgefallen ist. Wahrscheinlich wirst du diesen auch nicht wahrnehmen, da er erst so richtig bei großen Softwareprojekten mit vielen



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

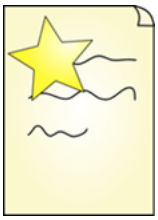
Klassen etc. eine Rolle spielt: In Abhängigkeit vom Anwendungsfall können Geschwindigkeitseinbußen bei einem Aufruf einer Methode (beispielsweise eines Getters) auftreten, anstatt direkt auf das Attribut zuzugreifen, was schneller vonstattengeht.



**Zusatzaufgabe 1.2**

Stelle dir vor, du bist ein erfolgreicher Unternehmer bzw. erfolgreiche Unternehmerin und hast gerade einen schnellen Algorithmus gefunden, um alle Primzahlen bis zu einer Zahl n zu bestimmen. Du möchtest erlauben, dass jemand die Implementierung deines Algorithmus benutzen darf. Dabei soll er/sie, aber nicht sehen, wie dieser genau funktioniert und welche Attribute er verwendet. Warum ist das Prinzip der Datenkapselung in diesem Fall für dich interessant?

Bis jetzt hast du dich eher mit dem Sinn und Zweck der Datenkapselung beschäftigt. Nun wird es etwas konkreter, indem es zum Abschluss des Additums mit der Umsetzung in der Programmiersprache Java losgeht.



**Zusatzaufgabe 1.3**

In der Programmiersprache Java gibt es für die Datenkapselung spezielle Schlüsselwörter, mit denen man die Methoden und Attribute einer Klasse kennzeichnen kann. Informiere dich beispielsweise im Internet über die Bedeutung der folgenden Schlüsselwörter:

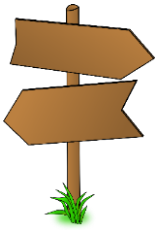
1. private
2. protected
3. public

## 1.9 LÖSUNGEN ZU DEN ADDITUM-AUFGABEN

Auch zu den Additum-Aufgaben gibt es Musterlösungen bei deiner Lehrkraft oder dem Betreuer*innenteam.



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).



**Übersicht**

Im ersten Teil dieses Kapitels wirst du eine grafische, formale Darstellung für Listen entwickeln, mit der du dann auch in den folgenden Kapiteln noch arbeiten wirst. Sie ist grundlegend für die informatische Sichtweise der Liste. Im zweiten Teil des Kapitels wirst du zusammen mit der neuen Sichtweise die auf Listen definierten Operationen (Einfügen, Suchen, Löschen usw.) erarbeiten.

**Lernziele**

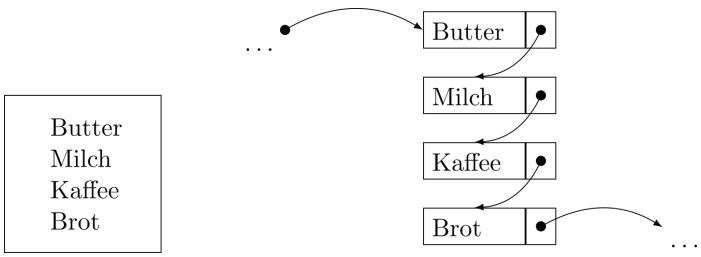
Nachdem du dieses Kapitel bearbeitet hast,

- kannst du erklären, wie eine dynamische Liste auf einem Computer verwirklicht werden kann.
- kannst du eine grafische Notation zur Veranschaulichung von Listen im Speicher eines Computers interpretieren.
- kannst du mithilfe der vorgestellten Notation die Operationen auf einer Liste visualisieren.

## 2.1 AUFBAU EINER EINFACH VERKETTETEN LISTE

Bevor du dich mit den genauen Funktionen einer Liste beschäftigen wirst, solltest du erst den Aufbau einer Liste betrachten. Du musst dir also zuerst überlegen, wie eine Liste formal beschrieben werden kann, da eine Liste auf verschiedenste Arten darstellbar ist. Berechtigterweise stellt sich jetzt die Frage: Warum ist das relevant?

Eine grafische Notation wird eingeführt, um zu verstehen, wie der Computer intern mit einer Liste arbeitet bzw. wie er eine Liste in seinem Speicher ablegt. Weiter ist es mit der neuen Schreibweise, die gleich eingeführt wird, recht einfach, die Operationen, wie beispielsweise Einfügen oder Suchen, zu erklären und zu verstehen. Später, wenn du Listen implementieren willst, hilft dir diese Darstellung ebenfalls weiter: Die Grafiken müssen „nur“ in Quellcode übersetzt werden. Anschließend hast du dann eine fertige Liste, in der du beliebige Daten abspeichern kannst. Normalerweise stehen die Listenelemente, wenn beispielsweise der Posteingang aus Kapitel 1 betrachtet wird, untereinander. Dies bedeutet bezogen auf ein einzelnes Element, dass immer ein Element unter ihm steht, es sei denn es ist das letzte Element der Liste. Dadurch entsteht eine Art Verkettung zwischen den Elementen, jedes Element hat ein nachfolgendes Element, dieses hat ebenfalls ein Nachfolgeelement, usw. Bevor nun Listenelemente formal definiert werden, soll zunächst thematisiert werden, wie man eine gewöhnliche Einkaufsliste, notiert auf einem Blatt Papier, formaler aufschreiben kann:




Wie du siehst, ist jedem einzelnen Eintrag der ursprünglichen Liste ein einzelner „Rahmen“ zugeordnet worden. Neben dem Feld, in dem der Gegenstand eingetragen ist, der eingekauft werden soll,



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

gibt es noch ein zweites Feld. In eben diesem zweiten Feld beginnt ein Pfeil, der auf den nächsten Eintrag zeigt. Du wirst dich jetzt sicherlich fragen, warum das so ist. Die Antwort ist recht einfach, aber von zentraler Bedeutung: Menschen sehen sofort, welcher Eintrag in einer Liste der nächste ist, weil sie intuitiv sehen, wie die Liste angeordnet ist und einfach feststellen können, wo sich das nächste Element befindet, da sie gelernt haben (in der deutschen Sprache), einen Text von links nach rechts und von oben nach unten zu lesen. Der Computer hingegen weiß das nicht. Da dem Computer aber beigebracht werden soll, mit Listen zu arbeiten, musst du ihm sagen, wo der nächste Eintrag steht bzw. wo er ihn finden kann. Im Folgenden soll das erreicht werden, indem zu jedem Eintrag ein Verweis hinzugefügt wird, der auf den nächsten Eintrag zeigt.

Nach der Verdeutlichung durch ein simples Beispiel anhand einer Einkaufsliste kann jetzt genau formalisiert werden, was im Folgenden unter einem Listenelement verstanden wird.

**Definition: Listenelement**

Ein Eintrag in einer Liste wird ab jetzt als Listenelement bezeichnet. Ein solches Listenelement besteht aus einem Objekt (zum Beispiel einem Text) und einem Zeiger auf ein anderes Listenelement.

Die Objekte deiner Einkaufsliste sind Lebensmittel, wobei diese neben dem Namen noch weitere Eigenschaften besitzen, beispielsweise Mengenangaben. Das Objekt „Lebensmittel“ besteht also aus einem Namen und einer Mengenangabe. Um die recht abstrakte Definition des Listenelements noch besser zu verdeutlichen, wird folgende grafische Notation eingeführt:

Hier sieht man nun, dass die einzelnen Listenelemente immer mit genau einem nachfolgenden Element verbunden sind. Der Zeiger wird hier durch einen Pfeil zwischen den beiden Elementen dargestellt. Dies bedeutet im Sinne der obigen Definition, dass man von einem Element aus seinen Nachfolger „erreichen“ kann. Dadurch entsteht zwangsläufig die Verkettung der Elemente, da jedes Element auf genau ein weiteres zeigt. Im Unterschied zu den „normalen“ Darstellungen werden die Elemente nebeneinander und nicht mehr untereinander aufgeschrieben. Dies geschieht aber eher aus Platzgründen; prinzipiell könnte man die Elemente immer noch untereinander oder beliebig verteilt aufschreiben, weil man durch die Einführung des Zeigers jederzeit weiß, wo sich das Nachfolgerelement befindet. Weiter sieht man, dass jedes Element, bis auf das Erste, einen eindeutigen Vorgänger hat. Diese Form der Darstellung wird im weiteren Verlauf immer wieder aufgegriffen, um beispielsweise die Operationen auf Listen bildlich zu verdeutlichen. Nachdem nun die Listenelemente beschrieben wurden, stellt sich noch die Frage, was jetzt noch zur gesamten Liste fehlt. Es muss bekannt sein, wo die Liste anfängt und wo sie endet, also welches Element das erste und welches das letzte ist.

**Definition: Liste**

Eine Liste besteht aus einer beliebigen Anzahl an Listenelementen. Es existiert zudem ein Zeiger auf das erste Element der Liste, der im Folgenden Kopf der Liste genannt wird. Das letzte Element der Liste ist das einzige Element, das keinen Zeiger auf ein nächstes besitzt. Auch auf dieses Element gibt es einen Zeiger, der als Fuß bezeichnet wird.

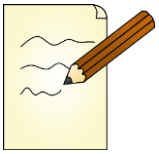
Durch diese Definition wird die Liste eindeutig beschrieben, da ausgehend vom Kopf der Liste alle Elemente miteinander verkettet wurden. Dies könnte dann zum Beispiel so aussehen:



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

Die Zeiger werden hier als Pfeile zwischen den Elementen dargestellt. Der Pfeil zwischen dem ersten und dem zweiten Element bedeutet, dass das erste Element auf das zweite Element „zeigt“. Da das letzte Element (hier: Objekt3) keinen Nachfolger besitzt, existiert auch kein Zeiger auf ein weiteres Element. An dieser Stelle ist die Liste damit zu Ende. Kopf und Fuß sind keine Elemente, sondern sind Zeiger auf das erste bzw. das letzte Element.

Nach so viel Theorie, wird es wieder Zeit, etwas praktischer zu werden:



**Aufgabe 2.1**

Du wirst von deinen Großeltern gebeten, für sie einzukaufen. Deine Großmutter gibt dir telefonisch ihre Einkaufsliste durch. Auf der Liste stehen

- ein Brot,
- 10 Frikadellen,
- 500 g Butter,
- 2 kg Rinderbraten,
- eine Gurke,
- eine Schale Pilze und
- 1 kg Tomaten.

(a) Schreibe den Einkaufszettel deiner Großeltern in der Notation auf, die gerade eingeführt wurde. Dabei soll ein Listenelement folgende Informationen für jedes Lebensmittel separat speichern:
   - Name
   - Menge

   Ein Listenelement sieht also wie folgt aus:

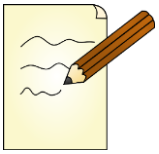
   Zeichne auch die Zeiger für Kopf und Fuß ein.

(b) Ist die folgende Aussage richtig? Wenn nein, dann verbessere sie!
„Der Kopf ist das erste und der Fuß das letzte Element der Liste.“

(c) Was unterscheidet die Listenelemente aus der Definition von den Listenelementen, die für die Lösung dieser Aufgabe vorgegeben wurden? Was schließt du daraus?

Hinweis: Schau dir dazu die Bemerkung nach der Definition „Listenelement“ an.

Bevor du dich jetzt endlich mit den „Operationen auf Listen“ beschäftigen kannst, gibt es zum Abschnittsende noch eine weitere ähnliche Aufgabe, auf die im Verlaufe des Kapitels noch zurückgegriffen werden wird.



**Aufgabe 2.2**

Es wird nun wieder der Posteingang aus Kapitel 1 betrachtet. Wie könntest du diese Liste nun mit der oben vorgestellten grafischen Notation darstellen? Versuche, eine Liste mit fünf Elementen auf Papier aufzuschreiben. Da es sich bei den Objekten in



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

den Listenelementen um E-Mails handelt, sollten diese mit ”E-Mail1“, ”E-Mail2“, usw. beschriftet werden.

## 2.2 OPERATIONEN AUF LISTEN

Um mit der formal eingeführten Liste arbeiten zu können, müssen einige Funktionsweisen der Liste genauer erklärt werden. Die Funktionen müssen so definiert werden, dass durch Verändern der Liste diese immer noch formal korrekt arbeitet. Es muss zum Beispiel jedes Element weiterhin einen Zeiger auf seinen Nachfolger besitzt, und Kopf und Fuß jeweils auf das erste beziehungsweise letzte Element zeigen. Diese angesprochenen Funktionen werden auch Operationen genannt.

**Suchen**

Als Erstes wirst du dich damit beschäftigen, wie in einer Liste bestimmte Elemente gesucht werden können. Dies erscheint gerade bei Listen mit wenigen Elementen sehr einfach. Man überschaut die Liste und sieht, ob ein Element vorhanden ist und an welcher Stelle es steht. Aus zwei Gründen sollte man sich hier aber etwas genauer damit beschäftigen. Zum einen könnte deine Liste sehr groß sein, so dass du auf Anhieb nicht siehst, ob ein Element in der Liste ist. Zum anderen ist ein Computer nicht fähig, eine Liste komplett zu überschauen, da er nicht alle Elemente zur selben Zeit betrachten kann. Man muss sich daher einen Algorithmus überlegen, mit dem systematisch ein Element in einer Liste gesucht werden kann. Dabei wird wie folgt vorgegangen:

**Algorithmus: Suchen eines Listenelements**

(1) Suche das erste Element der Liste. Dieses ist leicht aufzufinden, da vom Kopf der Liste direkt ein Zeiger auf dieses Element zeigt.
(2) Schaue nun in dieses Element hinein und prüfe, ob es das gesuchte Element ist. Wenn ja, hast du das Element schon gefunden.
(3) Wenn dieses Element nicht das gesuchte ist: Betrachte den Nachfolger des Elementes, wenn es einen Nachfolger gibt. Dieses ist durch den vorhandenen Zeiger auf das nächste Element ebenfalls einfach zu finden.
(4) Prüfe, ob dieses Element das gesuchte ist. Wenn ja, hast du das Element gefunden. Wenn nein, gehe zurück zu Schritt 3 und durchlaufe so die Liste bis an die Stelle, wo sich das gesuchte Element befindet.
(5) Durchläufst du die Liste bis zum Ende, ohne das gesuchte Element gefunden zu haben, weißt du, dass die Liste das gesuchte Element nicht enthält.

Das soll nun anhand eines Beispiels noch einmal erläutert werden. Es wird wieder der Posteingang eines E-Mail-Accounts betrachtet, diesmal aber direkt in der formalen Schreibweise.

Die Aufgabe lautet: Finde alle Mails, die vom Absender „Dietmar Scholz“ gesendet wurden. Um den Absender einer Mail festzustellen, muss in das jeweilige Listenelement hineingeschaut werden. Für das Beispiel soll davon ausgegangen werden, dass die Objekte „E-Mail3“ und „E-Mail5“ den gesuchten Absender enthalten. Nun wird wie folgt vorgegangen:

1. Beginne beim ersten Element („E-Mail1“).
2. Prüfe, ob der Absender des aktuellen Elements der Gesuchte ist. Wenn nicht, gehe zum nachfolgenden Element und wiederhole Schritt 2.
3. Das gesuchte Element (im Beispiel bei „E-Mail3“) wurde gefunden.



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

**Aufgabe 2.3**

Es gibt also einen Algorithmus, um Elemente in einer Liste zu suchen. Ist das Suchergebnis aus dem letzten Beispiel vollständig? Ergänze den oben genannten Algorithmus so, dass das Suchergebnis immer vollständig ist!

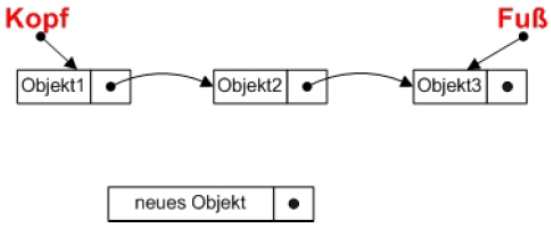
### Einfügen

Damit überhaupt etwas gesucht werden kann, muss die Liste einige Einträge enthalten. Wie kommen die Einträge in die Liste? Durch Einfügen! Um ein Element in eine Liste einzufügen, wird wie folgt vorgegangen:

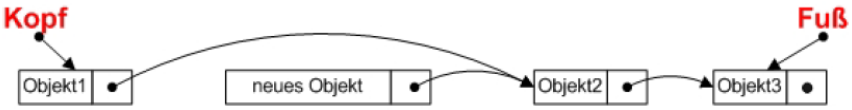
**Algorithmus: Einfügen eines Listenelements**

(1) Suche zuerst die Stelle, an der das Element eingefügt werden soll (siehe Kapitel 2.2.1). Merke dir sowohl den Vorgänger als auch den Nachfolger des einzufügenden Elementes.
(2) Setze den Zeiger des Elementes auf den Nachfolger.
(3) Setze den Zeiger des Vorgängers, der ebenfalls noch auf den Nachfolger des eingefügten Elementes zeigt, auf das neueingefügte Element.

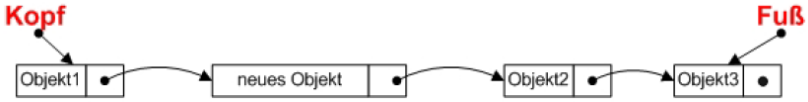
Da diese Handlungsbeschreibung sehr abstrakt ist, findest du im Folgenden noch eine graphische Darstellung. Als Beispiel dienen eine dreielementige Liste und ein neues Listenelement, das an zweiter Stelle der Liste eingefügt werden soll.




Zuerst muss die Stelle gefunden werden, an der das neue Element eingefügt werden soll. Um die zweite Stelle zu finden, muss, die Liste vom ersten Element aus durchlaufen werden, bis das Element gefunden wurde, vor dem einfügt werden soll. Dies geht hier natürlich recht schnell, da schon an der zweiten Stelle einfügt werden soll. Das Objekt „neues Objekt“ muss nun den Nachfolger „Objekt2“ erhalten, da es an der zweiten Stelle eingefügt werden soll.




Es besteht jetzt das Problem, dass das Element „neues Objekt“ ausgehend vom Kopf noch nicht erreichbar ist. Zudem besitzen zwei Elemente den Nachfolger „Objekt2“. Somit ist die Liste nicht mehr eindeutig. Daher wird der Zeiger von „Objekt1“ auf „neues Objekt“ gesetzt. Damit ist „neues Objekt“ der Nachfolger des ersten Elementes und die Liste wieder eindeutig. Das eingefügte Element ist als Teil der Liste an zweiter Stelle erreichbar.

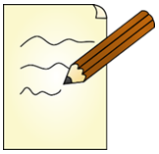





Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

Wenn die grafischen Veränderungen wieder in die Art und Weise der Speicherung im Computer übertragen wird, werden die Listenelemente nicht verschoben, sondern nur ihre Zeiger auf die Nachfolger verändert und das neue Listenelement an einer freien Stelle im Speicher abgelegt.

Je nach Verwendungszweck der Liste kann es erforderlich sein, neue Elemente immer an einer festen Stelle einzufügen, beispielsweise am Anfang oder am Ende der Liste.



**Aufgabe 2.4**

Was muss beachtet werden, wenn ganz am Anfang oder am Ende der Liste ein Element eingefügt wird? Schaue dir hierfür noch einmal die Definition der Liste an. Entscheide dich, ob du diese Aufgabe für den Anfang oder das Ende der Liste bearbeiten möchtest.

a) Schreibe dein Vorgehen formal auf, indem du vorgehst, wie unter "Suchen eines Listenelements" beschrieben.
b) Zeichne zu deiner Lösung aus a) ein Beispiel, in dem einer Liste aus drei Einträgen ein vierter Eintrag hinzugefügt wird. Nutze dazu die Notation, die du im Rahmen dieses Leitprogramms gelernt hast.

**Löschen**

Eine weitere Funktion, die eine Liste besitzen sollte, ist das Löschen von Elementen. Hierzu muss man sich überlegen, wie ein Element aus der Liste entfernt werden kann, ohne dass dabei die Struktur der Liste verloren geht.

**Aufgabe 2.5**

Schreibe auf einem Blatt Papier auf, wie eine Liste aufgebaut ist, beziehungsweise wodurch sie charakterisiert ist! Solltest du dies nicht mehr ganz im Kopf haben, kannst du natürlich noch einmal am Anfang des Kapitels nachsehen. Schreibe dir zudem eine beliebige Liste mit vier Elementen in der oben vorgegebenen grafischen Notation auf.

Zuerst muss man sich überlegen, wie ein Element, das mitten in der Liste, also nicht am Anfang oder am Ende steht, entfernt werden kann. Dies erfolgt auf folgende Weise:

**Algorithmus: Löschen eines Listenelements**

(1) Suche das Element, das gelöscht werden soll.
(2) Merke dir sowohl den Vorgänger als auch den Nachfolger des zu löschenden Elementes. Findest du dieses Element nicht in der Liste, brauchst du nichts weiter zu tun. Befolge ansonsten die beiden folgenden Schritte.
(3) Entferne nun das zu löschende Element aus der Liste.
(4) Setze den Zeiger des Vorgängerelementes auf das Nachfolgerelement.

**Aufgabe 2.6**

In der letzten Aufgabe solltest du eine beliebige Liste mit vier Elementen aufschreiben. Versuche nun das zweite Element dieser Liste mit der gerade vorgestellten Methode aus der Liste zu entfernen. Schreibe die Liste nach jedem durchgeführten Schritt neu auf.



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

Es wurde ein Element aus der Liste gelöscht. Die Listenstruktur blieb weiterhin erhalten, da die „Lücke“, die durch das Entfernen des Elementes entstanden ist, geschlossen wurde. Jetzt muss nur noch geklärt werden, wie das erste beziehungsweise das letzte Element der Liste gelöscht werden kann.

**Aufgabe 2.7**

Überlege dir, wie man das erste und das letzte Element aus einer Liste löschen kann. Versuche diese Vorgehensweise möglichst genau aufzuschreiben. Was muss an diesen beiden Stellen beachtet werden?

Man sieht, dass das Löschen am Anfang viel schneller möglich ist als das Löschen am Ende, da durch den Kopf, der auf das erste Element zeigt, direkt bekannt ist, wo sich das Element befindet und es nicht gesucht werden muss. Durch den Fuß ist zwar auch bekannt, wo sich das letzte Element der Liste befindet. Ohne vorheriges Suchen des Vorgängerelementes kann man dieses aber nicht löschen.

Als Sonderfall wäre noch eine Liste zu erwähnen, bei der das zu löschende Element sowohl das erste als auch das letzte Element ist, also die Liste nur aus einem Element besteht. In diesem Fall ist nach dem Löschen des letzten Elementes eine leere Liste übrig, das heißt sowohl Kopf als auch Fuß zeigen auf kein Element.

### Sortieren

Als letzte Operation soll nun – zumindest in Ansätzen – das Sortieren einer Liste erklärt werden. Vorher sollte man sich noch über den Sinn und die Vorteile einer sortierten Liste Gedanken machen. Hierzu wird erneut der Posteingang eines E-Mail-Accounts betrachtet. Bei den meisten Mailanbietern oder Mailprogrammen können die angekommenen Mails nach verschiedenen Kriterien sortiert werden, zum Beispiel nach dem Absender, der Betreffzeile oder der Ankunftszeit, was normalerweise als Standard eingesetzt wird. Die Sortierung bringt allerdings nicht nur für die Person, die vor dem Computer sitzt und die Mails betrachtet, Vorteile, sondern auch für den Computer, wenn dieser die Mails im Posteingang weiterverarbeiten möchte.

**Aufgabe 2.8**

Überlege dir, welche Vorteile eine sortierte Liste bei der Durchführung der oben beschriebenen Operationen mit sich bringt! Beachte besonders das Suchen eines Listenelementes.

Als kleiner Nachteil sollte erwähnt werden, dass das Erstellen einer sortierten Liste etwas aufwendiger ist, als die Elemente beliebig anzuordnen, da man zuerst die passende Stelle in der Liste suchen muss, bevor man ein Element einfügt oder löscht. Gerade bei besonders umfangreichen Listen ist dieser Nachteil nicht gravierend, da die Vorteile der sortierten Liste diesen Aufwand rechtfertigen. Leider ist es nicht einfach, eine unsortierte in eine sortierte Liste umzuwandeln. Man muss hier schrittweise vorgehen und immer paarweise zwei Elemente vertauschen. Dies kann durch Löschen und Einfügen der Elemente erreicht werden. Vielleicht hast oder wirst du dich noch im weiteren Verlauf des Informatikunterrichts mit dem Problem des Sortierens auseinandersetzen. Ein konkreter Algorithmus, der eine Liste komplett sortiert, soll hier nicht aufgeführt werden, da dies kein spezielles Problem der Liste ist und den Rahmen des Leitprogrammes sprengen würde.



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

## 2.3 LÖSUNGEN DER AUFGABEN

Alle Aufgaben bearbeitet? Dann kannst du deine Lehrkraft oder das Betreuer*innenteam nach den Musterlösungen fragen, um deine Antworten mit diesen abzugleichen. Solltest du dann noch Fragen haben, dann bespreche diese bitte mit deiner Lehrkraft oder einem Mitglied des Betreuerteams.

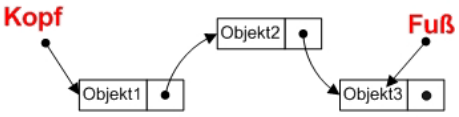


Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

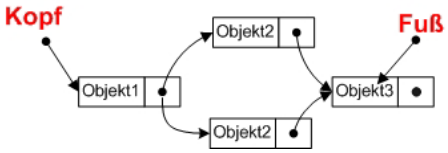
**Lernkontrolle 2.1**

Welche der hier dargestellten Listen sind per Definition KEINE Listen? Schaue dir die Listen genau an!

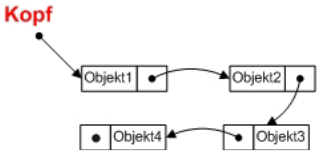
a)



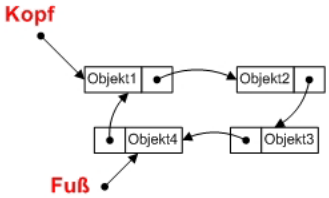

b)




c)




d)




**Lernkontrolle 2.2**

Vervollständige folgende Aussagen über eine Liste:

Eine Liste besteht aus Listenelementen, die durch einen _____ miteinander verkettet sind.

Damit bekannt ist, wo die Liste anfängt und endet, existieren zudem ein Zeiger auf das erste Element, der _____ genannt wird, und ein Zeiger auf

das letzte Element, der _____ genannt wird.

**Lernkontrolle 2.3**

Welche der aufgeführten Schritte müssen durchgeführt werden, um ein Element in eine Liste einzufügen? Schreibe die Reihenfolge in die Kästchen vor den Anweisungen. Schreibe ein „X“ wenn diese Anweisung nicht korrekt ist.

☐ Zeiger des Vorgängerelementes auf das Neue zeigen lassen.



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

- [ ] Zeiger des neuen Elementes auf seinen Vorgänger zeigen lassen.
- [ ] Zeiger des neuen Elementes auf seinen Nachfolger zeigen lassen.
- [ ] Suchen der Stelle, wo das Element eingefügt werden soll.
- [ ] Zeiger des Vorgängerelementes immer auf das letzte Element zeigen lassen.

## 2.5 LÖSUNGEN DER LERNFORTSCHRITTSKONTROLLE

Alle Aufgaben bearbeitet? Dann kannst du deine Lehrkraft oder das Betreuer*innenteam nach den Musterlösungen fragen, um deine Antworten mit diesen abzugleichen. Solltest du dann noch Fragen haben, dann bespreche diese bitte mit deiner Lehrkraft oder einem Mitglied des Betreuer*innen-teams.



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

## 2.6 ADDITUM: DIE DOPPELT VERKETTE LISTE

In diesem Additum soll die Liste, die du bereits kennengelernt hast, erweitert werden.

**Zusatzaufgabe 2.1**

(a) Beschreibe formal, wie du vorgehen würdest, wenn du für einen Listeneintrag in der Listennotation den jeweiligen Vorgänger suchen müsstest.
(b) Überlege dir, wie man die Liste erweitern könnte, damit man das Vorgängerelement sofort (d. h. ohne Suchen) bestimmen kann.

**Definition: doppelt verkettete Liste**

Eine doppelt verkettete Liste ist eine einfach verkettete Liste mit folgender Erweiterung: Jedes Element der Liste hat nicht nur einen Zeiger auf das nachfolgende, sondern auch einen zusätzlichen zweiten Zeiger auf das Vorgängerelement. Auch hier ist wieder ein Sonderfall zu beachten: Das erste Element hat keinen Vorgänger und damit folglich auch keinen Zeiger auf ein Vorgängerelement.

Diese Erweiterung bringt den Vorteil, dass zum einen Vorgängerelemente schnell und einfach gefunden werden können und zum anderen in beide Richtungen über die Liste gelaufen werden kann. Es gibt allerdings einen kleinen Nachteil an diesem neuen Komfort: Der zusätzliche Zeiger belegt Speicherplatz, so dass insgesamt mehr Speicherplatz für die Liste benötigt wird. Um nun die Veränderung der Operationen bezüglich der einfach verketteten Liste zu überlegen, soll auch hier erstmal eine grafische Notation entwickelt werden, die die interne Arbeit des Computers aufzeigt.

**Zusatzaufgabe 2.2**

Überlege dir eine grafische Schreibweise für die doppelt verkettete Liste analog zu der Notation, die dir schon von der einfach verketteten Liste bekannt ist.

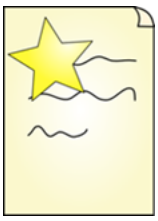
**Zusatzaufgabe 2.3**

Erinnere dich an die Einkaufsliste deiner Großeltern aus Aufgabe 2.1. Stelle diese Einkaufsliste als doppelt verkettete Liste dar. Nutze dazu die Notation aus der Lösung zu Aufgabe 2.1.

Da es zu ausführlich wäre, alle Operationen jetzt noch einmal zu erklären, wird sich auf das Einfügen eines Elementes beschränkt. Es stellt sich die Frage, was nun anders gemacht werden muss als bei der einfach verketteten Liste. Und die Antwort ist recht einfach: Nichts, es müssen nur zusätzlich die Zeiger zum Vorgängerelement bearbeitet werden.



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).



**Zusatzaufgabe 2.4**

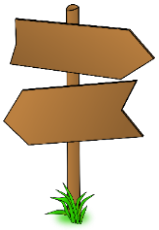
Überlege dir, wie die Zeiger auf Vorgängerelemente gesetzt werden müssen, wenn ein neues Element eingefügt wird!

## 2.7 LÖSUNGEN DER ADDITUM-AUFGABE

Eine Musterlösung zu allen Additum-Aufgaben gibt es bei der Lehrkraft oder dem Betreuer*innen-team.



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).



**Übersicht**

Nachdem du die Liste als Datenstruktur und die wichtigen Operationen auf der Liste kennengelernt hast, werden nun zwei weitere interessante Datenstrukturen vorgestellt:

- Queue
- Stack

Warum sind sie interessant? Lies weiter, um es herauszufinden!

**Lernziele**

Nach diesem Kapitel weißt du,

- was die Unterschiede von Queue und Stack zur „normalen“ Liste sind.
- welche Gemeinsamkeiten sie besitzen.
- warum man Queue und Stack braucht.
- was die Besonderheiten dieser „Arten“ von Listen sind.

## 3.1 QUEUE

Denke noch einmal an das Beispiel zum E-Mail-Programm aus dem ersten Kapitel. Was passiert, wenn du zwei E-Mails hintereinander an deinen Freund versendest? Welche kommt zuerst an? Du würdest bestimmt erwarten, dass die erste von dir verschickte E-Mail auch zuerst ankommt. Dies wird dann in der Regel auch der Fall sein. Dein E-Mail-Programm schickt die E-Mail zu einem E-Mail-Verteiler (Mail-Server), welcher dann dafür sorgt, dass die Aufträge, die zuerst ankommen, auch zuerst bearbeitet (verschickt) werden.

Dieses Arbeitsprinzip wird „First come, first serve“ (kurz FCFS, dt.: „Wer zuerst kommt, wird zuerst bedient“) genannt.

FCFS findet nicht nur in der Welt der E-Mails Einsatz, sondern auch in vielen weiteren Gebieten; zum Beispiel in der Spieleabteilung in dem Laden um die Ecke an der Kasse: Wenn du dort ein Spiel kaufst, willst du mit Sicherheit gerecht bedient werden. Das geht aber nur, wenn die Kundinnen und Kunden, die vor dir an der Kasse stehen, auch vor dir und entsprechend die Kundinnen und Kunden, welche nach dir kommen, nach dir bedient werden.

**Definition: Queue**

Queue (dt.: Warteschlange) ist eine Datenstruktur, welche eine beliebige Menge von Objekten aufnimmt und diese in der Reihenfolge ihres Einfügens wieder zurückgibt. Dazu stehen die Operationen

- enqueue(Objekt1) zum Hinzufügen eines Objekts und
- dequeue() zum Zurückholen und Entfernen des ersten Objektes zur Verfügung.

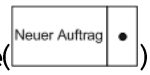
Dabei wird nach dem „First In, First Out“-Prinzip (dt.: zuerst hinein, zuerst hinaus, kurz FIFO) gearbeitet.

FIFO und FCFS besagen also das Gleiche. FIFO ist bequemer, wenn man über die Queue als Datenstruktur redet, weil die Objekte in die Warteschlange kommen und aus der Warteschlange später entnommen werden. Der Unterschied zu einer normalen Liste wird jetzt offensichtlich: Queue ist nichts anderes als eine Liste mit eingeschränkter (spezieller) Funktionalität.



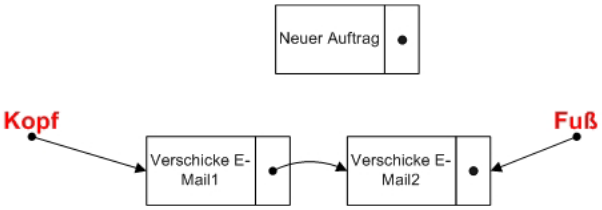
Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

Hier noch eine grafische Darstellung der Operationen:

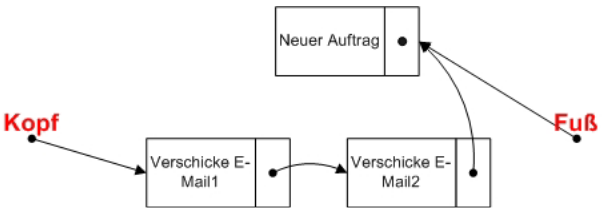
**enqueue**( 

)

„Neuer Auftrag“ wird jetzt in die Warteschlange hinten eingereiht.



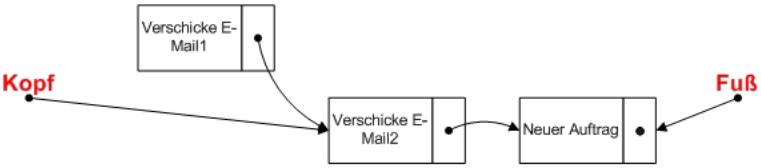

Dazu wird der Fußzeiger angepasst und zeigt auf das neue Element. Das bis dahin letzte Element der Queue zeigt dann auch auf das neue Element.



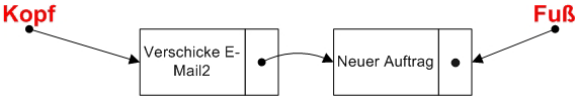

Jetzt ist die Operation zu Ende.

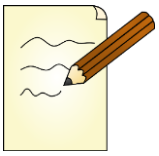
**dequeue**()

Bei dieser Operation wird das erste Element der Queue ausgelesen. Der Kopfzeiger wird dabei auf den Nachfolger des ersten Elements gesetzt.




Anschließend wird das ehemalige erste Element gelöscht.



**Aufgabe 3.1**

(a) Überlege, welchen Listenfunktionen enqueue und dequeue entsprechen?
(b) Welche Listenfunktionen werden bei Queue nicht gebraucht?

Es wird enqueue(Robert) geschrieben, wenn Robert in die Warteschlage gestellt werden soll und dequeue(), wenn die Person aus der Warteschlange entnommen werden soll, die an der Reihe ist.



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

**Aufgabe 3.2**

Zu Beginn ist die Warteschlange leer.

(a) Führe folgende Operationen aus, und notiere nach jeder Operation den Inhalt der Warteschlange: enqueue(Ulrike), enqueue(Thomas), dequeue(), enqueue(Thorsten), dequeue(), enqueue(Sarah).
(b) Zeichne nach jeder Operation die Objekte in der Warteschlange als Listenelemente.

Bei der Bearbeitung der Aufgabe hast du dir sicherlich die Frage gestellt, was passieren würde, wenn ein dequeue() ausgeführt wird, wenn die Warteschlange leer ist. Dies ist eine berechtigte Frage, denn es müssen bei einer Queue zwei Sonderfälle berücksichtigt werden:

(a) Die Queue ist voll, und es wird versucht, Objekte hinzuzufügen.
(b) Die Queue ist leer, und es wird versucht, Objekte zu entnehmen.

Was gemacht wird, wenn einer dieser beiden Fälle eintritt, hängt davon ab, wo die Queue eingesetzt wird. Queues werden sehr oft in Rechensystemen benutzt z. B. im Beschreiben von Festplatten oder bei der Dateneingabe mit Tastatur oder Maus. Man sagt zu der Queue in diesen Fällen auch „Puffer“ (engl.: buffer). Es kann in solchen Fällen unter Umständen zum Datenverlust kommen.

**Aufgabe 3.3**

Finde und nenne mindestens drei Beispiele, wo im Alltag oder im Computerleben noch Queues zu finden sind.

**Aufgabe 3.4**

In ein Geschäft mit einer Kasse gehen vier Personen zu folgenden Zeiten hinein:

- Katharina 13:55
- Jürgen 13:40
- Sabine 14:10
- Wolfgang 14:00

Katharina findet etwas, was ihr gefällt, in 15 Minuten, Jürgen stöbert 20 Minuten in den Wühlkörben, Sabine hat ihre Waren nach 5 Minuten, Wolfgang ist in 15 Minuten fertig.

Die Kassiererin ist gerade nicht an der Kasse, und es bildet sich eine Warteschlange. Wie sieht die Warteschlange an der Kasse aus? Zeichne die Objekte in der Warteschlange als Listenelemente, und überlege, wie mögliche Konflikte gelöst werden können.

## 3.2 STACK

Sicherlich hast du dich beim Schreiben einer E-Mail schon einmal vertan, oder zufällig eine wichtige E-Mail gelöscht. Was hättest du dich geärgert, wenn es nicht eine „Undo“-Funktion gegeben hätte. Zum Glück stellen die meisten E-Mail-Clients eine solche Funktion zur Verfügung. Wie funktioniert so eine Funktion? Vor allem bei allen modernen Textbearbeitungsprogrammen kann man mehrere Veränderungen am Dokument rückgängig machen. Natürlich wird die letzte Änderung als Erstes rückgängig gemacht. Diese Funktionalität wird meistens mithilfe eines Stacks erreicht und wird „Last



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

come, first serve“ (kurz LCFS, dt.: „Wer zuletzt kommt, wird zuerst bedient“) genannt. Die Änderungen, die du am Textdokument durchführst, werden Schritt für Schritt mithilfe eines Stacks mitprotokolliert. Willst du dann etwas rückgängig machen, so schaut das Programm auf dem Stack, was du als Letztes getan hast und stellt das Dokument vor dieser Aktion wieder her.

**Definition: Stack**

Stack (dt.: Stapelspeicher, Kellerspeicher) ist eine Datenstruktur, welche eine beliebige Menge von Objekten aufnimmt und diese in der entgegengesetzten Reihenfolge ihres Einfügens wieder zurückgibt. Dazu stehen dem Stack die Operationen

- *push(Objekt1)* (dt.: einkellern) zum Hinzufügen eines Objekts und
- *pop()* (dt.: auskellern) zum Zurückholen und Entfernen des zuletzt hinzugefügten Objektes zur Verfügung.

Dabei wird nach dem „Last In, First Out“-Prinzip (dt.: „zuletzt hinein, zuerst hinaus“, kurz LIFO) gearbeitet. LIFO und LCFS sind äquivalente Bezeichnungen für dasselbe Verfahren.

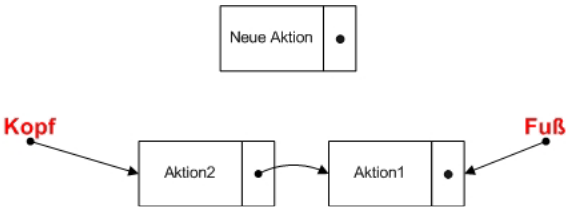
Der Stack ist allgegenwärtig. Das LIFO-Verfahren kann man sich auch gut durch Stapel von sauberen Tellern vorm Barbecue vorstellen. Wenn die Gäste kommen, so wird immer der obere Teller von dem Stapel genommen. Werden neue saubere Teller geliefert, so kommen sie ebenfalls obendrauf. Es wird auch hier offensichtlich, dass Stack nichts anderes als eine Liste mit spezieller Funktionalität ist.

Hier noch eine grafische Darstellung der Operationen:

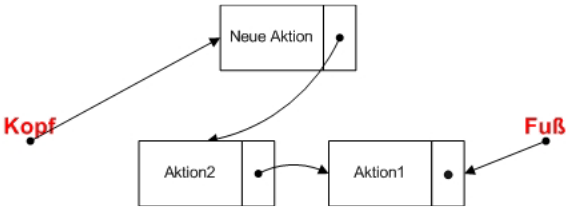
***push(***

***)***

Die neue Aktion wird vorne auf den Stack gelegt.




Dazu zeigt die neue Aktion auf das erste Element des Stacks. Der Kopfzeiger zeigt auf die neue Aktion.



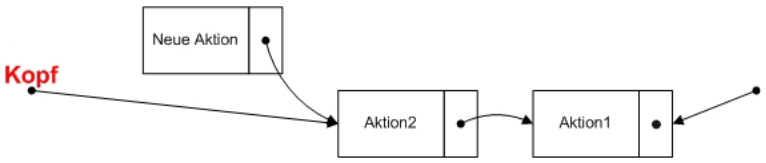

Jetzt ist die Operation zu Ende.

***pop()***

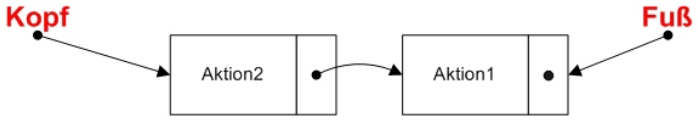
Bei dieser Operation wird das erste Element des Stacks ausgelesen. Der Kopfzeiger wird dabei auf den Nachfolger des ersten Elements gesetzt.

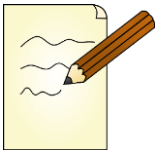


Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).




Anschließend wird das ehemalige erste Element gelöscht (vgl. Queue).



**Aufgabe 3.5**

Beantworte die folgenden Fragen:

(a) Überlege, welchen Listenfunktionen push und pop entsprechen?
(b) Welche Listenfunktionen werden bei einem Stack nicht gebraucht?
(c) Was sind die Unterschiede zur Queue?

Man schreibt *push(„invertiere die Farben“)*, wenn man etwas auf Stack legen will und *pop()*, wenn man das obere Objekt vom Stack nehmen will.

**Aufgabe 3.6**

Du bearbeitest ein Foto in einem Bildbearbeitungsprogramm. Du hast das Bild einmal im Uhrzeigersinn um 90° gedreht, dann hast du das Bild einmal heller gemacht, und anschließend hast du die Farben im Bild invertiert.

Unmerkbar für dich speichert das Bildbearbeitungsprogramm die Änderungen auf einem Stack ab. Auf einmal merkst du, dass du vor dem Invertieren der Farben das Bild nicht heller machen solltest. Wie oft müsstest du auf Rückgängig (undo) klicken, um die Helligkeit wiederherzustellen? Schreibe auf, wie der Inhalt des Stacks aussieht. Versuche diese Aufgabe mit *push()* und *pop()* aufzuschreiben und zu lösen. Stelle die Operationen und Stacks mithilfe von Listenelementen grafisch dar.

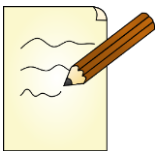
Auch beim Stack gibt es zwei Sonderfälle, die beachtet werden müssen:

(a) Der Stack ist leer, und es wird versucht, Objekte zu entnehmen.
(b) Der Stack ist voll, und es wird versucht, Objekte hinzuzufügen.

Was gemacht wird, wenn einer der Fälle eintritt, hängt wiederum von dem Anwendungsgebiet ab, in dem der Stack eingesetzt wird. Es kann in solchen Fällen unter Umständen zum Datenverlust kommen.

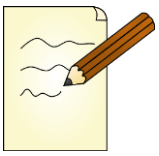


Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).



**Aufgabe 3.7**

Überlege, wo im Alltag oder Computerleben Stacks zu finden sind, und gib mindestens zwei weitere Beispiele an.



**Aufgabe 3.8**

Du erhältst folgende Zeile:

3 4 + 7 *

Führe folgende Operationen an dieser Zeile aus, wobei die Zeile von links nach rechts zu lesen ist:

- Liest du eine Ziffer, so lege sie auf dem Stack ab.
- Liest du + oder *, so hole die zwei letzten Stackeinträge und addiere bzw. multipliziere sie, lege das Ergebnis wieder auf Stack.

Schreibe in einer Tabelle die Operationen und den Inhalt des Stacks nach jeder Operation auf:

| gelesenes Zeichen | Stackbefehl | Stackinhalt |
|---|---|---|
| **3** | *push(3)* | 3 |

Wenn du alles richtig bearbeitet hast, befindet sich auf dem Stack nur das Ergebnis von (3+4)*7, also 49. So wie du gerade auf Papier gerechnet hast, rechnet der Rechner, wenn er arithmetische Ausdrücke auswerten soll. Die Art, wie die Eingabe in der Aufgabe oben geschrieben ist, heißt „Umgekehrte Polnische Notation“. Der Vorteil dieser Notation: Du brauchst keine Klammern zu schreiben, und du musst die Regel „Punkt vor Strich“ nicht beachten. Es wird immer einfach von links nach rechts ausgewertet.

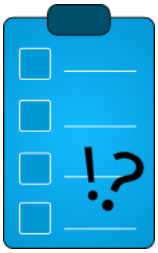
## 3.3 LÖSUNGEN DER AUFGABEN

Alle Aufgaben bearbeitet? Dann kannst du deine Lehrkraft oder das Betreuer*innenteam nach den Musterlösungen fragen, um deine Antworten mit diesen abzugleichen. Solltest du dann noch Fragen haben, dann bespreche diese bitte mit deiner Lehrkraft oder einem Mitglied des Betreuer*innenteams.



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

## 3.4 LERNFORTSCHRITTSKONTROLLE

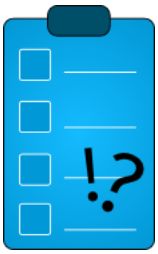


**Lernkontrolle 3.1**

In der Warteschlange stehen in der folgenden Reihenfolge: Denis, Meike und Sebastian.

(a) Führe folgende Operationen aus, und notiere nach jeder Operation den Inhalt der Warteschlange: *dequeue()*, *enqueue(Thomas)*, *dequeue()*, *enqueue(Thorsten)*, *enqueue(Ulrike)*, *dequeue()*, *dequeue()*, *dequeue()*, *enqueue(Sarah)*.

(b) Zeichne nach jeder Operation die Objekte in der Warteschlange als Listenelemente.



**Lernkontrolle 3.2**

Du willst mit deinen Eltern zelten gehen. Dafür packst du in den Kofferraum ihres Autos folgende Sachen in der folgenden Reihenfolge ein:

- Zelt
- Schlafsack
- Isomatte
- Holzkohle
- Lebensmittel

(a) Mithilfe welcher Datenstruktur würdest du den Kofferraum des Autos formal darstellen?

(b) Zeichne den geladenen Kofferraum als diese Datenstruktur in unserer Notation auf.

(c) In welcher Reihenfolge wirst du die Sachen wieder entladen?

## 3.5 LÖSUNGEN DER LERNFORTSCHRITTSKONTROLLE

Die Musterlösungen findest du – wie immer – bei deiner Lehrkraft oder dem Betreuer*innenteam.



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

In diesem Additum wirst du weitere Einsatzmöglichkeiten von Queue und Stack kennenlernen. Zum Warmwerden erstmal eine kleine Aufgabe:

**Zusatzaufgabe 3.1**

Es wird eine Queue mit Druckaufträgen betrachtet. Diese Warteschlange kann in diesem Fall nur zehn Objekte aufnehmen. Welches Problem tritt auf, wenn alle Plätze belegt sind und ein neuer Druckauftrag ankommt? Wie könnte man dieses Problem lösen?

Warteschlangen werden häufig als Ringpuffer implementiert. Die Besonderheit des Ringpuffers ist, dass, wenn der Puffer voll ist, die ältesten Inhalte wieder überschrieben werden. Eine korrekt implementierte Warteschlange sollte daher in diesem Fall entweder einen Pufferüberlauf signalisieren oder zusätzlichen Speicherplatz bereitstellen, es sei denn, der Datenverlust kann akzeptiert werden. Man stellt den Ringpuffer oft als Ring dar, daher der Name.

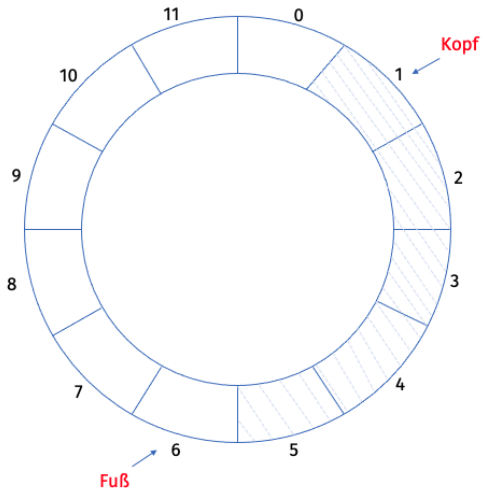



Abbildung 2: Ringpuffer

**Zusatzaufgabe 3.2**

Ist es sinnvoll, die Warteschlange für Druckaufträge als Ringpuffer zu implementieren? Begründe deine Antwort.

Die Blackbox im Flugzeug ist in der Regel ein Ringpuffer, sodass bei einem Absturz auch nur die letzten Minuten der aufgezeichneten Messwerte vorhanden sind.



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

**Zusatzaufgabe 3.3**

Überlege dir, wo man Ringpuffer noch wiederfindet, bzw. wo man diese einsetzt.

**Zusatzaufgabe 3.4**

Gegeben ist folgender Ausdruck in der *Umgekehrten Polnischen Notation* (vgl. Aufgabe 3.8):

2 28 + 40 * 800 - 10 / 2 +

(a) Werte diesen Ausdruck mithilfe eines Stacks aus, und notiere dein Vorgehen in einer Tabelle (vgl. Aufgabe 3.8).
(b) Wandle diesen Ausdruck in die „normale“ Notation um.

Beachte:

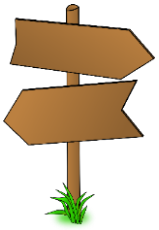
- Wenn du - liest, so wird die Zahl, die zuerst vom Stack geholt wird, von der zweiten subtrahiert.
- Falls du / liest, so teilt die Zahl, die zuerst vom Stack geholt wird, die zweite Zahl.

## 3.7 LÖSUNGEN DER ADDITUM-AUFGABEN

Frage nach den Musterlösungen, um deine Antworten mit diesen abzugleichen. Solltest du dann noch Fragen haben, dann bespreche diese bitte mit deiner Lehrkraft oder einem Mitglied des Betreuer*innenteams.

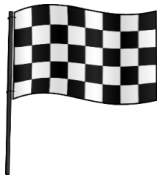


Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).



**Übersicht**

In diesem Kapitel wirst du beginnen, mit dem Computer zu arbeiten. Du wirst dich mit der Implementierung von Listen beschäftigen. Es soll also nun die Theorie aus Kapitel 2 in Java „übersetzt“ werden.
Damit die Implementierung nicht zu abstrakt wird, wurde als konkretes Beispiel eine E-Mail-Liste gewählt. Dieses Beispiel kannst du später mit nur wenig Aufwand auf eine Liste mit anderem Inhalt übertragen.



**Lernziel**

Nach der Bearbeitung dieses Kapitels bist du in der Lage, allgemeine Listen zu implementieren. Du berücksichtigst dabei die Sonderfälle, die bei der Implementierung von Listen zu beachten sind. Du wirst in der Lage sein, Quellcode in Zusammenhang mit Listen zu deuten und zu verstehen, sowie auf Korrektheit zu überprüfen.

Für dieses Kapitel wird der Java-Editor BlueJ verwendet.

**Aufgabe 4.1**

Starte deinen Java-Editor, und erstelle ein neues Projekt für die Bearbeitung der Aufgaben dieses Kapitels. In der Mediothek findest du weitere Hinweise dazu.
Gib dem Projekt einen aussagekräftigen Namen, sodass du das Projekt später für den Fall, dass du einmal auf eine implementierte Liste zurückgreifen willst, wiederfindest.

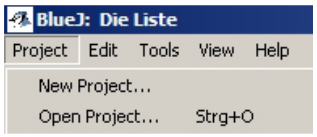



Abbildung 3: Neues Projekt

## 4.1 DIE KLASSE LISTENELEMENT – ATTRIBUTE

Bevor mit der Implementierung der eigentlichen Liste begonnen wird, müssen zuerst die einzelnen Listenelemente implementiert werden. Wie du aus dem zweiten Kapitel noch weißt, besteht ein Listenelement aus einem Datenobjekt und einem Zeiger auf das nächste Listenelement.

**Aufgabe 4.2**

Erstelle eine neue Klasse *ListenElement*.

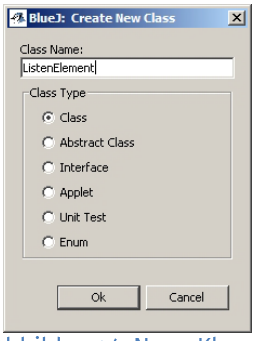



Abbildung 4: Neue Klasse



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

Wie in der Übersicht schon erwähnt wurde, wird in diesem Kapitel eine mögliche E-Mail-Liste implementiert. Damit wäre also der Dateninhalt eines Listenelements, wie du sicherlich schon erraten hast, eine E-Mail. Damit das Beispiel allerdings nicht zu sehr ausufert, soll eine E-Mail auf die folgenden drei Eckpunkte reduziert werden:

- Absender
- Betreff
- Inhalt (nur als reiner Text betrachtet, keine möglichen Dateianhänge)

**Aufgabe 4.3**

(a) Überlege dir geeignete Datentypen für die gerade aufgelisteten drei Eckpunkte einer E-Mail.
(b) Erstelle danach eine Klasse *EMail* mit diesen drei Attributen und einer Methode *drucke()*, die alle relevanten Informationen einer E-Mail auf der Konsole bzw. dem Terminal-Window ausgibt. Implementiere für die Klasse einen Konstruktor, der die folgenden Werte *String Absender; String Betreff; String Text* übergeben bekommt und anhand derer die Attribute der Klasse *EMail* gesetzt werden.
(c) Schreibe noch für die drei Attribute eine Methode *getAttributname()*, die den jeweiligen Wert des Attributs zurückgibt, wobei Attributname eines der drei Attribute bezeichnet.

Wenn du die letzte Aufgabe erfolgreich gemeistert hast, dann kannst du dich wieder der Klasse *ListenElement* zuwenden und ihre Attribute definieren. Aus dem zweiten Kapitel ist bekannt, dass ein Listenelement aus einem Objekt und einem Zeiger besteht. Als Objekt kann im konkreten Fall ein Objekt der Klasse *EMail* verwendet werden. Der Zeiger auf das nächste Element kann durch ein Objekt vom Typ *ListenElement* realisiert werden. Der Zeiger auf das nächste Element wird also durch eine Variable dargestellt, in der die Adresse des nächsten Listenelements gespeichert ist, da eine Variable (Objektvariable) in Java nicht das Objekt selbst, sondern die Adresse speichert, unter der das Objekt im Speicher abgelegt ist. Der Zeiger wird durch ein *ListenElement* namens *next* realisiert, welches auf das nächste Listenelement zeigen soll, und das Objekt durch eine *EMail* namens *mail*, das eine E-Mail darstellt.

**Aufgabe 4.4**

(a) Füge die oben beschriebenen Attribute in der Klasse *ListenElement* ein. Ebenso sollst du in dieser Aufgabe einen Konstruktor, der die beiden Attribute dem Konstruktor übergebenen Werte initialisiert, implementieren.
(b) Recherchiere im Internet über die Bedeutung der Konstanten *null* in Java., und überlege dir, wie du mittels dieser Konstanten den bereits implementierten Konstruktor dazu benutzen kannst, einen Konstruktor zu haben, der nur den Inhalt eines Listenelements setzt.

## 4.2 DIE KLASSE LISTENELEMENT – METHODEN

Nachdem im letzten Abschnitt der grobe Rahmen, also die Attribute und Konstruktoren der Klasse *ListenElement*, implementiert wurde, sollen jetzt die Methoden dieser Klasse thematisiert werden.
Als Erstes werden dafür zwei Methoden benötigt, um den Zeiger auf das andere Listenelement zu manipulieren, also verändern und auslesen, zu können.
Diese werden mit *setNext(: : :)* bzw. *getNext(: : :)* bezeichnet.



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

**Aufgabe 4.5**

Überlege dir...
(a) welche der beiden Methoden ein Argument braucht und welches dieses ist.
(b) welchen Rückgabewert die andere Methode hat.
Damit bist du nun in der Lage, dies zu implementieren.

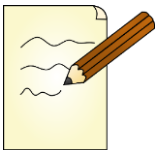
Um weiter sinnvoll auf den eigentlichen Inhalt eines jeden Listenelementes zugreifen zu können, wird noch die Methode *getContent(: : :) gebraucht*, die die in dem Listenelement gespeicherte E-Mail zurückliefert.

**Aufgabe 4.6**

Überlege dir auch hier wieder zuerst,...
(a) ob die Methode einen Parameter braucht.
(b) welchen Rückgabetyp die Methode hat.
Mit diesen Informationen kannst du nun auch diese Methode implementieren.

## 4.3 DIE KLASSE LISTE - ATTRIBUTE

Nachdem du nun eine ganze Reihe an Vorarbeiten erfolgreich abgeschlossen hast, beschäftigst du dich mit der Klasse Liste.



**Aufgabe 4.7**

Aus dem zweiten Kapitel weißt du, dass eine Liste aus verketteten Listenelementen und aus einem Zeiger auf das erste und das letzte Element besteht. Überlege dir geeignete Attribute für diese Eigenschaften sowie einen Konstruktor, der eine leere Liste anlegt (Hinweis: mittels der Konstanten *null*).

Für die Implementierung wird ein zusätzlicher Zeiger, welcher mit *currentElement* bezeichnet wird, eingefügt. Dieser Zeiger kann vorwärts über die Liste bewegt werden, also mit einem bestimmten Methodenaufruf ein Listenelement weitersetzen. Ferner kann auf dieses Listenelement zugegriffen werden – doch dazu gibt es im nächsten Abschnitt weitere Informationen.

**Aufgabe 4.8**

(a) Erstelle eine neue Klasse *Liste*.
(b) Implementiere deine Überlegungen (d. h. Konstruktor und Attribute) der letzten Aufgabe.
(c) Erstelle ein weiteres Attribut, das später die Implementierung des zusätzlichen Zeigers mit der Bezeichnung *currentElement* ermöglicht. Dieses soll vom Konstruktor auf *null* gesetzt werden. Vor der Implementierung solltest du dir überlegen, welchen Datentyp der Zeiger haben muss.

## 4.4 DIE KLASSE LISTE - METHODEN

Nachdem für die Klasse *Liste* der grobe Rahmen definiert wurde, wirst du dich jetzt mit den Methoden dieser Klasse auseinandersetzen. Jetzt wird es richtig spannend. In diesem Abschnitt wirst du die Liste „zum Leben erwecken“, in dem Sinne, dass du Methoden implementierst, durch die mit der Liste gearbeitet werden kann.
Als Erstes soll eine Methode geschrieben werden, die feststellt, ob die Liste leer ist. Wann ist das der Fall? Genau dann, wenn es keinen Kopf gibt. Programmiertechnisch gibt es dann keinen Kopf, wenn das entsprechende Attribut auf eine leere Referenz, also *null* zeigt.



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

**Aufgabe 4.9**

(a) Schreibe eine Methode *isEmpty()*, die einen booleschen Wert als Rückgabe hat, und überprüft, ob die Liste leer ist. Wenn die Liste leer ist, soll *true* zurückgegeben werden.
(b) In der Vorbemerkung wurde für eine entsprechende Abfrage auf das Kopfelement zurückgegriffen. Gibt es einen Grund, der dagegensprechen würde, eine analoge Abfrage mit dem Fuß zu implementieren?

Als Nächstes wendest du dich dem *currentElement* zu. Wie bereits erwähnt, zeigt es immer auf ein Element der Liste, und es kann immer mittels der Methode *next()* auf das Nachfolgeelement des jeweiligen Elementes gesetzt werden. Dabei ist ein Sonderfall zu beachten: Wenn das *currentElement* auf das letzte Element zeigt und *next()* aufgerufen wird, soll es auf das erste Element der Liste gesetzt werden. Um auch wieder sofort zum Anfang der Liste zu gelangen, soll eine weitere Methode *toFirst()* geschrieben werden, die das *currentElement* auf das erste Element setzt. Wenn die Liste leer ist, soll nichts verändert werden.

**Aufgabe 4.10**

(a) Schreibe eine Methode *toFirst()*, die die geforderten Eigenschaften hat.
(b) Überlege dir genau die Bedingungen für die Methode *next()*, bevor du diese implementierst. Danach kannst du sie implementieren.

**Aufgabe 4.11**

Eine Freundin zeigt dir das folgenden Stück Quelltext:

```
1  // Kommentar 1
2      public EMail getItem(){
3      if (!this.isEmpty()){
4          return currentElement.getContent();
5      } else {
6          return null;
7      }
8  }
```

(a) Beschreibe ausführlich, was diese Methode macht.
(b) Schreibe aus deinen Überlegungen einen knappen Kommentar für diese Methode, den man statt „Kommentar 1“ in den Quelltext einfügen kann.
(c) Übernehme diese paar Zeilen Quelltext in deine Implementierung.

Nachdem du nun die Operationen mit dem zusätzlichen Zeiger implementiert hast, ist als Nächstes eine der wichtigsten Methoden an der Reihe: das Einfügen. Wenn du nicht mehr genau weißt, wie das Einfügen funktioniert, dann nutze jetzt die Gelegenheit und lies es noch einmal im zweiten Kapitel nach.
Hier soll eine besondere Methode zum Einfügen genutzt werden: Die Methode *insertBehind(:::)* soll, wie der Name schon vermuten lässt, das neue Element immer hinter dem Element einfügen, auf das das *currentElement* zeigt. Dabei gibt es einige Sonderfälle zu beachten. Mehr dazu erfährst du in der folgenden Aufgabe:

**Aufgabe 4.12**

Die Freundin schickt dir auch noch eine weitere Methode. Leider sind bei der Übertragung ein paar Zeilen bzw. Anweisungen verloren gegangen.



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

```java
/**
 * This method allows us, to insert E-Mails into the list.
 * The mail will be inserted behind the currentElement.
 */
public void insertBehind(EMail mail){
    ListenElement insert = /* Übertragungsfehler 1*/
    if(this.isEmpty()){
                    /* Übertragungsfehler 2*/
        this.currentElement = insert;
    } else {
        if(currentElement == this.Fuss){
            /* Übertragungsfehler 3*/
        } else {
            insert.setNext((this.currentElement).getNext());
            /* Übertragungsfehler 4*/
        }
    }
}
```

(a) Versuche, aus dem Quelltext die drei möglichen Fälle herauszufinden, die auftreten können. Beschreibe dabei jeweils, wie die Liste für einen solchen Fall aussehen muss.
(b) Beschreibe, wie man bei jedem Fall vorgehen muss, damit die Anforderungen an die Methode erfüllt sind.
(c) Schreibe die Zeilen bzw. Anweisungen neu, bei denen es bei der Übertragung Fehler gegeben hat, und übernehme die gesamte Methode in deine Implementierung.
(d) Welchen Nachteil birgt diese Implementierung hinsichtlich der Positionen, an denen man einfügen kann?

Bevor du das Kapitel erfolgreich mit der fertigen Implementierung der Liste abschließen kannst und zur Lernkontrolle kommst, muss noch die Methode *delete()* implementiert werden. Was macht die Methode *delete()*? Die Methode soll das Element löschen, auf das das *currentElement* zeigt. Wie du sicherlich noch aus dem zweiten Kapitel weißt, sind beim Löschen einige Sonderfälle zu beachten, die hier noch einmal aufgezählt werden:

(1) Die Liste ist leer, es kann also nicht gelöscht werden.
(2) Die Liste beinhaltet nur ein Element. Die Liste ist also nach dem Löschen leer.
(3) Wenn die Kopf- oder Fußelemente gelöscht werden, dann müssen die entsprechenden Zeiger richtig gesetzt werden.

Wenn keiner der Sonderfälle auftritt, kann der Zeiger *next()* des Elements, das vor dem *currentElement* steht, auf das Element nach dem *currentElement* gesetzt werden. Jetzt kann nicht mehr beim Durchlaufen der Liste auf diesen Eintrag zugegriffen werden, da er in der Verkettung nicht mehr vorkommt. Jedoch zeigt *currentElement* noch auf diesen Eintrag der Liste. Um den Eintrag nun endgültig zu löschen, muss noch der Zeiger *currentElement* ändern. Danach zeigt kein Zeiger mehr auf den Listeneintrag, und dieser wird dann folglich bei der weiteren Ausführung eines Programms aus dem Speicher gelöscht. Die Anpassung des Zeigers soll folgendermaßen funktionieren: Wenn das zu löschende Element der Fuß der Liste war, dann soll das neue *currentElement* der neue Fuß sein. Sonst wird vereinbart: Das *currentElement* soll dann auf das Nachfolgeelement des zu löschenden Elements zeigen.

**Aufgabe 4.13**

(a) Überlege dir eine Struktur von Abfragen für die Sonderfälle.
(b) Schreibe eine Methode *getElementBefore(ListenElement elem)*, die nur innerhalb der Klasse *Liste* zur Verfügung steht und den Vorgänger des Listenelementes *elem* zurückliefert. Wenn die Liste leer ist oder *elem* der Kopf der Liste ist, soll *null* zurückgeben werden.
(c) Implementiere die Methode *delete()*.

Nun ist es an der Zeit, deine Implementierung zu testen, beispielsweise, indem du ein paar Mails einfügst, alle wieder löschst und überprüfst, ob die Liste wieder leer ist. Oder du kannst ein paar E-Mails



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

an unterschiedlichen Stellen einfügen und gucken, ob die Liste intern die gewünschte Reihenfolge hat. Sicherlich fallen dir noch weitere mögliche „Tests“ ein, um deine Implementierung auf Herz und Nieren zu prüfen. In der Mediothek ist ein Beispiel für einen solchen Test abgedruckt. Wenn du dich bei den Methodenbezeichnungen an unsere Vorgaben gehalten hast, dann sollte dieser Test ohne Probleme laufen.

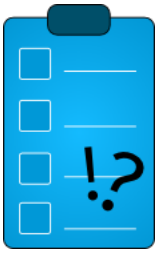
## 4.5 LÖSUNGEN DER AUFGABEN

Alle Aufgaben bearbeitet? Dann kannst du deine Lehrkraft oder die Betreuer*innen nach den Musterlösungen fragen, um deine Antworten mit diesen abzugleichen. Solltest du dann noch Fragen haben, dann bespreche diese bitte mit deiner Lehrkraft oder einem Mitglied des Betreuer*innenteams.



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

## 4.6 LERNFORTSCHRITTSKONTROLLE



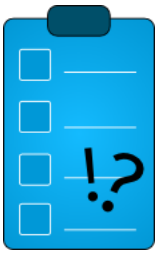
**Lernkontrolle 4.1**

Die Klasse *Liste* wurde um folgendes Attribut erweitert:

```
1        private int NumberOfElements;
```

In diesem Attribut soll gespeichert werden, wie viele Elemente in der Liste vorhanden sind.

(a) Diskutiere, inwieweit ein solches Attribut sinnvoll ist und was man ohne dieses Attribut machen müsste, um festzustellen, wie viele Elemente in der Liste vorhanden sind.
(b) Erweitere deine Implementierung so, dass das Attribut bei allen Operationen geändert wird, bei denen die Länge der Liste verändert wird.
(c) Schreibe eine Methode *getNumberOfElements()*, die die Länge der Liste als Rückgabewert hat. (Getter für das Attribut *NumberOfElements*)
(d) Wie kannst du nun auch noch ganz schnell herausfinden, ob die Liste leer ist oder nicht?



**Lernkontrolle 4.2**

Bis jetzt konnte sich das *currentElement* nur vorwärts über die Liste bewegen. Das soll nun geändert werden, indem ermöglicht wird, dass es auch rückwärtsgehen kann.

(a) Schreibe eine Methode *toLast()*, die das Attribut *currentElement* auf das letzte Element setzt.
(b) Schreibe eine Methode *previous()*, die das Attribut *currentElement* auf das Vorgängerelement des *currentElement* setzt. Welche Methode kann dir dabei behilflich sein? Welche Sonderfälle treten auf? Wie kann man darauf reagieren?
(c) Schreibe eine Methode *insertBefore(EMail mail)*, die analog zu *insertBehind(. . .)* die E-Mails vor dem *currentElement* einfügt. Überlege dir auch hier zuerst die Sonderfälle, und notiere diese. Wie muss man bei den Sonderfällen reagieren?

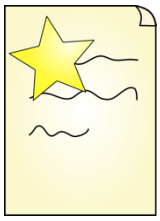
## 4.7 LÖSUNGEN DER LERNFORTSCHRITTSKONTROLLE

Die Musterlösungen findest du bei deiner Lehrkraft oder dem Betreuer*innenteam.



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

## 4.8 ADDITUM: IMPLEMENTIERUNG QUEUE UND STACK



In dem letzten Additum dieses Leitprogramms wird die Implementierung von Queue und Stack thematisiert. Zuerst sollen ganz allgemeine, aber eingeschränkte Listeneinträge implementiert werden, die für beide Datenstrukturen verwendet werden können. Die Listeneinträge sind deswegen allgemein, weil sie diesmal nicht nur E-Mails aufnehmen können, sondern ganz allgemeine Objekte. Diese Klasse wird im Folgenden *Entry* genannt. Danach kannst du dich entscheiden, ob du eine Queue oder einen Stack implementieren möchtest. Die Klassen werden entsprechend mit *Queue* bzw. *Stack* bezeichnet.

In diesem Additum wird es kaum Lehrtext geben. Es besteht fast ausschließlich aus Aufgaben, die du bearbeiten sollst. Nach den Implementierungen aus dem vierten Kapitel solltest du diese aber bewältigen können, da beide Datenstrukturen nur „abgespeckte“ Listen sind. Wenn du nicht mehr genau weißt, was Queue und Stack sind, dann nutze die Gunst der Stunde, und schaue noch einmal in die Definitionen aus dem dritten Kapitel. Die Lösungen zu den Aufgaben findest du am Ende des Additums in den jeweiligen Quelltexten zu den Klassen.

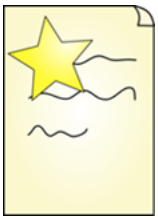
**Die Klasse *Entry***

Die Klasse *Entry* hat zwei Attribute:

- Ein Zeiger, der mit *next* bezeichnet werden soll, zeigt auf das jeweils nächste Element.
- Ein Zeiger, der mit *data* bezeichnet werden soll, der ebenfalls auf ein *Object* zeigt. Dieses *Object* ist genau das, was in diesem Eintrag von Queue oder Stack gespeichert werden soll.

Weiter hat die Klasse insgesamt vier Methoden:

Für jedes Attribut der Klasse gibt es eine *get(: : :)* und *set (: : :)* Methode, die die Werte ausliest bzw. schreibt.



**Zusatzaufgabe 4.**

Implementiere die Klasse *Entry*.

Jetzt musst du dich entscheiden, ob du einen Stack oder eine Queue implementieren möchtest. Für die Queue bearbeitest du nur den nächsten Abschnitt. Wenn du danach immer noch Zeit hast, kannst du dich auch noch an den übernächsten Abschnitt, der den Stack behandelt, wagen. Willst du allerdings lieber einen Stack implementieren, dann gehst du genau umgekehrt vor. Du bearbeitest zuerst den übernächsten Abschnitt und dann erst, wenn noch genug Zeit ist, den Abschnitt über die Queue. Stack und Queue arbeiten, wie im dritten Kapitel beschrieben, jedoch mit einer kleinen Änderung: *pop()* bzw. *dequeue()* entfernen nur das erste bzw. letzte Objekt, sie geben dieses nicht zurück. Um auf das Element zugreifen zu können, gibt es jeweils separate Methoden. Wie diese heißen, erfährst du in den folgenden beiden Abschnitten.

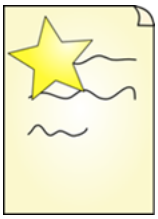
**(A) Die Klasse *Queue***

Die Klasse *Queue* hat zwei Attribute: einen Zeiger auf das erste und einen Zeiger auf das letzte Element der Queue. Weiter gibt es einen Konstruktor *Queue()*, der eine leere Queue erzeugt. Die Methode *isEmpty()* prüft, ob die Queue leer ist. Die Anfrage liefert den Wert *true*, wenn die Queue keine Elemente enthält, sonst liefert sie den Wert *false*. Die Methode *enqueue(Object pObject)* legt *pObject* als letztes Element in der Queue ab. *dequeue()* entfernt das vorderste Element der Queue. Dabei wird



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

vorausgesetzt, dass die Queue nicht leer ist. Ein Aufruf der Methode *front()* liefert das vorderste Element der Queue. Die nichtleere Queue ist danach unverändert.

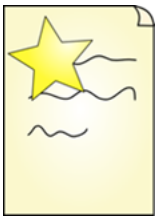


**Zusatzaufgabe 4.2A**

Implementiere die Klasse *Queue*.

**(B) Die Klasse *Stack***

Die Klasse *Stack* hat genau ein Attribut. Dies ist der Zeiger auf das oberste Element. Ferner hat die Klasse einen Konstruktor *Stack()*, der einen leeren Stack erzeugt. Die Methode *isEmpty()* beantwortet die Frage, ob der Stack leer ist. Wenn er leer ist, dann liefert sie *true* zurück, ansonsten *false*. *push(Object pObject)* legt das Objekt oben auf dem Stack ab. Entsprechend entfernt *pop()* das zuletzt hinzugefügte Element. Die Methode *top()* liefert das oberste Stackelement und verändert dabei den Stack nicht.



**Zusatzaufgabe 4.2B**

Implementiere die Klasse *Stack*.

## 4.9 LÖSUNGEN DER ADDITUM-AUFGABEN

Alle Additum-Aufgaben bearbeitet? Dann kannst du deine Lehrkraft oder das Betreuer*innenteam ein letztes Mal nach den Musterlösungen fragen, um deine Antworten mit diesen abzugleichen. Solltest du dann noch Fragen haben, dann bespreche diese bitte mit deiner Lehrkraft oder einem Mitglied des Betreuer*innenteams.



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

## QUELLENVERZEICHNIS


- Barnes, David J. und Kölling, Michael (2006): *Java lernen mit BlueJ. Eine Einführung in die objektorientierte Programmierung*, 3. Auflage, Pearson Studium.
- Hermann Ney, *Skript Datenstrukturen und Algorithmen*, http://wwwi6.informatik.rwth-aachen.de/web/Teaching/Lectures/SS06/Datenstrukturen/skript/DSAL-skript.pdf , abgerufen am 29.09.2006 [online nicht mehr verfügbar].
- Materialien zu den zentralen Abiturprüfungen im Fach Informatik, http://www.learnline.nrw.de/angebote/abitur-gost/download/materialien-ifjava060930.pdf , abgerufen am 30.09.2006 [online nicht mehr verfügbar].
- Wikipedia, *FIFO*, https://en.wikipedia.org/wiki/FIFO_(computing_and_electronics) , abgerufen am 16.02.2021.
- Wikipedia, *Queue*, https://en.wikipedia.org/wiki/Queue_(abstract_data_type) , abgerufen am 16.02.2021.
- Wikipedia, *Warteschlange*, http://de.wikipedia.org/wiki/Warteschlange , abgerufen am 16.12.2006.
- Wikipedia, *Stack*, https://en.wikipedia.org/wiki/Stack_(abstract_data_type) , abgerufen am 16.02.2021.
- Wikipedia, *Stapelspeicher*, http://de.wikipedia.org/wiki/Stapelspeicher , abgerufen am 16.12.2006.


## ABBILDUNGSVERZEICHNIS


- **Abb. 1** – Quelle: Screenshot eines E-Mail-Accounts in Outlook (https://outlook.office.com), erstellt am 17.01.2022
- **Abb. 2** – Quelle: InfoSphere
- **Abb. 3 und 4** – Quelle: Screenshot aus BlueJ (https://www.bluej.org), GNU General Public License 2.0 (https://www.gnu.org/licenses/old-licenses/gpl-2.0), erstellt am 30.09.2006.
- **Arbeitssymbole** , – Quelle: InfoSphere




Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

Wenn du in einer Aufgabe etwas recherchieren sollst, dann sind eure Schulbibliothek, Wikipedia und Google sicherlich gute erste Anlaufstellen. Prüfe aber besonders bei Internetquellen immer, ob diese zuverlässig sind. Für die Bearbeitung des vierten Kapitels brauchst du einen Computer mit einem Java-Compiler. Bei der Entwicklung des Leitprogramms haben wir die Version 1:5:0 06 benutzt. Bei anderen Versionen sollte es jedoch keinerlei Probleme geben.
Als Editor haben wir BlueJ benutzt, den man kostenlos über http://www.bluej.org/ beziehen kann. Wenn du noch nicht so fit im Umgang mit BlueJ bist, dann kannst du vorher einen Blick in das folgende Tutorial zu BlueJ werfen: http://www.bluej.org/tutorial/bluej-tutorial-deutsch.pdf. Alternativ kann aber auch jeder andere Editor gebraucht werden.
Weiter geben wir dir hier noch eine mögliche Testklasse an, mit der du deine Implementierung einmal testen kannst:

```
public class test{

    /**
     * This will test the operations on the list
     */
    public static void main(String[] args) {
    // create some E-Mails
    EMail Mail1 =
        new EMail("Tobias Reimes", "Bla", "Dies ist eine Standard bla-Mail von mir");
    EMail Mail2 =
        new EMail("Vitaly", "Bla S  lz", "Dies ist eine Standard bla-Mail von mir");
    EMail Mail3 =
        new EMail("Christian", "Bla und Bla", "Dies ist eine Standard bla-Mail von mir");
    EMail Mail4 =
        new EMail("Tobias", "Bla^3", "Dies ist eine Standard bla-Mail von mir");

    //create Liste
    Liste liste1 = new Liste();
    System.out.println("Leer?: "+ liste1.isEmpty());

    //insert the E-Mails
    liste1.insertBefore(Mail1);
    liste1.insertBefore(Mail2);
    liste1.insertBefore(Mail3);
    liste1.insertBefore(Mail4);

    //set it to first
    liste1.toFirst();

    //delete the kopf
    liste1.delete();

    //set it to the last entry
    liste1.toLast();

    //add an E-Mail at the end
    liste1.insertBehind(Mail2);

    //test it
    liste1.test();
    System.out.println("Leer? "+ liste1.isEmpty());

    }

}
```



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

## EINFÜHRUNG UND MOTIVATION UND DIE LISTE

**Definition: Liste**

Liste (von italienisch lista: Leiste, Papierstreifen) bezeichnet ein bestimmtes Verzeichnis oder generell eine Form von Verzeichnisstruktur. Eine Liste ist eine thematische Sammlung bestimmter Objekte, die nach einem bestimmten Prinzip angeordnet sind, beispielsweise alphabetisch oder chronologisch.

Eine Liste besteht aus einer beliebigen Anzahl an Listenelementen. Es existiert zudem ein Zeiger auf das erste Element der Liste, der Kopf der Liste genannt wird. Das letzte Element der Liste ist das einzige Element, das keinen Zeiger auf ein nächstes besitzt. Auch auf dieses Element gibt es einen Zeiger, der als Fuß bezeichnet wird.

Durch diese Definition wird die Liste eindeutig beschrieben, da ausgehend vom Kopf der Liste alle Elemente miteinander verkettet wurden. Dies könnte dann zum Beispiel so aussehen:

Die Zeiger werden hier als Pfeile zwischen den Elementen dargestellt. Der Pfeil zwischen dem ersten und dem zweiten Element bedeutet, dass das erste Element auf das zweite Element „zeigt“. Da das letzte Element (hier Objekt3) keinen Nachfolger besitzt, existiert auch kein Zeiger auf ein weiteres Element. An dieser Stelle ist die Liste damit zu Ende. Kopf und Fuß sind keine Elemente, sondern sind Zeiger auf das erste bzw. das letzte Element.

**Definition: dynamische Liste**

Eine Liste wird als dynamisch bezeichnet, wenn sie folgende Eigenschaften besitzt:

- Die Anzahl der Listenelemente ist nicht beschränkt oder vorher festgelegt.
- Die Listenelemente können beliebig an jeder Stelle der Liste eingefügt oder gelöscht werden.

**Definition: Datenkapselung (Additum)**

Kapselung ist ein wichtiges Konzept innerhalb der objektorientierten Programmierung. Als Datenkapselung bezeichnet man in der Programmierung das Verbergen der genauen Implementierung. Dazu gehört beispielsweise der kontrollierte Zugriff auf Methoden bzw. Attribute von Klassen.

**Definition: Listenelement**

Ein Eintrag in einer Liste wird als Listenelement bezeichnet. Ein solches Listenelement besteht aus einem Objekt (zum Beispiel einem Text) und einem Zeiger auf ein anderes Listenelement.



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

Um die recht abstrakte Definition des Listenelements noch besser zu verdeutlichen, wird folgende grafische Notation eingeführt:

Hier sieht man nun, dass die einzelnen Listenelemente immer mit genau einem nachfolgenden Element verbunden sind. Der Zeiger wird hier durch einen Pfeil zwischen den beiden Elementen dargestellt. Dies bedeutet im Sinne der obigen Definition, dass man von einem Element aus seinen Nachfolger „erreichen“ kann. Dadurch entsteht zwangsläufig die Verkettung der Elemente, da jedes Element auf genau ein weiteres zeigt. Im Unterschied zu den „normalen“ Darstellungen werden die Elemente nebeneinander und nicht mehr untereinander aufgeschrieben.

**Algorithmus: Suchen eines Listenelements**

(1) Suche das erste Element der Liste. Dieses ist leicht aufzufinden, da vom Kopf der Liste direkt ein Zeiger auf dieses Element zeigt.
(2) Schaue nun in dieses Element hinein und prüfe, ob es das gesuchte ist. Wenn ja, hast du das Element schon gefunden.
(3) Wenn dieses Element nicht das gesuchte ist: Betrachte den Nachfolger des Elementes, wenn es einen Nachfolger gibt. Dieses ist durch den vorhandenen Zeiger auf das nächste Element ebenfalls einfach zu finden.
(4) Prüfe, ob dieses Element das gesuchte ist. Wenn ja, hast du das Element gefunden. Wenn nein, gehe zurück zu Schritt 3 und durchlaufe so die Liste bis an die Stelle, wo sich das gesuchte Element befindet.
(5) Durchläufst du die Liste bis zum Ende, ohne das Gesuchte gefunden zu haben, weißt du, dass die Liste das gesuchte Element nicht enthält.

**Algorithmus: Einfügen eines Listenelements**

(1) Suche zuerst die Stelle, an der das Element eingefügt werden soll (siehe Kapitel 2). Merke dir sowohl den Vorgänger als auch den Nachfolger des einzufügenden Elementes.
(2) Setze den Zeiger des Elementes auf den Nachfolger.
(3) Setze den Zeiger des Vorgängers, der ebenfalls noch auf den Nachfolger des eingefügten Elementes zeigt, auf das neu eingefügte Element.

**Algorithmus: Löschen eines Listenelements**

(1) Suche das Element, das gelöscht werden soll (siehe Kapitel 2). Merke dir sowohl den Vorgänger als auch den Nachfolger des zu löschenden Elementes. Findest du dieses Element nicht in der Liste, brauchst du nichts weiter zu tun. Befolge ansonsten die beiden folgenden Schritte.
(2) Entferne nun das zu löschende Element aus der Liste.
(3) Setze den Zeiger des Vorgängerelementes auf das Nachfolgerelement.

## ARTEN VON LISTEN: QUEUE, STACK, DOPPELT VERKETTETE LISTE

**Definition: Queue**



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

Queue (dt.: Warteschlange) ist eine Datenstruktur, welche eine beliebige Menge von Objekten aufnimmt und diese in der Reihenfolge ihres Einfügens wieder zurückgibt. Dazu stehen Warteschlangen die folgenden Operationen zur Verfügung:

- enqueue(Objekt1) zum Hinzufügen eines Objekts und
- dequeue() zum Zurückholen und Entfernen des ersten Objektes.

Dabei wird nach dem „First In, First Out"-Prinzip (dt.: zuerst hinein, zuerst hinaus, kurz FIFO) gearbeitet.

Der Unterschied zu einer normalen Liste wird jetzt offensichtlich: Queue ist nichts anderes als eine Liste mit eingeschränkter (spezieller) Funktionalität.

**Definition: Stack**

Stack (dt.: Stapelspeicher, Kellerspeicher) ist eine Datenstruktur, welche eine beliebige Menge von Objekten aufnimmt und diese in der entgegengesetzten Reihenfolge ihres Einfügens wieder zurückgibt. Dazu stehen dem Stack die Operationen

- *push(Objekt1)* (dt.: einkellern) zum Hinzufügen eines Objekts und
- *pop()* (dt.: auskellern) zum Zurückholen und Entfernen des zuletzt hinzugefügten Objektes zur Verfügung.

Dabei wird nach dem „Last In, First Out"-Prinzip (dt.: „zuletzt hinein, zuerst hinaus", kurz LIFO) gearbeitet. LIFO und LCFS sind äquivalente Bezeichnungen für dasselbe Verfahren.

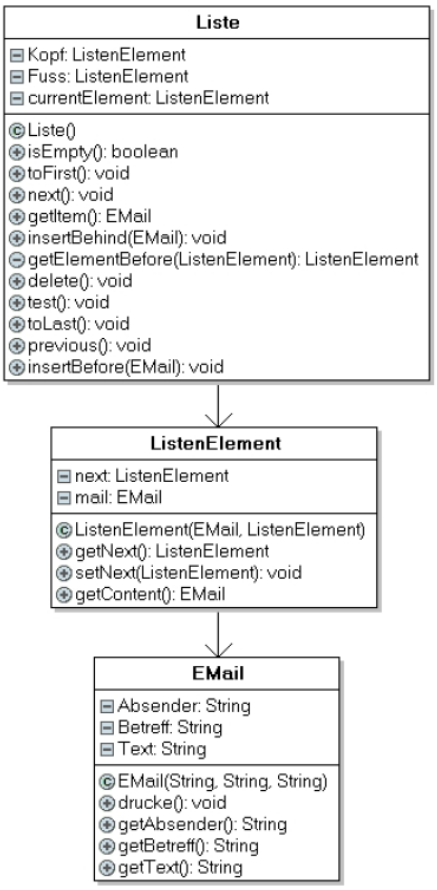
**Definition: doppelt verkettete Liste (Additum)**

Eine doppelt verkettete Liste ist eine einfach verkettete Liste mit folgender Erweiterung:
Jedes Element der Liste hat nicht nur einen Zeiger auf das nachfolgende, sondern auch einen zusätzlichen zweiten Zeiger auf das Vorgängerelement. Auch hier ist wieder ein Sonderfall zu beachten: Das erste Element hat keinen Vorgänger und folglich auch keinen Zeiger auf ein Vorgängerelement.



Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).

UML-Diagramm der erstellen Implementierung:






Dieses Werk ist lizensiert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (http://creativecommons.org/licenses/by-sa/4.0/).