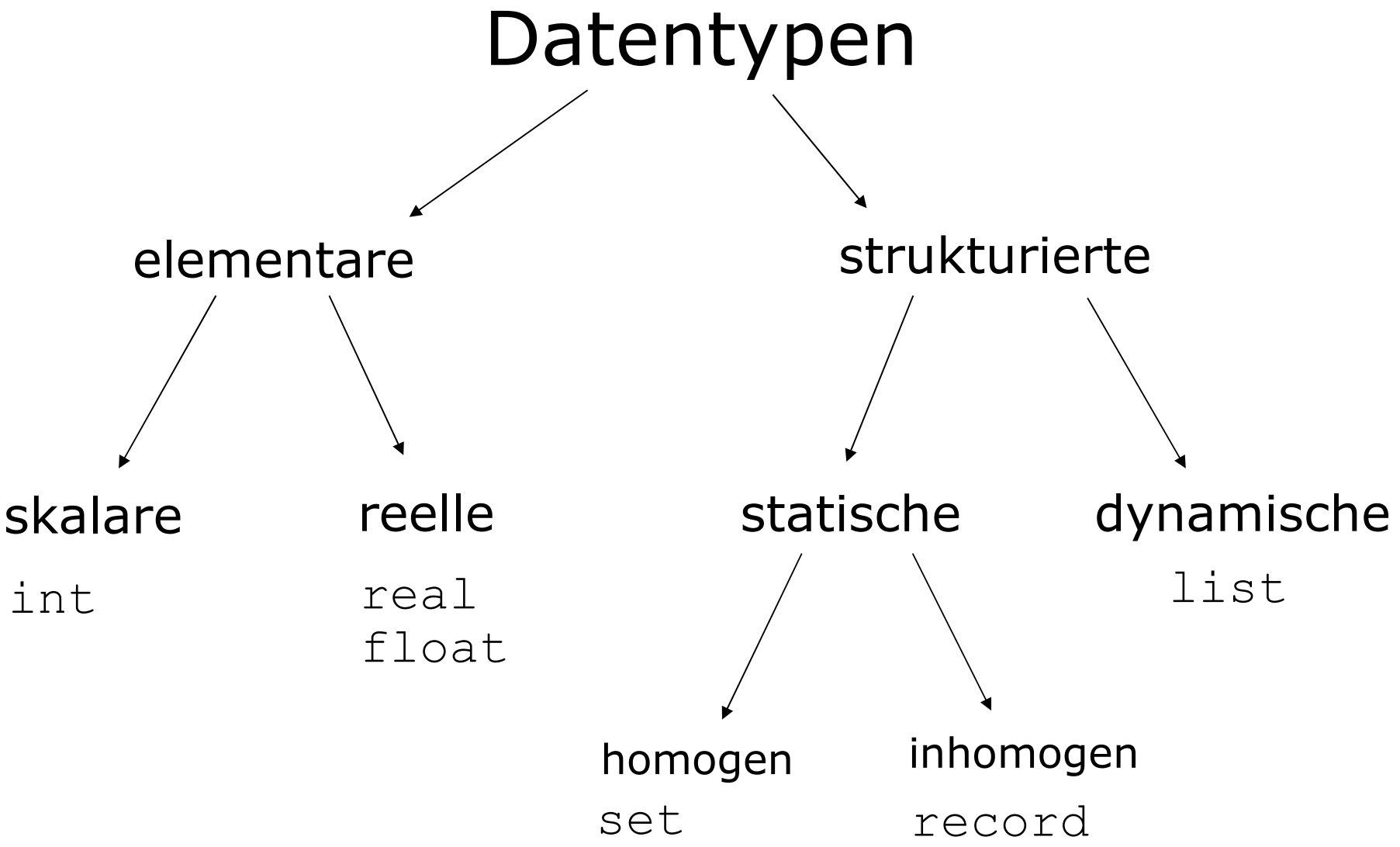

Datentypen
elementare
strukturierte
skalare
int
reelle
real
float
statische
dynamische
list
homogen
set
inhomogen
record

Der elementare Datentyp **nat**

bestehend aus einer Objektmenge und den darauf definierten Operationen.
Die Objektmenge von **nat** sein hier die Menge der natürlichen Zahlen.

Syntax:

```
Name          nat
Operationen   new  :              -> nat   (Konstruktor)
              +    : nat x nat -> nat
              -    : nat x nat -> nat
```

Sematik:

```
Menge         nat : Menge der natürlichen Zahlen
Funktionen    new  = {}
              +    : nat x nat (Summe)
              -    : nat x nat (Differenz)
```

Probleme vordefinierter Datentypen

Die interne Repräsentation und die Operatoren der in den Programmiersprachen vordefinierten elementaren Datentypen sind meist stark programmiersprachen- und maschinenabhängig:

- Wieviel Speicher belegt der Typ **`nat`**?
- An welcher Speicheradresse steht welches Byte?
- Wie funktionieren die Operatoren?
- ...

Eine Prozedur, die den Datentyp **`nat`** benutzt ist darauf angewiesen, daß dessen Operatoren immer gleich 'funktionieren'.

Typen von Operatoren

Konstruktoren:
erzeugen Instanzen

```
Auto mein_Auto = new Auto();
String str = "Datentypen";
char[] zeichen;
```

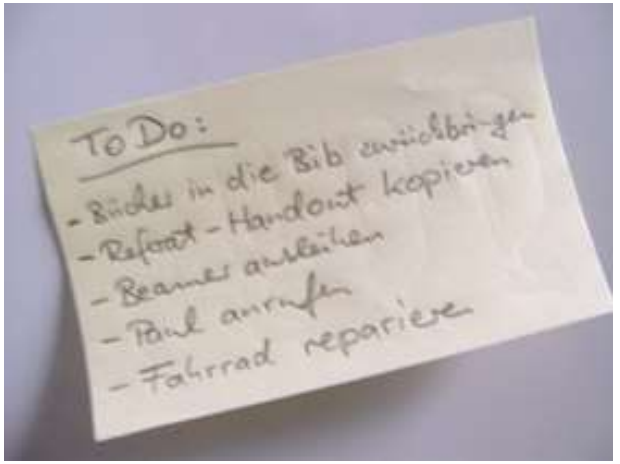
Modifikatoren:
verändern eine Instanz

```
zeichen[0] = "a";
```

Observatoren/Selektoren:
liefern Informationen über eine Instanz

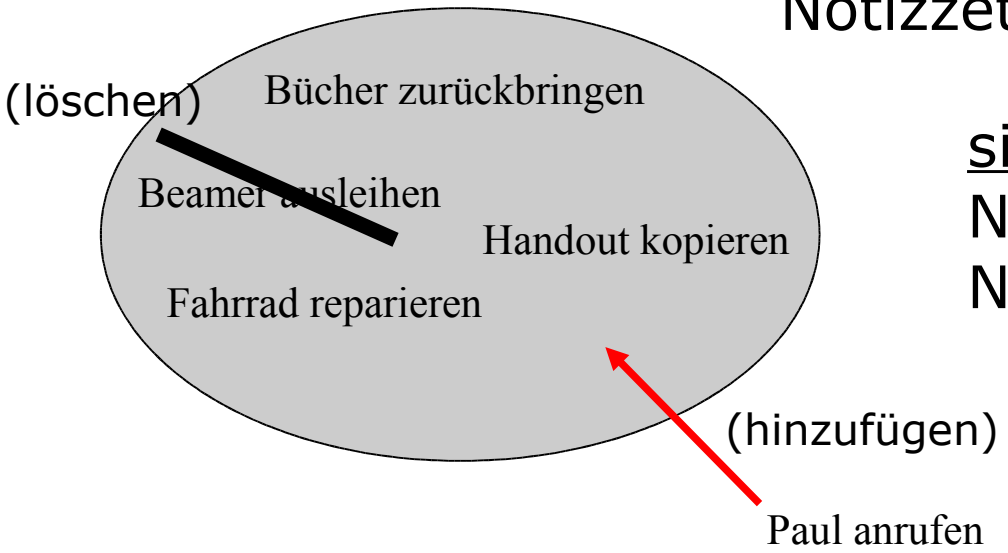
```
charAt(int index);
```

komplexere Datenstrukturen



Beispiel:
Notizzettel mit den Dingen, die man heute noch erledigen muß

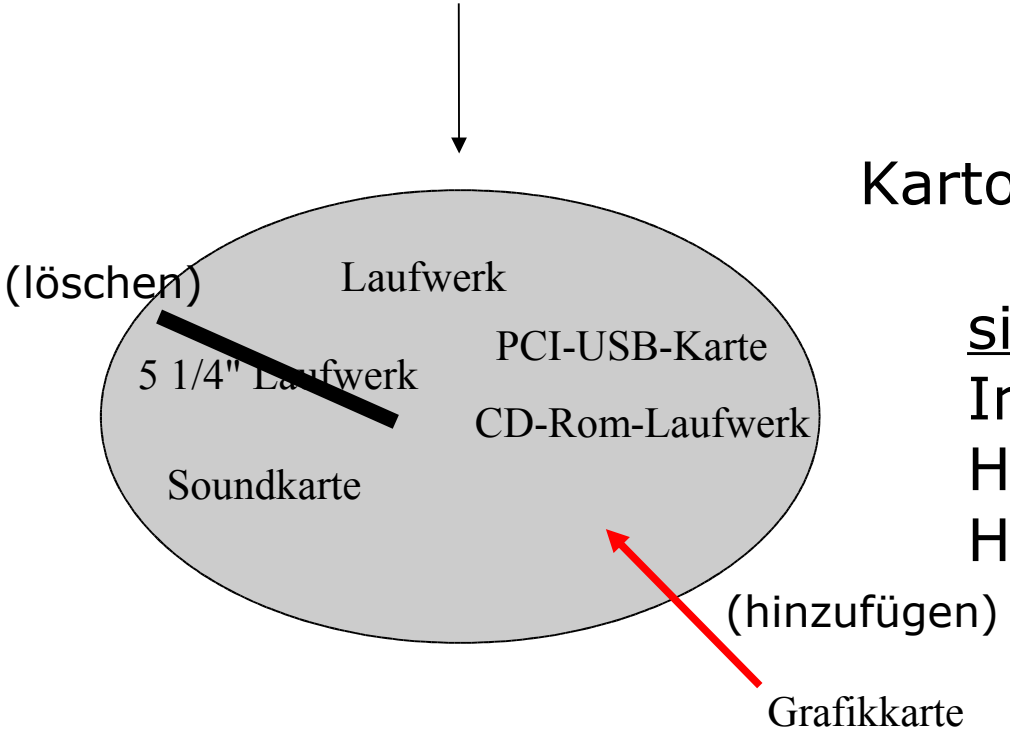
Notizzettel als Menge der Notizen



sinnvolle Operatoren:
Notiz lesen, Notiz hinzufügen, Notiz streichen, ...

komplexere Datenstrukturen

Beispiel:
Eine Karton mit
alter Hardware



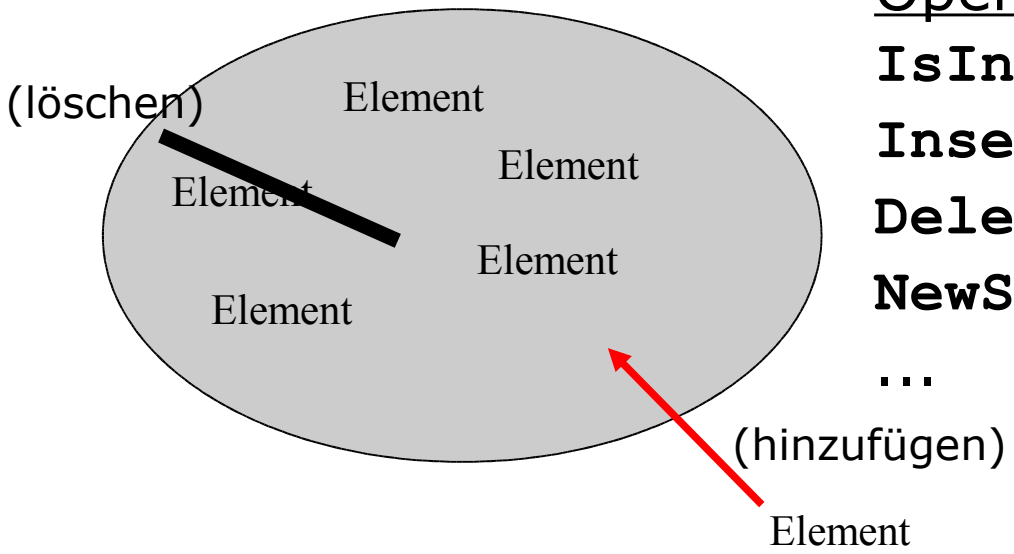
Karton als Menge

sinnvolle Operatoren:
Im Karton nachsehen,
Hardware hineinlegen,
Hardware herausnehmen, ...

komplexere Datenstrukturen

Beide Datenstrukturen (der Notizzettel und die Kiste) sind einfache Mengen von Elementen und unterscheiden sich nicht bei den Operatoren. Ineffizient wäre hier, je eine unterschiedliche Datenstruktur zu konstruieren.
Ziel: Wiederverwendbarkeit
Lösung: Abstraktion

Menge



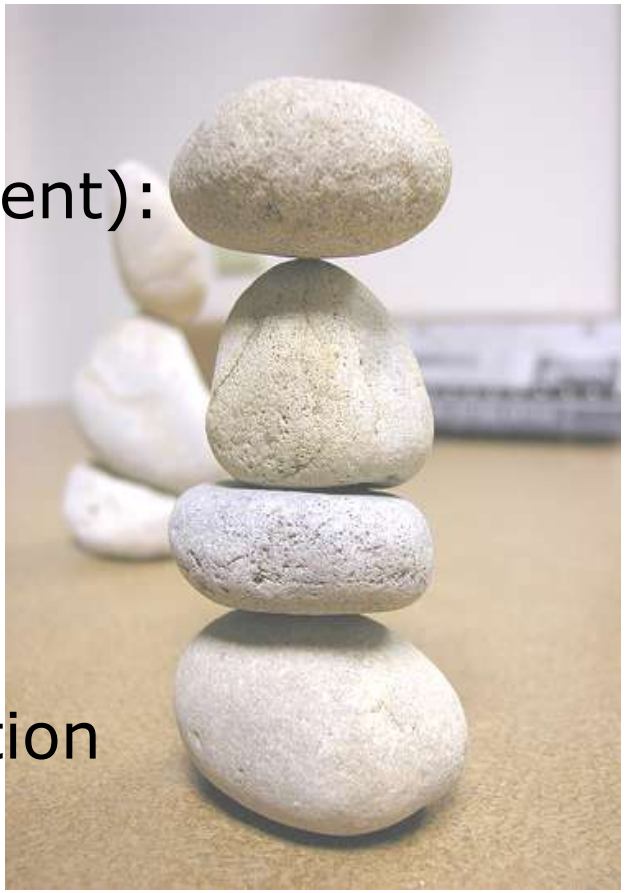
Operatoren:
`IsIn` (ist ein Element in der Menge?)
`Insert` (Element hinzufügen)
`Delete` (Elemet löschen)
`NewSet` (neue Menge; Konstruktor)
...

abstrakte Datenstrukturen

- von dem konkreten Problem verallgemeinert
- Spezifikation ist unabhängig von ihrer Implementierung und der konkreten Verwendung => Wiederverwendbarkeit in anderen Programmen
- Die interne Realisierung der Mechanismen der Operatoren und der Anordnung der Daten spielt für den 'Benutzer' der Datenstruktur keine Rolle, kann und soll sogar verborgen werden (Geheimnisprinzip).
- Es gibt eine präzise beschriebene Schnittstelle mit den erforderlichen Operatoren, auf die Daten kann nur über diese Schnittstelle zugegriffen werden (Kapselung).
- Die Datenstruktur kann ohne Auswirkungen auf Programme, die diese benutzen, geändert werden.

Stack

Ein Stapel kann eine Menge von Objekten aufnehmen und gibt diese entgegengesetzt zur Reihenfolge der Eingabe wieder zurück.
('Last In, First Out-Prinzip' = 'LIFO')



Operatoren (Zugriff auf das oberste Element):
- *push* (Hinzufügen eines Objektes)
- *pop* (Entfernen des Objektes)
- *top* (Auslesen des Objektes)

Anwendungen (z.B.):
- Auswertung von Klammerausdrücken
- Auflösung von Ausdrücken in Infix-Notation
- Parsing

Queue (Warteschlange)

Ein Queue kann eine Menge von Objekten aufnehmen und gibt diese in der Reihenfolge ihres Einfügens wieder zurück.
('First In, First Out-Prinzip' = 'FIFO')

<u>Operationen</u>:
- *enqueue* (Einreihen eines Objektes)
- *dequeue* (Entfernen eines Objektes)

Anwendung (z.B.):
- Datenübergabe zwischen asynchronen Prozessen (Tastaturpuffer o.ä.)

Baum

Die Positionen der Elemente im Baum ist von ihrem Wert oder Schlüssel abhängig.

Operationen:

- *tree traversal* in Pre-, Post-, oder Inorder-Reihenfolge
- Suchen nach einem Wert/Schlüssel
- Hinzufügen und Löschen von Elementen

Anwendung (z.B.):

- Anwendungen, die Suchen und Sortieren erfordern.

Lexika

- Wiederverwendbarkeit
- Zugriff über eine standardisierte Schnittstelle
- Interne Organisation für Abfragen unwichtig
- Die Daten müssen geschützt sein

Operatoren (z.B.):

- Frequenz, Häufigkeitsklasse einer Wortform
- Flexionsformen einer Wortform
- Synonyme, Oberbegriffe
- ...

Textkorpora

- Maschinenlesbare Korpora besitzen meist sehr spezifische technische Merkmale (=> schlecht wiederverwendbar)
- bei Änderungen müssen auch die zugreifenden Prozeduren geändert werden (=> schlecht wartbar)
- 'jeder' Benutzer erstellt und verwendet eigene Zugriffsoperationen (ineffizient)

Lösung: Abstraktion - Korpus als Datenstruktur (=>Geheimnisprinzip, Kapselung, Zugriff über Schnittstelle)

Operatoren (z.B.):

- nächste Einheit [Satz, Silbe, ...]
- Länge oder Position im Text