1. Die rekursive Datenstruktur Liste

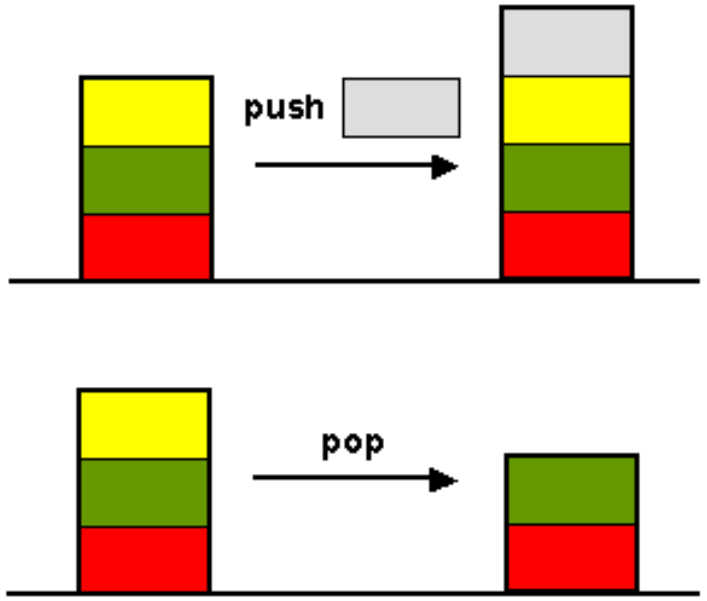
1.6 Die Datenstruktur Stapel

Ein **Stack**, auch **Stapel** oder **Keller** genannt, ist eine Datenstruktur, bei der die Elemente nur an **einem** Ende der Folge eingefügt bzw. gelöscht werden können.

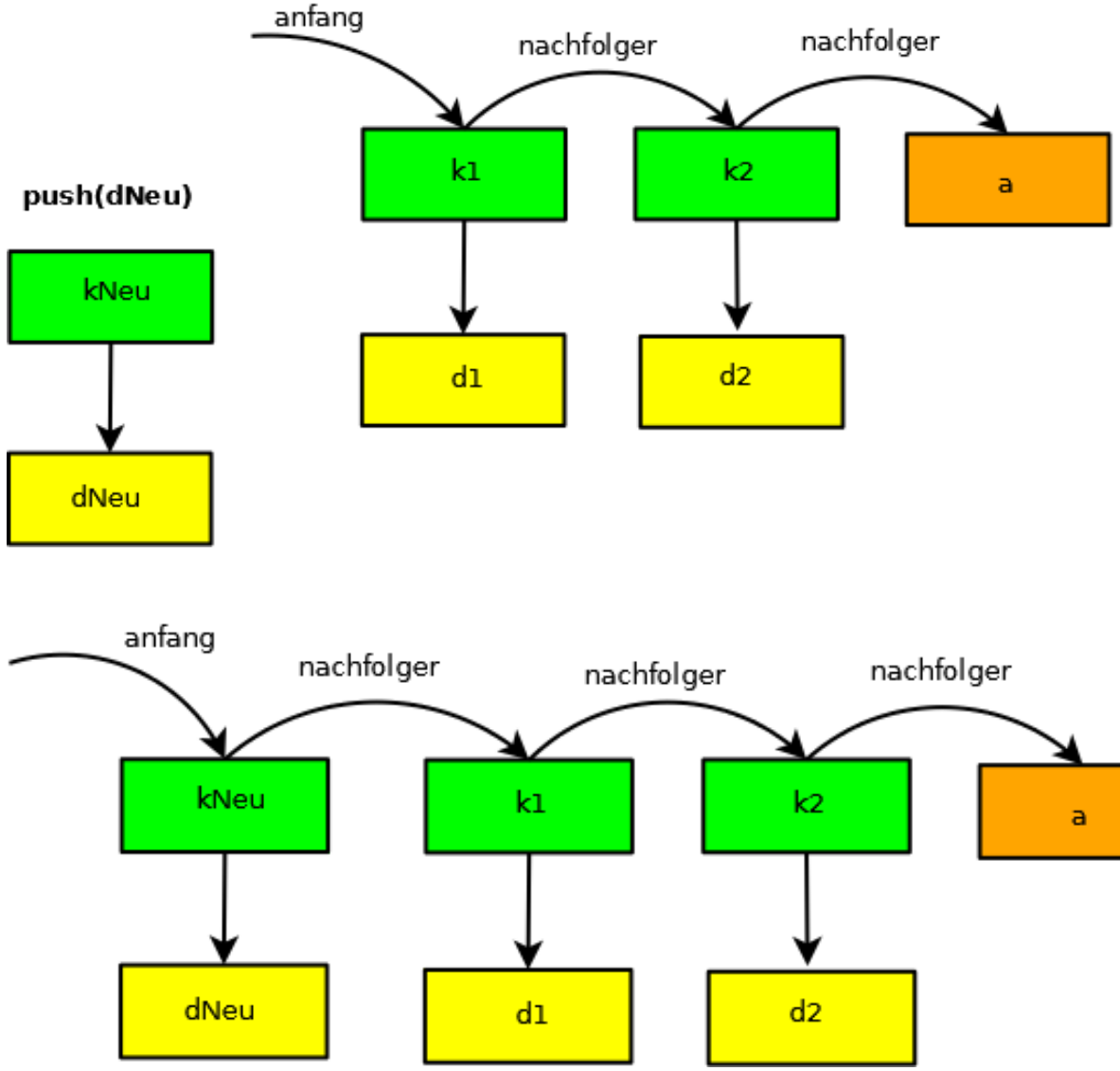
Dies bedeutet u.a., dass das Element, das als letztes Element in den Stack eingefügt wurde, als erstes Element wieder vom Stack entfernt werden muss. Man spricht deshalb oft vom **LIFO-Prinzip (Last In First Out)**.

Dementsprechend kann das als erstes in den Stack eingefügte Element erst als letztes Element wieder vom Stack entfernt werden.

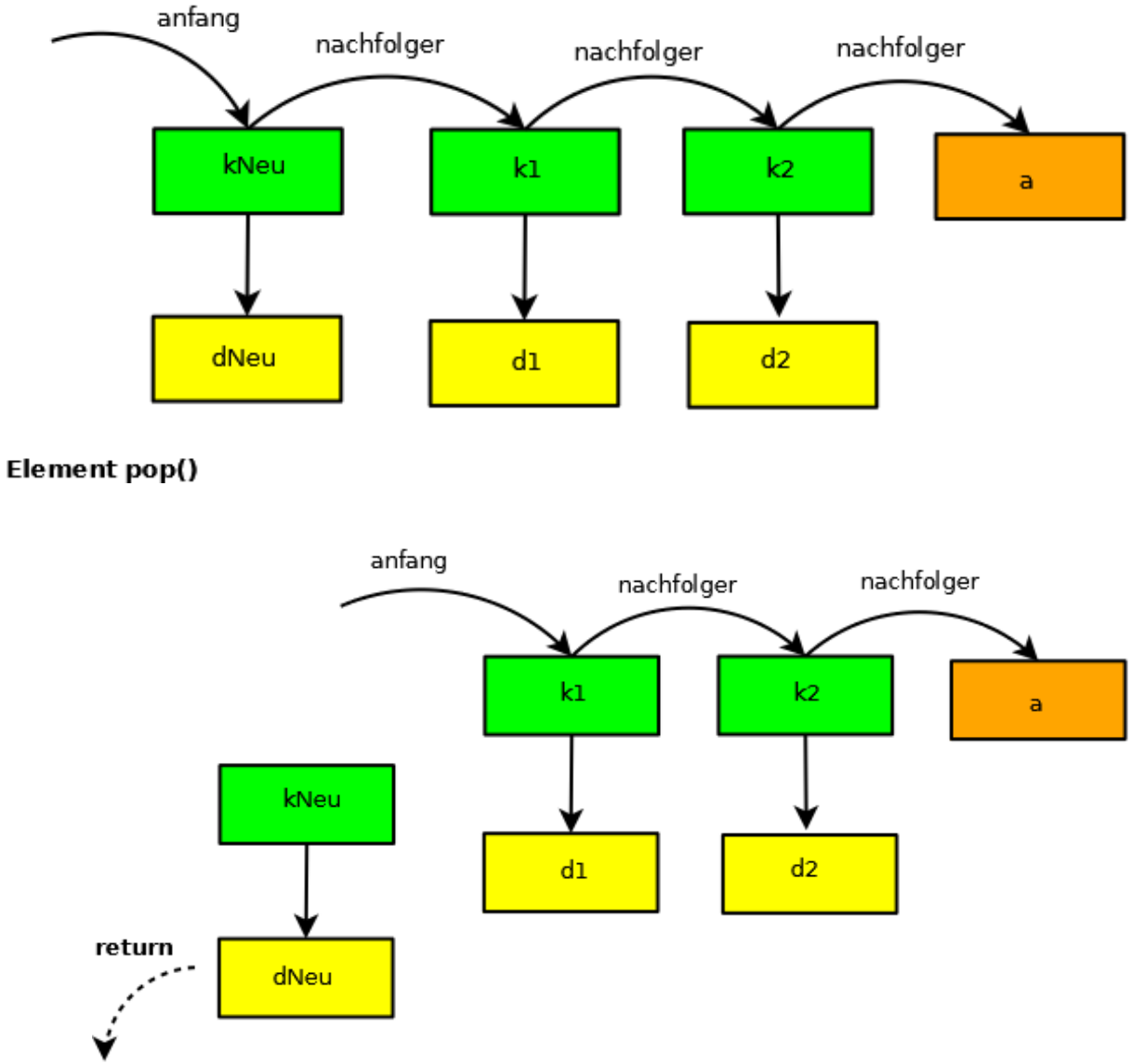
Ein Stack kann bildlich sehr gut mit einem Stapel von Kisten verglichen werden.
Die Operation zum Hinzufügen eines Elements heißt üblicherweise **push**, die Operation zum Entfernen eines Stackelements **pop**.



Methode zum Einfügen (push)



Methode zum Entfernen (pop)



In Java ist die Klasse **Stack** als Unterklasse von **Vector** implementiert.
Informiere dich in der offiziellen Java-Dokumentation (Java API) über diese Klassen.

All Methods		
Modifier and Type	Method	Description
boolean	empty()	Tests if this stack is empty.
E	peek()	Looks at the object at the top of this stack without removing it from the stack.
E	pop()	Removes the object at the top of this stack and returns that object as the value of this function.
E	push(E item)	Pushes an item onto the top of this stack.
int	search(Object o)	Returns the 1-based position where an object is on this stack.

Übung 1

Öffne das BlueJ-Projekt

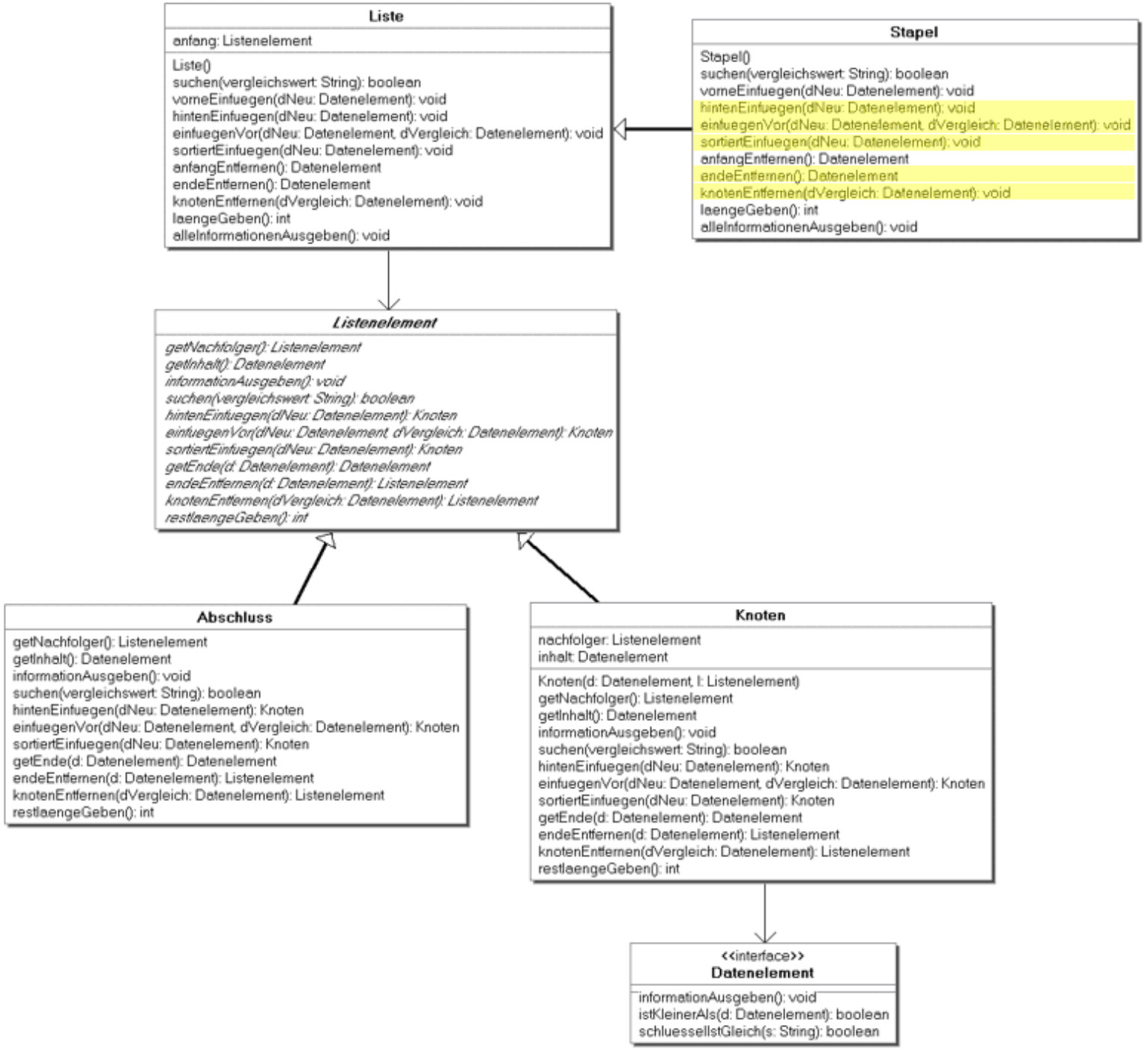
Java_Liste_Queue_Stack

In den Klassen wird mithilfe der offiziellen Javaklassen jeweils eine Liste, eine Warteschlange (Queue) und ein Stapel (Stack) implementiert.

Untersuche den Quelltext und führe jeweils die Methode **teste()** aus.

Experimentiere mit den Methoden.

Umbau der Datenstruktur Liste
(vgl. nächste Folie)



Ausgehend von unserer Datenstruktur Liste können wir einen Stapel implementieren, indem wir in Liste die nicht benötigten Methoden einfach löschen.

Eleganter schreibt man jedoch eine Klasse Stapel als Unterklasse von Liste.

Die nicht benötigten Methoden (hintenEinfuegen ( ) , einfuegenVor ( ), sortiertEinfuegen ( ) , endeEntfernen ( ), knotenEntfernen ( )) werden dadurch allerdings ebenfalls von Liste geerbt.

Deshalb muss man sie in Stapel mit leeren Methodenrümpfen überschreiben. (Im Diagramm farbig gekennzeichnet)

Übung 2

Öffne aus dem Kapitel 1.5 das BlueJ-Projekt

liste_abschluss_lsg_alle_Methoden

und ändere es gemäß der Beschreibung aus der vorherigen Folie ab.

Teste das Projekt ausführlich.

Übung 3

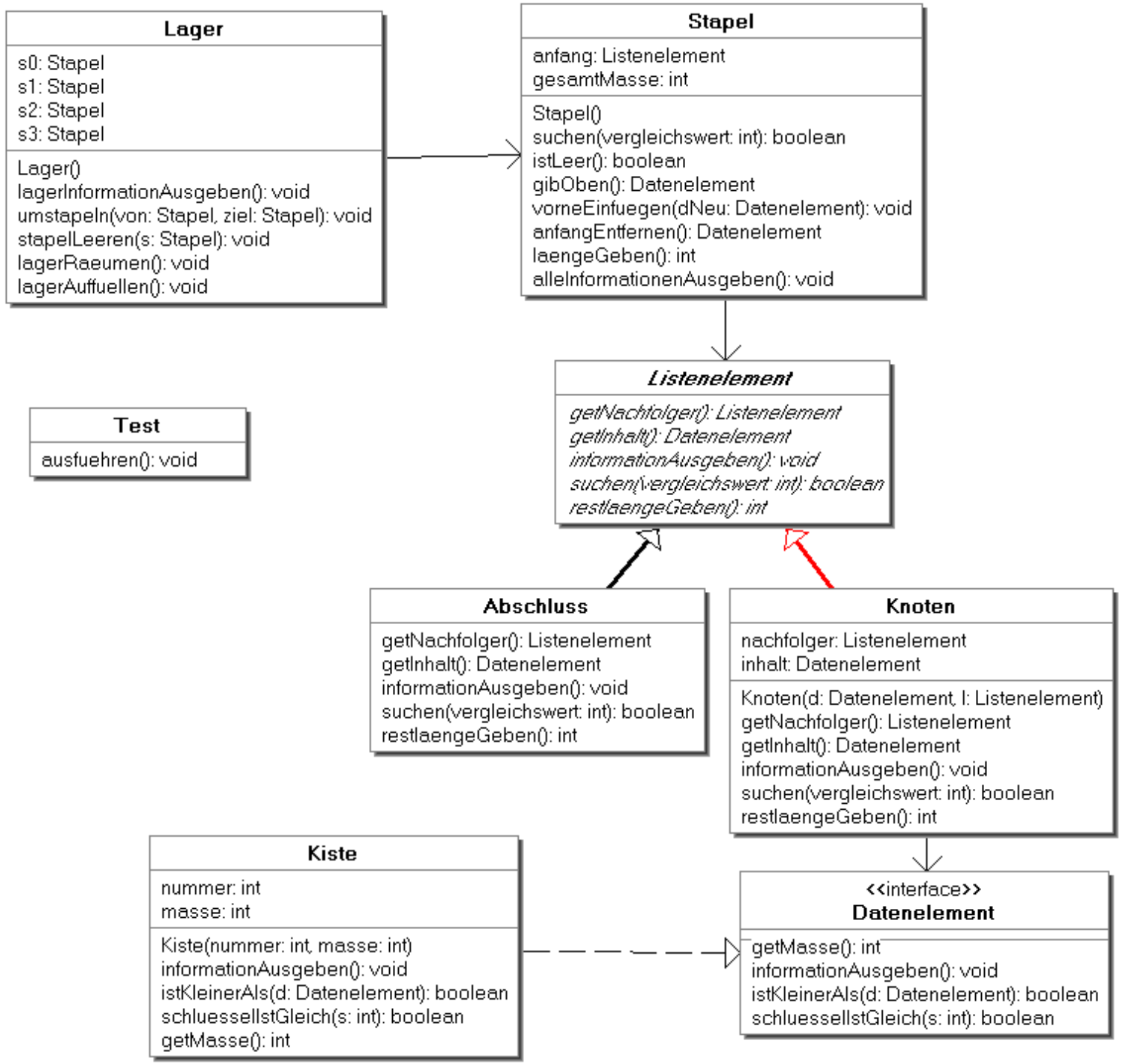
Im BlueJ-Projekt

stapel_kiste

wird die Lagerung von Kisten in einem Lager implementiert.

Klassendiagramm: nächste Folie

Übung 3


Lager
s0: Stapel
s1: Stapel
s2: Stapel
s3: Stapel
Lager()
lagerInformationAusgeben(): void
umstapeln(von: Stapel, ziel: Stapel): void
stapelLeeren(s: Stapel): void
lagerRaeumen(): void
lagerAuffuellen(): void
Stapel
anfang: Listenelement
gesamtMasse: int
Stapel()
suchen(vergleichswert: int): boolean
istLeer(): boolean
gibOben(): Datenelement
vorneEinfuegen(dNeu: Datenelement): void
anfangEntfernen(): Datenelement
laengeGeben(): int
alleInformationenAusgeben(): void
Test
ausfuehren(): void
Listenelement
getNachfolger(): Listenelement
getInhalt(): Datenelement
informationAusgeben(): void
suchen(vergleichswert: int): boolean
restlaengeGeben(): int
Abschluss
getNachfolger(): Listenelement
getInhalt(): Datenelement
informationAusgeben(): void
suchen(vergleichswert: int): boolean
restlaengeGeben(): int
Knoten
nachfolger: Listenelement
inhalt: Datenelement
Knoten(d: Datenelement, l: Listenelement)
getNachfolger(): Listenelement
getInhalt(): Datenelement
informationAusgeben(): void
suchen(vergleichswert: int): boolean
restlaengeGeben(): int
Kiste
nummer: int
masse: int
Kiste(nummer: int, masse: int)
informationAusgeben(): void
istKleinerAls(d: Datenelement): boolean
schluesselIstGleich(s: int): boolean
getMasse(): int
<<interface>>
Datenelement
getMasse(): int
informationAusgeben(): void
istKleinerAls(d: Datenelement): boolean
schluesselIstGleich(s: int): boolean

Übung 3

a)

Implementiere in der Klasse Stapel die Methoden **istLeer()** und **gibOben()**

istLeer() gibt an, ob der Stapel leer ist.

gibOben() liefert den Inhalt des Anfangs zurück ohne ihn vom Stapel zu entfernen.

Übung 3

b)

Die Klasse **Lager** verwaltet vier Stapel:

Der Stapel s0 darf beliebig aufgefüllt werden. Vom Stapel s0 werden durch einmaliges Aufrufen der Methode lagerAuffuellen( ) die Kisten auf die Stapel s1, s2 und s3 verteilt. Die Gesamtmasse der Stapel s1, s2 und s3 darf dabei maximal 100 sein.

Zuerst soll s1 so weit wie möglich aufgefüllt werden, dann s2 und s3.

Ist s3 voll und der Stapel s0 noch nicht leer, soll eine Meldung ("Lager ist voll." o.ä.) ausgegeben werden.

Die Methode **umstapeln(von, ziel)** entfernt das oberste Element des Stapels von und legt es auf den Stapel ziel. Sie dient lediglich dazu, die Methode **lagerAuffuellen ( )** übersichtlich zu gestalten.

Implementiere die Klasse Lager.

In der Klasse Test kannst du mit der Methode **ausfuehren()** deine Methoden testen.

Übung 3

c)*

Implementiere die Methode **lagerAuffuellen2( )**, in der die Reihenfolge des Auffüllens geändert ist:
Zuerst wird ein Element auf s1 abgelegt, das nächste auf s2, das nächste auf s3, das nächste auf s1, usw.