Objektorientierte Programmierung

Geschichte

Dahl, Nygaard: Simula 67
(Algol 60 + Objektorientierung)

Kay et al.: Smalltalk
(erste rein-objektorientierte Sprache)

Object Pascal, Objective C, C++
(wiederum Erweiterungen einer gegebenen Basissprache)

Java
(C++ ohne Hacks?)

Beispiel aus vorigem Kapitel

Lineare Suche in Arrays:

```
public static boolean
enthaelt(int[] array, int fuellstand, int elem) {
    int i;

    for (i = 0; i <= fuellstand; i++) {
        if (array[i] == elem) {
            return true;
        }
    }
    return false;
}
```

Problem 1

Array und Füllstand müssen für jede Funktion separat als Parameter übergeben werden.

$\rightsquigarrow$ Lästig, insbesondere falls mehrere Mengen in einem Programm benutzt werden (Verwechselungsgefahr).

Lösung: zusammengehörige Variable zusammenfassen (record, struct).
z.B. in C:

```
struct Menge {        /* KEIN JAVA! */
  int array[1000];
  int fuellstand;
}
```

Problem 2

Beispiel:

> Ein Programm benutzt unsortierte Arrays, um Mengen zu implementieren.
> Es liefert korrekte Ergebnisse (wenn auch zu langsam).

Problem 2

Beispiel:

> Ein Programm benutzt unsortierte Arrays, um Mengen zu implementieren.
> Es liefert korrekte Ergebnisse (wenn auch zu langsam).
>
> Einer der Programmierer entschließt sich, unsortierte Arrays durch sortierte Arrays zu ersetzen.
> Er tauscht darum die Funktionsrümpfe der Funktionen `enthaelt` und `fuegeEin` aus.

Problem 2

Beispiel:

> Ein Programm benutzt unsortierte Arrays, um Mengen zu implementieren.
> Es liefert korrekte Ergebnisse (wenn auch zu langsam).
>
> Einer der Programmierer entschließt sich, unsortierte Arrays durch sortierte Arrays zu ersetzen.
> Er tauscht darum die Funktionsrümpfe der Funktionen `enthaelt` und `fuegeEin` aus.
>
> Daraufhin liefert das Programm falsche Ergebnisse. Warum?

Problem 2

Analyse des Programms zeigt:

Im Programm treten unsortierte Arrays auf.

Die neue Funktion `enthaelt` liefert dann falsche Ergebnisse.

Problem 2

Analyse des Programms zeigt:

Im Programm treten unsortierte Arrays auf.
Die neue Funktion `enthaelt` liefert dann falsche Ergebnisse.

Des Rätsels Lösung:

Einer anderer Programmierer hatte nicht die Funktion `fuegeEin` aufgerufen, sondern „von Hand“ die Variable `fuellstand` erhöht und den neuen Wert am Ende des Arrays eingetragen.

Dieser Programmteil wurde nicht geändert.

Problem 2

Wenn eine Komponente A auf Implementierungsdetails
einer anderen Komponente B zugreifen kann,
dann kann die Implementierung von B nicht ohne weiteres
ausgewechselt werden.

Lösung

Forderung:

Trennung von Schnittstelle (was?) und Implementierung (wie?).

Kapselung der Implementierung:
Der Rest des Programms darf nur über die Schnittstelle auf die Implementierung zugreifen.

Benutzer (und Implementierer) einer Komponente bekommen die wesentlichen Informationen: Schnittstelle/Verhalten/Kontrakt (und nicht mehr).

Kapselung

Weitere Gründe:

- Zugriff auf interne Variable erhöht generell Gefahr von Fehlern.
- Überschaubarkeit.
- Bei größeren Software-Projekten:
  - Entwurf: (wiederholte) Zerlegung in Komponenten.
  - Implementierung: verschiedene Komponenten parallel!
  - Integration: Zusammenbau der Komponenten.

Idee

Zusammenfassen:

- Zusammengehörige Variable (wie bei record/struct),
- Operationen, mit denen diese Variable manipuliert werden dürfen.
- Zusätzlich:
  - Unterscheidung zwischen öffentlichen und privaten Variablen und Operationen

Objekte und Klassen

Objekt = Zustand (Variable) + Verhalten (Operationen, „Methoden“)

Jedes Objekt ist Exemplar (Instanz) einer Klasse.
Klassen definieren das Verhalten ihrer Exemplare.

Objekte und Klassen

```
public class Menge {
    int[] array;                          // Variable
    int fuellstand;

    ...                                   // Konstruktoren

    public boolean enthaelt(...) {        // Methoden
        ...
    }
    public void fuegeEin(...) {
        ...
    }
}
```

Objekte und Klassen

Syntax: Methoden sind Bestandteile des Objekts und werden auch so angesprochen (analog zu Recordkomponenten).

```
Menge mg; int n;
...
n = mg.fuellstand;
mg.fuegeEin(0);      // statt: fuegeEin(mg, 0)
```

Vergleiche:

```
String s1, s2;
TextField tf;
...
s1 = tf.getText();
s2 = s1.substring(0,1);
```

Objekte und Klassen

Nomenklatur:

Ein anderes Objekt schickt dem Objekt `mg` eine Nachricht `fuegeEin(0)`.
`mg` führt daraufhin eine Methode aus.
(Die Verantwortung für das Interpretieren der Nachricht liegt beim Empfänger `mg`.)

Berechnung = Kommunikation:

gut geeignet für hochgradig interaktive Programme,
gut geeignet für Simulationen der realen Wert,
gut verträglich mit Parallelismus.

Objekte und Klassen

```
public class Menge {
    // Variable

    int[] array;          // ohne public: nicht (bzw. nur
    int fuellstand;       // bedingt) von außen zugreifbar.


    // Konstruktoren (später)
    ...
```

Objekte und Klassen

```
// Methoden

public boolean enthaelt(int elem) {
    int i;

    for (i = 0; i <= fuellstand; i++) {
        if (array[i] == elem) {
            return true;
        }
    }
    return false;
}
```

Objekte und Klassen

```
    public void fuegeEin(int elem) {
        fuellstand++;
        array[fuellstand] = elem;
    }
}
```

Objekte und Klassen

Erzeugung von Objekten:

```
Menge mg;
```

```
mg = new Menge();
```

Objekte und Klassen

Konstruktoren sind optional:

falls kein Konstruktor angegeben wird, werden die Variablen in Objekten mit `0`, `'\000'`, `false`, `null` initialisiert.

aber: das wäre hier ungünstig.
`fuellstand` sollte zu Beginn den Wert `-1` haben,
`array` sollte mit einem Array initialisiert sein.

Objekte und Klassen

```
public class Menge {
    int[] array;                        // Variable
    int fuellstand;

    Menge() {                           // Konstruktoren
        fuellstand = -1;                // beachte: kein Typ,
        array = new int[1000];          // kein return,
    }                                   // Name = Klassenname

    // Methoden wie bisher
    ...
}
```

Objekte und Klassen

Konstruktoren können parameterisiert sein:

```
Menge(int maxfuellstand) {
    fuellstand = -1;
    array = new int[maxfuellstand];
}
```

Aufruf:

```
Menge mg;
mg = new Menge(2000);
```

Noch ein Beispiel

```
class Zaehler {
    public int n;

    Zaehler() {
        n = 1;
    }

    public int wert() {
        return n;
    }

    public void inkr() {
        n++;
    }
}
```

Noch ein Beispiel

```
Zaehler z;
z = new Zaehler();

System.out.println(z.n);          // -> 1    (*)
System.out.println(z.wert());     // -> 1

z.inkr();
z.inkr();
System.out.println(z.wert());     // -> 3

z.n = 15;                         //         (*)
System.out.println(z.wert());     // -> 15

// (*) geht nur, falls n als public deklariert ist!
//     (was man gewöhnlich NICHT tut)
```

Noch ein Beispiel

Auf Objekte wird nur über Referenzen zugegriffen:

Gleichheitstest (`z1 == z2`):

überprüft, ob `z1` und `z2` dasselbe Objekt bezeichnen.

Zuweisung (`z1 = z2`):

`z1` bezeichnet nun dasselbe Objekt wie `z2`:

```
Zaehler z1, z2;
z1 = new Zaehler();
z2 = z1;
System.out.println(z1.wert()); // -> 1
z2.inkr();
System.out.println(z1.wert()); // -> 2
```