

Programmierung in der gymnasialen Oberstufe
für Lehramtsstudierende im Fach Informatik

**Dynamische Datenstrukturen**

Autor:innen: Maximilian Wadenpohl, Paula Bobach

Hinweise zur Durchführung von Veranstaltungen mit den Materialien:

Das Material besteht aus Aufgabenstellungen zu BlueJ und fortgeschritteneren Programmierinhalten in der gymnasialen Oberstufe: Die generischen Klassen Stack, Queue und List. Neben den Aufgaben zur Orientierung und zum Aufdecken von Nachholbedarfen bei den Studierenden stehen im Anhang einige Materialien zur inhaltlichen Auffrischung zur Verfügung, die die Studierenden insbesondere im Selbststudium nutzen können. Auf dem Material aufbauend lassen sich beispielsweise Seminarsitzungen gestalten, die auf die Programmierhinhalte der gymnasialen Oberstufe fokussieren oder didaktischen Diskussionen Raum geben. Bei der angehängten Beispiellösung der Programmieraufgaben handelt es sich nicht um eine Musterlösung. Diese soll weitere Diskussionsanlässe schaffen und enthält daher z.B. keine Kommentare. Zusätzlich haben wir einige mögliche Tafelbilder zu den Inhalten erstellt. Diese können mit den Studenten im Seminar erarbeitet werden und als Diskussionsgrundlage dienen.

Weitere Anmerkung, die sich im Zusammenhang der Durchführung von Seminarsitzungen mit dem Material ergeben haben:

* Insbesondere Studierende ohne Erfahrungen mit Java oder dem Java-Editor benötigen teils viel Zeit zur Vorbereitung des gesamten Materials.
* Die Materialien können Anlass zur Diskussion über viele verschiedene didaktische Aspekte sein, die im Zusammenhang mit der Programmierung im Informatikunterricht stehen. Eine Fokussierung erscheint hier für die Vorbereitung einer Seminarsitzung notwendig.

Mögliche Diskussionsthemen auf Basis dieses Materials:

* Vor- und Nachteile des Arbeitens mit BlueJ. Welche Einsatzmöglichkeiten bieten sich für BlueJ besonders an?
* Ist Verkürzung von Code grundsätzlich sinnvoll? (didaktische Perspektive)
* „projektartiges“ Arbeiten im Informatikunterricht
* NRW: Vorgabe der generischen Klassen Stack, Queue, List auch für Abiturprüfungen. Welche Konsequenzen ergeben sich daraus?

Das Vorbereitungsmaterial besteht neben diesem Dokument auch aus dem Ordner „Supermarkt“, der ein kleines Programmierprojekt enthält, welches in abgewandelter Form in der Qualifikationsphase im Zusammenhang mit dynamischen Datenstrukturen entstehen könnte. Anhand dieses Beispiels sollen Sie u.a. feststellen können, ob Sie sich inhaltlich auf diese Unterrichtsinhalte vorbereitet fühlen.

**Aufgabe 1**: (BlueJ)

1. Installieren Sie BlueJ mithilfe der Instruktionen auf dieser Website: <https://www.bluej.org/>
2. Öffnen Sie die Entwicklungsumgebung und erstellen Sie zu Testzwecken ein Projekt unter Projekt 🡪 Neues Projekt.

Nun soll eine kleine Klasse implementiert werden, die die Funktionen von BlueJ veranschaulicht. Orientieren Sie sich dabei an einer Klasse, die beispielsweise von Schüler:innen der Einführungsphase nach wenigen Wochen Programmierung in der Schule umgesetzt werden könnte.

1. Erzeugen Sie nun oben links in der Entwicklungsumgebung über den Button „Neue Klasse…“ eine neue Klasse und vergeben Sie bei Aufforderung einen passenden Klassenbezeichner.

Es wurde eine Karte mit dem Klassenbezeichner in BlueJ erzeugt, die optisch an eine Klassenkarte erinnert. Über das Doppelklicken auf diese Karte gelangen Sie zum dazugehörigen Quelltext. Beim Öffnen dieses Quelltextes können Sie feststellen, dass im Quelltext der Klasse bereits einige Hilfestellungen zur Implementierung vorhanden sind.

Abbildung : Objekt erzeugen

1. Implementieren Sie nun eine Beispielklasse, die mindestens eine Anfrage (🡪 Rückgabewert) enthält. Weiterhin sollen dem Konstruktor Parameter übergeben werden müssen.

Bevor Sie Objekte erzeugen können, müssen Sie ihre Implementation kompilieren. Dazu klicken Sie in der Projektübersicht oder den einzelnen Klassen auf „Übersetzen“.

1. Erzeugen Sie nun wie in Abbildung 1 dargestellt durch Rechtsklick auf die Klassenkarte in der BlueJ-Oberfläche ein Objekt Ihrer Klasse. Dabei öffnet sich nun aufgrund der Parameter des Konstruktors ein Dialogfenster, in welchem Sie die entsprechenden Parameter festlegen sollen (im Falle eines Strings müssen hier auch die Anführungsstriche angegeben werden).
2. Das erzeugte Objekt wird nun unten in der BlueJ-Oberfläche dargestellt. Machen Sie sich mit der „Inspizieren“-Funktion vertraut, welche bei Rechtsklick auf das Objekt (siehe Abbildung 2) aufgerufen werden kann.

Abbildung : Objekte in BlueJ

1. Beschreiben Sie, wie sie die „Inspizieren“-Funktion möglicherweise gewinnbringend im Unterricht einsetzen können.
2. In Abbildung 2 sind ebenfalls die verwendbaren Methoden aufrufbar. Testen Sie auf diese Weise alle Methoden Ihrer Klasse. Im Falle der von Ihnen implementierten Anfrage sollte sich ein Fenster öffnen, welches den Rückgabewert angibt.

**Aufgabe 2**: (Supermarkt)

Im Folgenden wird ein kleines Programmierprojekt betrachtet, welches in ähnlicher Form im Zusammenhang mit dynamischen Datenstrukturen in der Qualifikationsphase entstehen könnte.

Das Projekt modelliert einen Supermarkt, in dem Produkte in einem Einkaufskorb gestapelt werden können. Von dort aus werden die Produkte auf das Kassenband einer Kasse gelegt, von welchem sie kassiert werden können, um einen Gesamtpreis für den jeweiligen Einkauf zu bestimmen. Weiterhin verfügt jede Kasse über eine alphabetisch sortierte Angebotsliste, die Produkte und deren Preise enthält. Bei Bedarf kann diese Liste auch aufsteigend nach dem Preis der Produkte ausgegeben werden.

Der Einkaufskorb wird mithilfe der Klasse Stack, das Kassenband mithilfe der Klasse Queue und die Angebotsliste mithilfe der Klasse List implementiert.

1. Öffnen Sie das BlueJ-Package im Ordner Supermarkt. Die beschriebenen Klassen sind hier bereits angelegt. Die Klassen Produkt, List, Queue und Stack sind vollständig implementiert. Machen Sie sich zunächst mithilfe der Quelltexte von List, Queue, Stack sowie den Informationen dazu im Anhang dieses Dokuments mit den verschiedenen dynamischen Datenstrukturen und ihren Methoden vertraut.
2. Implementieren Sie mithilfe der Klassendokumentationen im Anhang die fehlenden Bestandteile (hier befinden sich jeweils Markierungen im Quelltext) der Klassen Einkaufskorb, Kassenband, Angebotsliste und Kasse. Empfehlung: Beginnen Sie mit der Klasse Einkaufskorb.
3. Erstellen Sie mithilfe der BlueJ-Oberfläche wie in Aufgabe 1 Objekte und rufen Sie deren Methoden auf, um die Funktionalität ihrer Implementierung zu überprüfen. Verwenden Sie dabei insbesondere auch die „Inspizieren“-Funktion, um Queue und Stack zu untersuchen. Rufen Sie abschließend die Methode **test()** eines erzeugten Objekts der Klasse Kasse auf, um Ihre Implementation umfänglich zu testen.
4. Erläutern Sie ausführlich Ihre Implementation der Methoden **produktHinzufuegen(Produkt p)** der Klasse Angebotsliste. Erörtern Sie, welche Ausführungen Sie von Ihren Schüler:innen erwarten würden.
5. Bitte bereiten Sie ihre Implementation so vor, dass Sie diese Ihren Kommiliton:innen in der folgenden Seminarsitzung präsentieren können und bringen Sie Ihre Lösung auf einem USB-Stick mit.

Folgend werden einige Kompetenzen aus dem Kernlehrplan NRW GOSt 2014 S. 25ff. aufgeführt, die möglicherweise im Rahmen der hier vorgestellten Aufgabe geschult werden könnten.

**Kompetenzbereiche:**

Argumentieren (A)

* Die Schülerinnen und Schüler analysieren und erläutern Computerprogramme.

Implementieren (I)

* Die Schülerinnen und Schüler implementieren auf der Grundlage von Modellen oder Modellausschnitten Computerprogramme
* Die Schülerinnen und Schüler testen und korrigieren Computerprogramme systematisch

Kommunizieren und Kooperieren (K)

* Die Schülerinnen und Schüler verwenden die Fachsprache bei der Kommunikation über informatische Sachverhalte.

**Inhaltsfelder:**

Daten und ihre Strukturierung; Objekte und Klassen

* Die Schülerinnen und Schüler implementieren Klassen in einer Programmiersprache auch unter Nutzung dokumentierter Klassenbibliotheken.

Algorithmen; Analyse, Entwurf und Implementierung von Algorithmen

* Die Schülerinnen und Schüler analysieren und erläutern Algorithmen und Programme.
* Die Schülerinnen und Schüler implementieren iterative und rekursive Algorithmen auch unter Verwendung von dynamischen Datenstrukturen.
* Die Schülerinnen und Schüler testen Programme systematisch anhand von Beispielen.

Algorithmen in ausgewählten informatischen Kontexten

* Die Schülerinnen und Schüler erläutern Operationen dynamischer (linearer oder nichtlinearer) Datenstrukturen.
* Die Schülerinnen und Schüler implementieren und erläutern iterative und rekursive Such- und Sortierverfahren
* Die Schülerinnen und Schüler beurteilen die Effizienz von Algorithmen unter Berücksichtigung des Speicherbedarfs und der Zahl der Operationen

Formale Sprachen und Automaten; Syntax und Semantik einer Programmiersprache

* Die Schülerinnen und Schüler nutzen die Syntax und Semantik einer Programmiersprache bei der Implementierung und zur Analyse von Programmen.
* Die Schülerinnen und Schüler beurteilen die syntaktische Korrektheit und die Funktionalität von Programmen

**Anhang**

**Dokumentationen der verwendeten Klassen**

**Die Klasse Kasse**

Objekte der Klasse Kasse verwalten eine Kasse.

**Auszug aus der Dokumentation der Klasse Kasse**

**Kasse()**

Ein Objekt der Klasse Kasse wird initialisiert.

**void kassiereNaechstesProdukt()**

Das nächste Produkt auf dem Kassenband wird dort entfernt und dessen Preis dem insgesamt zu zahlenden Betrag hinzugefügt. Der Produktname des kassierten Produkts sowie dessen Preis werden in der Konsole ausgegeben.

**void zahlen()**

Der insgesamt zu zahlende Betrag wird in der Konsole ausgegeben. Das entsprechende Attribut wird auf 0 zurückgesetzt.

**void listen(Produkt p)**

Der Angebotsliste wird das übergebene Produkt hinzugefügt.

**void ausListeEntfernen(String produktName)**

Das Produkt mit dem übergebenen Produktnamen wird aus der Angebotsliste entfernt.

**String preisanzeige(int preis)**

Der übergebene Centbetrag wird als String der Form xx,xx EUR zurückgegeben.

**void test()**

Testet die Funktionsweise aller implementierten Klasse über entsprechende Beispiele.

**Die Klasse Produkt**

Objekte der Klasse Produkt verwalten ein Produkt.

**Auszug aus der Dokumentation der Klasse Produkt**

**Produkt(String name, int preis)**

Ein Objekt der Klasse Produkt wird mit den übergebenen Parametern initialisiert.

**String getName()**

Gibt den Namen des Produkts zurück.

**int getPreis()**

Gibt den Preis des Produkts in Cent zurück.

**Die Klasse Einkaufskorb**

Objekte der Klasse Einkaufskorb verwalten einen Einkaufskorb.

**Auszug aus der Dokumentation der Klasse Einkaufskorb**

**Einkaufskorb()**

Ein Objekt der Klasse Einkaufskorb wird initialisiert.

**void pHinzufuegen(Produkt p)**

Legt das übergebene Produkt auf den Stack.

**Produkt pHerausnehmen()**

Entfernt das oberste Produkt vom Stack und gibt dieses zurück.

**void aufBandLegen(Kassenband kband)**

Nimmt das oberste Produkt aus dem Einkaufskorb und legt es auf das übergebene Kassenband.

**Die Klasse Kassenband**

Objekte der Klasse Kassenband verwalten ein Kassenband.

**Auszug aus der Dokumentation der Klasse Kassenband**

**Kassenband()**

Ein Objekt der Klasse Kassenband wird initialisiert.

**void pHinauflegen(Produkt p)**

Legt das übergebene Produkt auf das Kassenband, indem es an das Ende der Warteschlange angefügt wird.

**Produkt pKassieren()**

Entfernt das vorderste Produkt des Kassenbands und gibt dieses zurück.

**Die Klasse Angebotsliste**

Objekte der Klasse Angebotsliste verwalten eine Angebotsliste.

**Auszug aus der Dokumentation der Klasse Angebotsliste**

**Angebotsliste()**

Ein Objekt der Klasse Angebotsliste wird initialisiert.

**void produktHinzufuegen(Produkt p)**

Fügt der Angebotsliste das übergebene Produkt hinzu. Dabei müssen Produkte innerhalb der Liste jederzeit alphabetisch sortiert sein. Jeder Produktname darf in der Liste nur einmalig vorkommen. Bei dem Versuch, ein gleichnamiges Produkt einzufügen, geschieht nichts.

**void produktEntfernen(String name)**

Entfernt das Produkt mit dem übergebenen Namen aus der Liste.

**List<Produkt> getListeNachPreis()**

Gibt eine Kopie der Angebotsliste zurück, die aufsteigend nach dem Preis der Produkte sortiert ist.

**void printList()**

Gibt die in der Angebotsliste gespeicherten Produkte alphabetisch sortiert mit Namen und Preis (in ct) in der Konsole aus.

**void printList(List<Produkt> liste)**

Gibt die Produkte der übergebenen Produktliste beginnend beim ersten Produkt der Reihe nach mit Produktnamen und dem jeweiligen Preis (in ct) in der Konsole aus.

### Nachfolgend finden sich einige Informationen der Qualitäts- und UnterstützungsAgentur – Landesinstitut für Schule, Materialien zum schulinternen Lehrplan Informatik SII (Leistungskurs) zu dynamischen Datenstrukturen. Diese wurden unter der CreativeCommons-Lizenz CC BY-SA 4.0 veröffentlicht. Zu finden sind diese auch unter <https://www.schulentwicklung.nrw.de/materialdatenbank/material/view/5651>

**Klasse Queue**

****

**Die generische Klasse Queue<ContentType>**

Objekte der generischen Klasse **Queue** (Warteschlange) verwalten beliebige Objekte vom Typ **ContentType** nach dem First-In-First-Out-Prinzip, d.h., das zuerst abgelegte Objekt wird als erstes wieder entnommen. Alle Methoden haben eine konstante Laufzeit, unabhängig von der Anzahl der verwalteten Objekte.

**Dokumentation der generischen Klasse Queue<ContentType>**

|  |  |
| --- | --- |
| **Konstruktor** | **Queue<ContentType>()** |
|  | Eine leere Schlange wird erzeugt. Objekte, die in dieser Schlange verwaltet werden, müssen vom Typ **ContentType** sein. |
| **Anfrage** | **boolean isEmpty()** |
|  | Die Anfrage liefert den Wert **true**, wenn die Schlange keine Objekteenthält, sonst liefert sie den Wert **false**. |
| **Auftrag** | **void enqueue(ContentType pContent)** |
|  | Das Objekt **pContent** wird an die Schlange angehängt. Falls **pContent** gleich null ist, bleibt die Schlange unverändert. |
| **Auftrag** | **void dequeue()** |
|  | Das erste Objekt wird aus der Schlange entfernt. Falls die Schlange leer ist, wird sie nicht verändert. |
| **Anfrage** | **ContentType front()** |
|  | Die Anfrage liefert das erste Objekt der Schlange. Die Schlange bleibtunverändert. Falls die Schlange leer ist, wird **null** zurückgegeben. |

**Klasse Stack**



Die generische Klasse **Stack<ContentType>**

Objekte der generischen Klasse Stack (Keller, Stapel) verwalten beliebige Objekte vom Typ

**ContentType** nach dem Last-In-First-Out-Prinzip, d.h., das zuletzt abgelegte Objekt wird als

erstes wieder entnommen. Alle Methoden haben eine konstante Laufzeit, unabhängig von

der Anzahl der verwalteten Objekte.

Dokumentation der generischen Klasse **Stack<ContentType>**

|  |  |
| --- | --- |
| **Konstruktor** | **Stack<ContentType>()** |
|  | Ein leerer Stapel wird erzeugt. Objekte, die in diesem Stapelverwaltet werden, müssen vom Typ **ContentType** sein. |
| **Anfrage** | **boolean isEmpty()** |
|  | Die Anfrage liefert den Wert **true**, wenn der Stapel keine Objekteenthält, sonst liefert sie den Wert **false**. |
| **Auftrag** | **void push(ContentType pContent)** |
|  | Das Objekt **pContent** wird oben auf den Stapel gelegt. Falls**pContent** gleich **null** ist, bleibt der Stapel unverändert. |
| **Auftrag** | **void pop()** |
|  | Das zuletzt eingefügte Objekt wird von dem Stapel entfernt. Falls derStapel leer ist, bleibt er unverändert. |
| **Anfrage** | **ContentType top()** |
|  | Die Anfrage liefert das oberste Stapelobjekt. Der Stapel bleibtunverändert. Falls der Stapel leer ist, wird **null** zurückgegeben. |

**Klasse List**



Die generische Klasse **List<ContentType>**

Objekte der generischen Klasse **List** verwalten beliebig viele, linear angeordnete Objekte vom Typ **ContentType**. Auf höchstens ein Listenobjekt, aktuelles Objekt genannt, kann jeweils zugegriffen werden. Wenn eine Liste leer ist, vollständig durchlaufen wurde oder das aktuelle Objekt am Ende der Liste gelöscht wurde, gibt es kein aktuelles Objekt. Das erste oder das letzte Objekt einer Liste können durch einen Auftrag zum aktuellen Objekt gemacht werden. Außerdem kann das dem aktuellen Objekt folgende Listenobjekt zum neuen aktuellen Objekt werden.

Das aktuelle Objekt kann gelesen, verändert oder gelöscht werden. Außerdem kann vor dem aktuellen Objekt ein Listenobjekt eingefügt werden.

Dokumentation der Klasse **List<ContentType>**

|  |  |
| --- | --- |
| Konstruktor | **List<ContentType>()** |
|  | Eine leere Liste wird erzeugt. |
| Anfrage | **boolean isEmpty()** |
|  | Die Anfrage liefert den Wert **true**, wenn die Liste keine Objekte enthält, sonst liefert sie den Wert **false**. |
| Anfrage | **boolean hasAccess()** |
|  | Die Anfrage liefert den Wert **true**, wenn es ein aktuelles Objekt gibt, sonst liefert sie den Wert **false**. |
| Auftrag | **void next()** |
|  | Falls die Liste nicht leer ist, es ein aktuelles Objekt gibt und dieses nicht das letzte Objekt der Liste ist, wird das dem aktuellen Objekt in der Liste folgende Objekt zum aktuellen Objekt, andernfalls gibt es nach Ausführung des Auftrags kein aktuelles Objekt, d.h. **hasAccess()** liefert den Wert false.  |
| Auftrag | **void toFirst()** |
|  | Falls die Liste nicht leer ist, wird das erste Objekt der Liste aktuelles Objekt. Ist die Liste leer, geschieht nichts. |
| Auftrag | **void toLast()** |
|  | Falls die Liste nicht leer ist, wird das letzte Objekt der Liste aktuelles Objekt. Ist die Liste leer, geschieht nichts. |
| Anfrage | **ContentType getContent()** |
|  | Falls es ein aktuelles Objekt gibt (**hasAccess()==true**), wird das aktuelle Objekt zurückgegeben. Andernfalls (**hasAccess()==false**) gibt die Anfrage den Wert **null** zurück. |
| Auftrag | **void setContent(ContentType pContent)** |
|  | Falls es ein aktuelles Objekt gibt (**hasAccess()==true**) und **pContent** ungleich **null** ist, wird das aktuelle Objekt durch **pContent** ersetzt. Sonst bleibt die Liste unverändert. |
| Auftrag | **void append(ContentType pContent)** |
|  | Ein neues Objekt **pContent** wird am Ende der Liste eingefügt. Das aktuelle Objekt bleibt unverändert. Wenn die Liste leer ist, wird das Objekt **pContent** in die Liste eingefügt und es gibt weiterhin kein aktuelles Objekt (**hasAccess()==false**). Falls **pContent** gleich **null** ist, bleibt die Liste unverändert.  |
| Auftrag | **void insert(ContentType pContent)** |
|  | Falls es ein aktuelles Objekt gibt (**hasAccess()==true**), wird ein neues Objekt **pContent** vor dem aktuellen Objekt in die Liste eingefügt. Das aktuelle Objekt bleibt unverändert. Falls die Liste leer ist und es somit kein aktuelles Objekt gibt (**hasAccess()==false**), wird **pContent** in die Liste eingefügt und es gibt weiterhin kein aktuelles Objekt. Falls es kein aktuelles Objekt gibt (**hasAccess()==false**) und die Liste nicht leer ist oder **pContent==null** ist, bleibt die Liste unverändert.  |
| Auftrag | **void concat(List<ContentType> pList)** |
|  | Die Liste **pList** wird an die Liste angehängt. Anschließend wird pList eine leere Liste. Das aktuelle Objekt bleibt unverändert. Falls **pList==null** oder eine leere Liste ist, bleibt die Liste unverändert. |
| Auftrag | **void remove()** |
|  | Falls es ein aktuelles Objekt gibt (**hasAccess() == true**), wird dasaktuelle Objekt gelöscht und das Objekt hinter dem gelöschten Objekt wird zum aktuellen Objekt. Wird das Objekt, das am Ende der Liste steht, gelöscht, gibt es kein aktuelles Objekt mehr (**hasAccess() == false**). Wenn die Liste leer ist oder es kein aktuelles Objekt gibt (**hasAccess() == false**), bleibt die Liste unverändert. |

**Quellenverzeichnis**

Ministerium für Schule und Weiterbildung des Landes Nordrhein-Westfalen: Kernlehrplan für die Sekundarstufe II Gymnasium/Gesamtschule in Nordrhein-Westfalen, Informatik, 1. Auflage 2014. <https://www.schulentwicklung.nrw.de/lehrplaene/upload/klp_SII/if/KLP_GOSt_Informatik.pdf>, zuletzt abgerufen am 06.03.2023 14:13 Uhr.

Qualitäts- und UnterstützungsAgentur – Landesinstitut für Schule: Ergänzungsmaterial zum SiLP Informatik LK UV Q1-2, <https://www.schulentwicklung.nrw.de/materialdatenbank/material/view/5651>, zuletzt abgerufen am 29.03.2023 12:17 Uhr. Veröffentlicht unter der Lizenz CC BY-SA 4.0: <https://creativecommons.org/licenses/by-sa/4.0/deed.de>