Listen

M. Jakob

Gymnasium Pegnitz

28. Oktober 2018

Inhaltsverzeichnis

Hinführung: Wartenschlangen (6 Std.)

Grundprinzip von Listen — Rekursion (10 Std.)

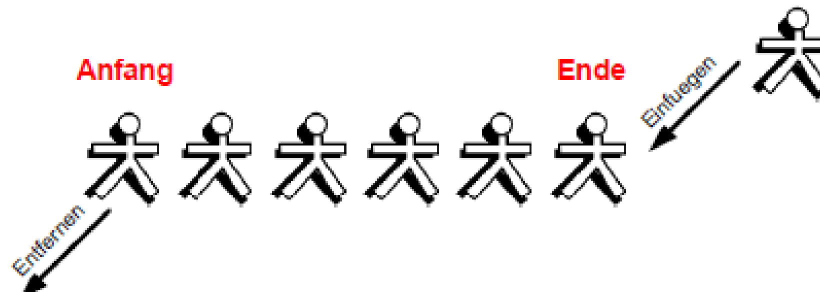
Die einfach verkettete Liste als Kompositum (10 Std.)
- Klasse `LISTENELEMENT` ? Entwurfsmuster Kompositum
- Suchen, Löschen, Einfügen

Spezialfall: Stapel Und Schlangen — FILO und FIFO (2 Std.)

Grundprinzip — FIFO-Prizip

Grundprinzip — FIFO-Prizip

- Neue Objekte werden hinter dem Ende eingefügt.
- Gespeicherte Objekte werden vom Anfang abgerufen und dabei aus der Schlange entfernt.
- Die gespeicherten Objekte werden in der gleichen Reihenfolge wieder ausgegeben, wie sie eingetragen wurden (FIFO-Prinzip, „first in first out“).

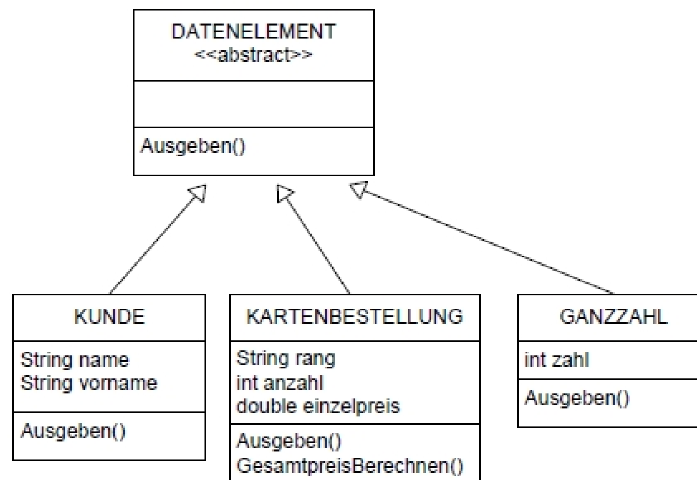


Beispiele von FIFO-Puffern

Beispiel	Datenelemente
Kunden an Supermarktkasse	Menschen
Taxi-Wartestand	Autos
Druckerwarteschlange	Dokumente
to-do-Liste	beliebiege Objekte

Implementierung von abstrakten Datensätzen

Klassendiagramm



- Ü 1.1: Implementieren

Trennung von Inhalt und Struktur

Trennung von Inhalt und Struktur

Die Strukturierung von Datenelementen (z.B. in einem FIFO-Puffer) kann unabhängig von deren Inhalt erfolgen. Dazu muss man

- eine abstrakte Schnittstelle `Datenelement` deklarieren,
- konkrete Datensätze als Unterklassen von `Datenelement` implementieren und die
- Warteschlangenverwaltung für die Datenelemente implementieren

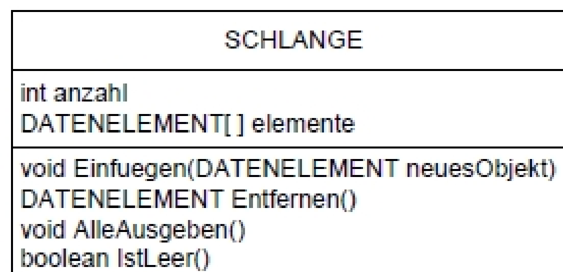
Skizze . . .

Implementieren der Liste

- KD Array

Benötigte Methode: . . .

Klassendiagramm Warteschlange als Array



Übungen

- Ü 1.2: Diskutiere Vor- und Nachteile der Implementationen

Eine Warteschlage sei als Array implementiert (muss nicht erledigt werden)

(a) Erkläre in Worten, welche Aufgaben die Methoden Einfügen() und Entfernen() erledigen müssen.
(b) Formuliere die beiden Methoden in Pseudocode.
(c) Welche Eigenschaft von Warteschlagen kann durch die Implementierung als Array nicht modelliert werden?

- Ü 1.3: Implementieren eine Warteschlange nach obigem Klasendiagramm

Rekursion

- Kassier, Botschaft an alle Kunden: iterativ und rekursiv
- Listenlänge bestimmen: iterativ und rekursiv durchspielen

Definition Rekursion

Von Rekursion spricht man, wenn bei der Lösung eines Problems bei mindestens einem der Teilprobleme die gleiche Stategie angewendet werden muss, wie bei dem Gesamtproblem, nur in verkleinerter, oder irgendwie anders reduzierter Form. Durch eine Abbruchbedingung terminiert die Rekursion.
Programmiertechnisch bedeutet das, dass eine Methode sich selbst aufruft.

Iteration

Definition Iteration

Von Iteration spricht man, wenn man bei der Lösung eines Problems Schritt für Schritt vorgeht, ohne rekursive Methodenaufrufe zu verwenden
Programmiertechnisch bedeutet das, dass Methoden sich nicht selbst aufrufen, sondern Schleifen und Sequenzen benutzen.

- Beispiele

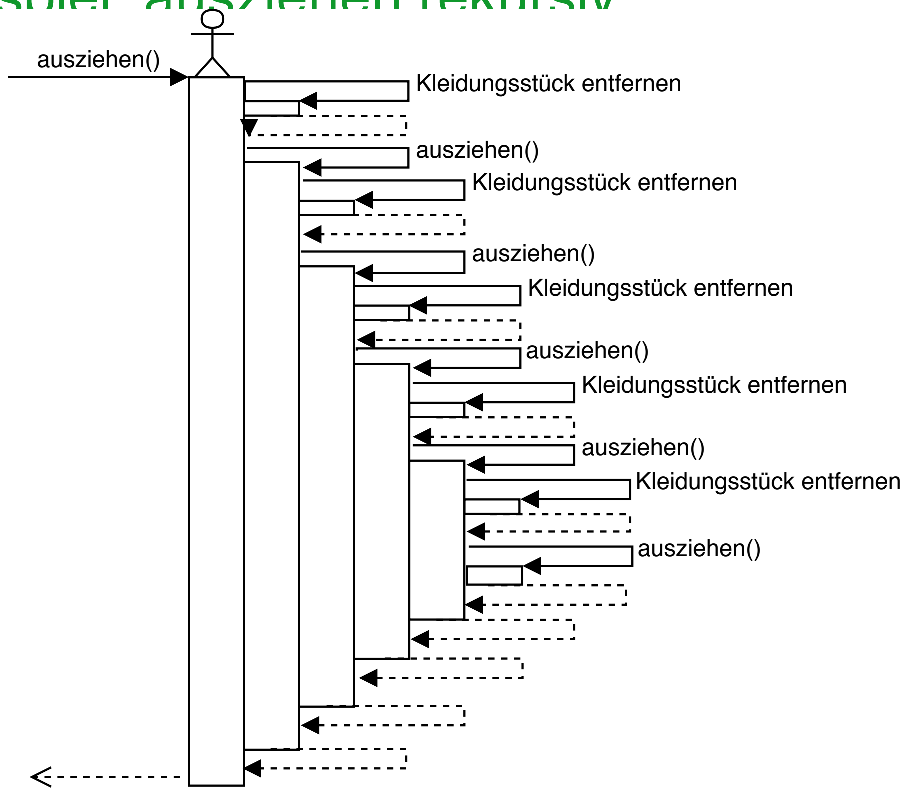
Rekursive Methoden in Pseudocode

```
1 rekMethode
2   wenn Abbruchbedingung erfüllt
3     gib ergebnis aus
4   sonst
5     bestimme Teilergebnis und wende
6     rekMethode auf den Nachfolger an
```


Beispiel: ausziehen rekursiv

```
1 ausziehen()
2   wenn ( kein Kleidungsstück vorhanden )
3     tue nichts
4   sonst
5     Kleidungsstück entfernen
6     ausziehen()
```

Beispiel: ausziehen rekursiv



Beispiel: ausziehen iterativ

```
1 ausziehen()
2   solange ( Kleidungsstück vorhanden )
3       Kleidungsstück entfernen
```

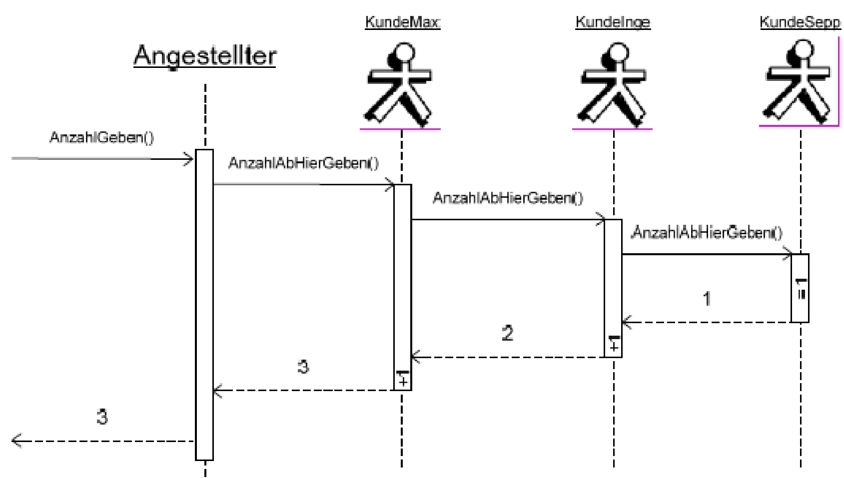
Beispiel: Listenlänge bestimmen rekursiv

```
1 listenlaenge()
2   wenn ( Ende erreicht )
3     return 1
4   sonst
5     return 1+listenlaenge()
```


Grundprinzip rekursiver Methodenaufrufe

Beispiel Listenlänge bestimmen:
Abbruchbed.: Listenende erreicht (ergebnis = 0)
rekursiver Aufruf: 1 + Listenlänge ab Nachfolger

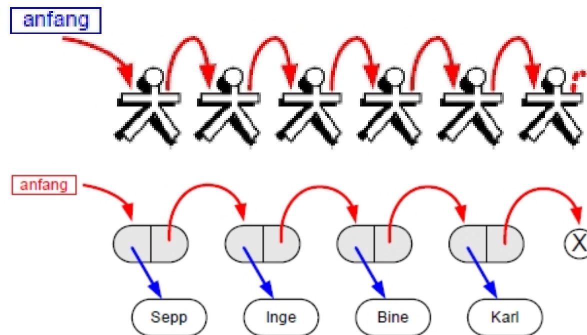
Syntaxdiagramm



- Ü 2.1: Implementierung einfacher rekursiver Methoden
`run:Arbeitsmaterial/ImplementierungEinfacherRekMethoden/package.bluej`

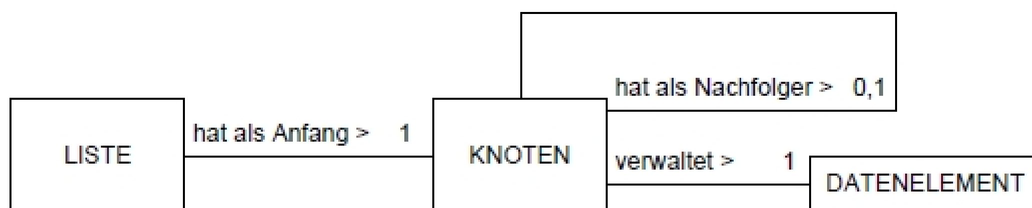
Listen als rekursive Datenstrukturen

- Kundenwarteschlange, Referenz, Botschaft, einfache Verkettung



- Jede Liste besitzt einen Anfangsknoten
- Jeder Knoten besitzt ein Datenelement und einen (ggf. leeren) Nachfolger, der ebenfalls ein Knoten ist.

Klassendiagramm einer einfach verkettenen Liste

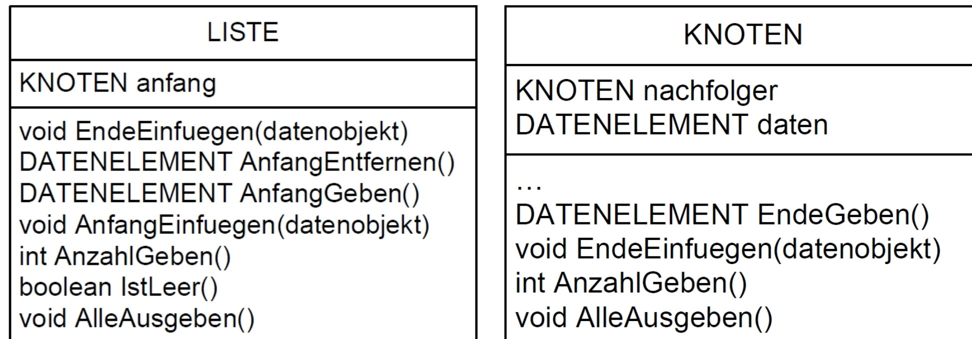


Implementation einer Liste

- KD Knoten und Liste

Benötigte Methode: ...

Klassendiagramm Liste und Konten



Übungen

- Ü 2.2: Implementation der Datenstruktur
- Ü 2.3: Formuliere die rekursive Methode `ListenlaengeBestimmen()` in Pseudocode
- Ü 2.4: `Liste.AnfangEntfernen()` symbolisch, als Pseudocode und in Java (vgl. HR S. 17)
- Ü 2.5: `Liste.EndeEinguegen()` symbolisch, als Pseudocode und in Java (vgl. HR S. 18)
- Ü 2.6: Restliche Methoden implementieren.

In diesem Abschnitt

Die einfach verkettete Liste als Kompositum (10 Std.)

Klasse `LISTENELEMENT` ? Entwurfsmuster Kompositum

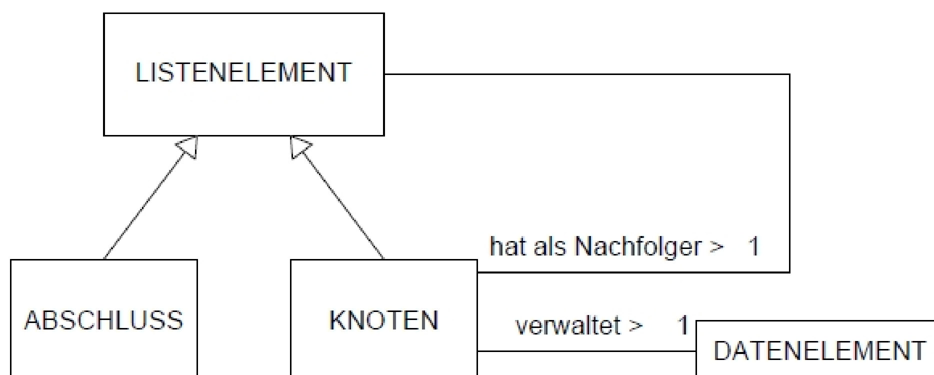
Suchen, Löschen, Einfügen

Klassendiagramm einer einfach verkettenen Liste

Sicherkeitsabfrage an die leere Liste machten den Programmcode aufwändig.

Idee: Abstrakte Klasse `LISTENELEMENT` das entweder ein Knoten ist oder der Abschluss

Skizze. . .



Übungen

- KK LISTENELEMENT an linker Tafel

- Ü 3.1: Implementation der Klasse `LISTENELEMENT`

Implementiere die abstrake Klasse `LISTENELEMENT` so, dass sie die Klasse `KNOTEN` aufnehmen kann. (Vorlage: ListeKompositumVorlage) Wie abstrakte Klassen implementiert werden kannst du an der Klasse `Datenelement` erkennen.

- Ü 3.2: Implementation der Getter- und Setter-Methoden

Passe die Getter und Setter-Methoden der Klasse `KNOTEN` auf die neuen Gegebenheiten an und Implementiere sie auch in der Klasse `ABSCHLUSS`.

- Ü 3.3: Implementation der rekursiven Methoden

- Ü 3.4: Anpassung der Klasse `LISTE`

In diesem Abschnitt

Die einfach verkettete Liste als Kompositum (10 Std.)

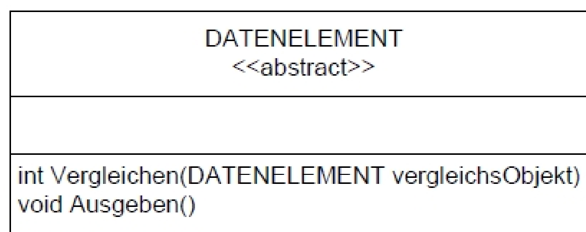
Klasse `LISTENELEMENT` ? Entwurfsmuster Kompositum

Suchen, Löschen, Einfügen

Vergleichen von Datenelementen

Um spezielle Elemente der Liste ausfindig zu machen, muss man sie mit anderen vergleichen können. Unsere Vergleichsoperation soll nicht nur Gleichheit, sonder auch die Kleiner- und Größer-Relation beinhalten.

Klassendiagramm Warteschlange als Array



Kunde.Vergeleichen(DATENELEMENT)

```
1      int Vergleichen(DATENELEMENT
          vergleichsObjekt)
2      {
3          KUNDE vergleichsKunde = (KUNDE)
              vergleichsObjekt;
4          int ergebnis;
5
6          ergebnis = name.compareTo(
              vergleichsKunde.name);
7          if (ergebnis == 0)
8              ergebnis = vorname.compareTo(
                  vergleichsKunde.vorname);
9
10         return ergebnis;
11     }
```

Übung

- Ü 3.5: `Vergleichen(DATENELEMENT)` in restlichen Datenelementen implementieren

Implementiere `Karte.Vergleichen(DATENELEMENT)` und `Ganzzahl.Vergeleichen(DATENELEMENT)`. Vergleiche in der Klasse `Karte` das Attribut `rang` mit der `compareTo`-Methode. In der Klasse `Ganzzahl` sollen die Werte $-1, 0, +1$ zurückgegeben werden, je nachdem, ob die Vergleichszahl größer, gleich oder kleiner der Ganzzahl ist. Achte auf die Tyenapassung (typecasting)

Listen
└ einfache Verkettung
└ Suchen, Löschen, Einfügen

Übung

- Ü 3.6: `suchen(DATENELEMENT)`

Gegeben ist eine Liste mit 3 Elementen. Zeichne die Sequenzdiagramme falls `d2` bzw. ein nicht vorhandenes Element gesucht wird und implementiere anschließend `boolean suchen(DATENELEMENT)` in dem Paket `LISTE`

- Ü 3.7: `entfernen(DATENELEMENT)`

Zeichne geeignete Sequenzdiagramme und implementiere anschließend `LISTENELEMENT entfernen(DATENELEMENT)` in dem Paket `LISTE`

- Ü 3.8: `sortiertEinfuegen(DATENELEMENT)`

Zeichne geeignete Sequenzdiagramme und implementiere anschließend `LISTENELEMENT sortiertEinfuegen(DATENELEMENT)` in dem Paket `LISTE`

Stapel

Es soll die Klasse `STAPEL` implementiert werden, die die Klasse `LISTE` nutzt und nach dem FILO-Prinzip arbeitet.
Erforderliche Methoden der Klasse `STAPEL`

`STAPEL()` Konstruktor

`Einfuegen(DATENEELEMENT)` Legt (die Referenz auf) das Datenelement auf dem Stapel ab (engl. push)

`Entfernen(DATENEELEMENT)` Das oberste (zuletzt abgelegt) Element wird vom Stapel entfernt und (eine Referenz auf) dieses Datenelement zurückgegeben (engl. pop)

`NaechstenGeben()` liefert (eine Referenz auf) das oberste Datenelement, ohne dieses aus dem Stapel zu entfernen (engl. top oder peek).

`IstLeer()` prüft, ob auf dem Stapel irgendein Datenelement (bzw. eine Referenz auf ein Datenelement) abgelegt ist.

Übung

- Ü 4.1: `STAPEL`

Implementiere die Klasse `STAPEL`. Sie soll nur ein Attribut `LISTE liste` besitzen.

- Ü 4.2: `SCHLANGE`

Implementiere die Klasse `SCHLANGE`, die im Gegensatz zu Stapel nicht nach dem FILO, sondern dem FIFO-Prinzip arbeitet.