

Programmieraufgabe am 30.11.2005

- Anmeldung unter der bekannten Webseite http://www.informatik.uni-koeln.de/ls_speckenmeyer/teaching/ws05-06/programmierkurs.html
- Angabe einer Präferenzordnung unter den 5 Zeitblöcken 8-10, 10-12, 13-15, 15-17, 17-19
- Bestätigungsmail an angegebene E-Mail-Adresse
- Muss für Anmeldung beantwortet werden
- Frist bis 16.11.2005, dann Benachrichtigung über tatsächlichen Prüfungszeitpunkt

Verbunde (records, structs)

- Arrays modellieren also Beziehungen zwischen Elementen gleichen Typs
- Oft bestehen aber auch **Beziehungen zwischen Werten unterschiedlichen Typs**
 - Etwa zwischen Name und Monatsverdienst eines Beschäftigten
- Wir verbinden zusammengehörige Daten unterschiedlichen Typs zu einem **Verbund** (*record, structure, struct*)
- **Beispiel:** Stammdaten

Name	"Mustermann"
Vorname	"Martin"
GebTag	10
GebMonat	05
GebJahr	1930
Familienstand	"verheiratet"
...	...

Verbunde (records, structs)

- Übliche Syntax zur Auswahl: Punkt-Notation
 - Beispiel:

Sei ein konkretes Stammdatenblatt s gegeben. Dann ist

`s.Name = "Mustermann"`

der Wert der Komponente Name von s . Entsprechend gilt `s.GebTag = 10` usw.

- Komponenten eines Verbunds können von beliebigem Typ sein
 - Also auch wieder Verbunde, Reihungen, etc.

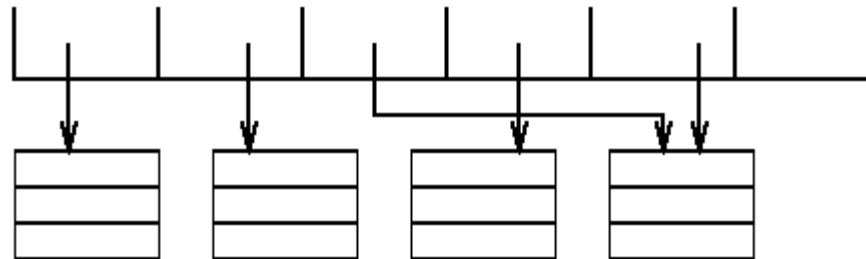
Modellierung des Enthaltenseins - Referenzen

- Ein Verbund kann in einem anderen enthalten sein
 - Vgl. Beispiel von Datum und Stammdatenblatt
 - Diese Beziehung des Enthaltenseins (*containment*) kann auf zweierlei Arten modelliert werden
 - Als Enthaltensein durch Wert (*by value*)
 - Als Enthaltensein durch Referenz (*by reference*)
-

Modellierung des Enthaltenseins - Referenzen

- **Beispiel:**

- Studentin **Musterfrau** belegt zwei verschiedene Übungen
 - Übung durch Verbund realisiert
 - Enthält u.a. String **Übungsleiter**, Tabelle (Reihung) mit Stammdaten des Studierenden sowie einer Tabelle mit Punkten
 - Stammdatenblatt **Musterfrau** also in zwei verschiedenen Verbunden realisiert
 - Wenn dies durch *Enthaltensein durch Wert* modelliert wird, dann existieren zwei *separate Exemplare* des Stammdatenblatts. Wenn dies durch *Enthaltensein durch Referenz* modelliert wird, dann existiert nur **ein Exemplar** des Stammdatenblatts **Musterfrau**, auf das beide Verbunde *verweisen*



- Idee: **Datenstrukturen + Algorithmen** gehören zusammen
- Ein **abstrakter Datentyp** (*abstract data type*) bündelt die Definition einer **Trägermenge** (*carrier set*) von **Objekten** mit einer **Spezifikation** der zugehörigen Operationen (**Methoden**)
 - **algebraischer abstrakter Datentyp**:
 - Spezifikation algebraisch durch Gleichungen
- Datenrepräsentation und Implementierung der Methoden bleiben hinter einer abstrakten **Aufrufsstelle** (*call interface*) verborgen
 - Daten in den Objekten können nur über die Methoden manipuliert werden
 - **Geheimnisprinzip** (*principle of information hiding*)
 - Dadurch muss nur noch der **Hersteller (Programmierer)** des abstrakten Datentyps die Datenrepräsentation und die **Programmierung der Funktionen verstehen**
 - **Benutzer** braucht nur die Funktionsweise der **Schnittstelle zu verstehen**
- Beispiele:
 - $\langle \mathbb{N}; 0, 1, +, -, * \rangle$

$\langle B; 0, 1, \wedge, \vee, \neg \rangle$

- Der abstrakte Datentyp **spezifiziert** erschöpfend die **Operationen**, die auf einem **Objekt ausgeführt** werden können
- Er kümmert sich nicht um die Realisierung
- Wegen dieser **Abstraktionsebene** heißt der Datentyp **abstrakt**
- Ein abstrakter Datentyp wird durch eine (Objekt-) **Klasse** implementiert (realisiert).
 - Danach sind die **Operationen sehr konkret ausführbar**
 - Z. B. Sortieren der Übungsliste
- Ein **Objekt** in einem abstrakten Datentyp ist im allgemeinen ein Verbund-Objekt, das **Daten** als **Objekt-Zustand** speichert
- Auf den Objekten operieren **Methoden**, die im Datentyp spezifiziert und in der **Klasse** implementiert sind und über das Objekt aufgerufen werden können.

- Ein **Objekt** ist ein (Daten-)Verbund zusammen mit den Algorithmen (Funktionen, Methoden), die auf Verbunden dieses Typs operieren können
 - Der Verbund speichert den Objekt-Zustand
 - Die Algorithmen (Methoden) definieren das mögliche Objekt-Verhalten
 - Manche Methoden können von außen auf dem Objekt aufgerufen werden
 - indem man dem Objekt eine (Aufruf-)Nachricht schickt
 - Andere Methoden sind von außen unsichtbar
 - Sichtbare Methoden definieren die **Schnittstelle** (*interface*) des Objekts
- Eine (Objekt-)**Klasse** fasst gleichartige Objekte zusammen
 - mit gleichen Methoden und Daten gleichen Typs
 - **Klassendeklaration** definiert den Typ des Verbundes und listet die Implementierung der Methoden
 - Eine Klasse **implementiert** einen abstrakten Datentyp
- Ein **Objekt** ist eine konkrete **Instanz** einer Klasse
 - ein neues Exemplar eines Verbundes mit separaten Daten
 - gebündelt mit den für die Klasse definierten Methoden
 - jede Methode wird auf einem konkreten Objekt aktiviert

Motivation



Wie würde man ein Datum speichern (z.B. 13. November 2004)?

3 Variablen

```
int day;  
String month;  
int year;
```

Unbequem, wenn man mehrere Exemplare davon braucht:

```
int day1;  
String month1;  
int year1;  
int day2;  
String month2;  
int year2;  
...
```

Idee: die 3 Variablen zu einem eigenen Datentyp zusammenfassen

Datentyp Klasse

Speicherung verschiedenartiger Werte unter einem gemeinsamen Namen

Deklaration

```
class Date {
  int day;
  String month;
  int year;
}
```

Felder der Klasse *Date*

Verwendung als Typ

```
Date x, y;
```

Zugriff

```
x.day = 13;
x.month = "November";
x.year = 2004;
```



Date-Variablen sind Zeiger auf Objekte

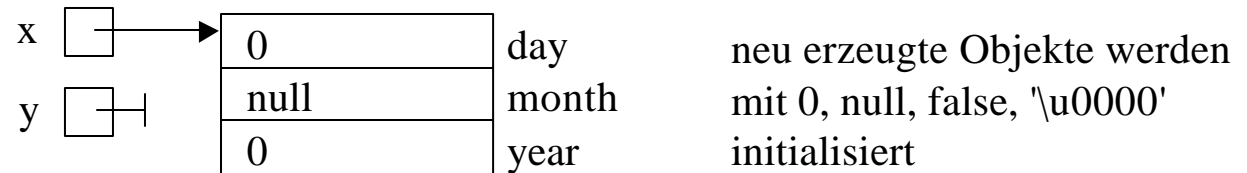
Objekte einer Klasse müssen vor ihrer ersten Benutzung erzeugt werden

Date x, y; reserviert nur Speicher für die Zeigervariablen



Erzeugung

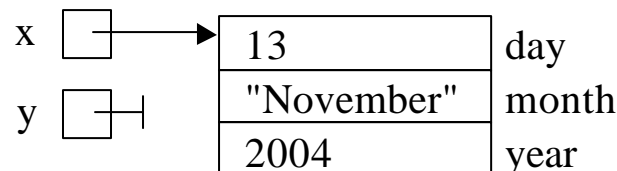
x = new Date(); erzeugt ein *Date*-Objekt und weist seine Adresse x zu



Eine Klasse ist wie eine Schablone, von der beliebig viele Objekte erzeugt werden können.

Benutzung

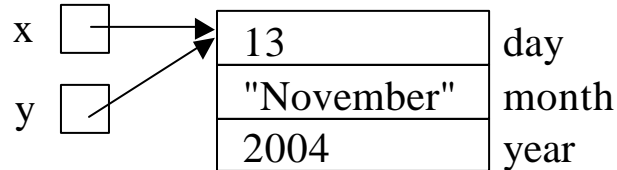
x.day = 13;
x.month = "November";
x.year = 2004;



Zuweisungen

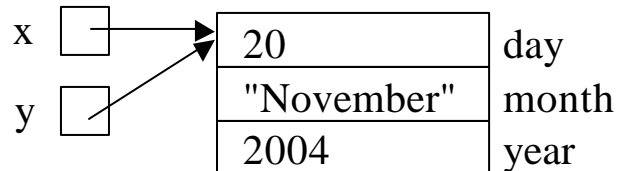


`y = x;`



Zeigerzuweisung!

`y.day = 20;`



ändert auch x.day!

Zuweisungen sind erlaubt, wenn die Typen gleich sind

```
class Date {  
    int day;  
    String month;  
    int year;  
}
```

```
class Address {  
    int number;  
    String street;  
    int zipCode;  
}
```

```
Date d1, d2 = new Date();  
Address a1, a2 = new Address();
```

```
d1 = d2;    // ok, gleiche Typen  
a1 = a2;    // ok, gleiche Typen  
d1 = a2;    // verboten: verschiedene Typen trotz gleicher Struktur!
```

Vergleiche



Zeigervergleich

`x == y` vergleicht nur Zeiger

`x != y`

`x < y` nicht erlaubt

`x <= y`

`x > y`

`x >= y`

Wertvergleich muß mittels Vergleichsmethode selbst implementiert werden

```
static boolean equalDate (Date x, y) {  
    return  
        x.day == y.day &&  
        x.month.equals(y.month) &&  
        x.year == y.year;  
}
```

Wo werden Klassen deklariert



In einer einzigen Datei (auf äußerster Ebene)

MainProgram.java

```
class C1 {  
    ...  
}
```

```
class C2 {  
    ...  
}
```

```
class MainProgram {  
    public static void main (String[] arg) {  
        ...  
    }  
}
```

In getrennten Dateien

C1.java

```
class C1 {  
    ...  
}
```

C2.java

```
class C2 {  
    ...  
}
```

MainProgram.java

```
class MainProgram {  
    public static void main (String[] arg) {  
        ...  
    }  
}
```

Übersetzung

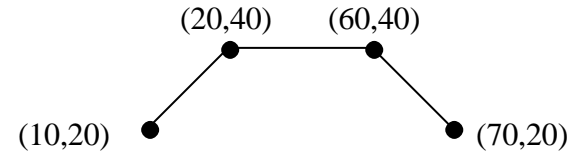
```
javac Mainprogram.java
```

```
javac Mainprogram.java C1.java C2.java
```

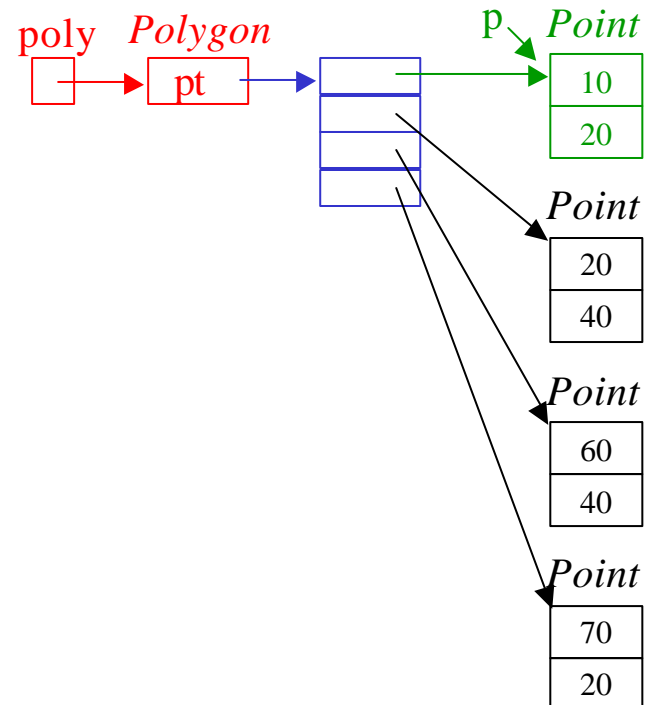
Beispiel: Polygone



```
class Point { int x, y; }  
class Polygon { Point[] pt; }
```



```
class Program {  
    Polygon poly = new Polygon();  
    poly.pt = new Point[4];  
    Point p = new Point(); p.x = 10; p.y = 20;  
    poly.pt[0] = p;  
    p = new Point(); p.x = 20; p.y = 40;  
    poly.pt[1] = p;  
    p = new Point(); p.x = 60; p.y = 40;  
    poly.pt[2] = p;  
    p = new Point(); p.x = 70; p.y = 20;  
    poly.pt[3] = p;  
    ...  
}
```



Methoden mit mehreren Rückgabewerten



Java-Funktionen haben nur 1 Rückgabewert

Will man mehrere Rückgabewerte, muß man sie zu einer Klasse zusammenfassen

Beispiel: Umrechnung von Sekunden auf Std, Min, Sek

```
class Time {  
    int h, m, s;  
}
```

```
class Program {  
    static Time convert (int sec) {  
        Time t = new Time();  
        t.h = sec / 3600; t.m = (sec % 3600) / 60; t.s = sec % 60;  
        return t;  
    }  
    public static void main (String[] arg) {  
        Time t = convert(10000);  
        Out.println(t.h + ":" + t.m + ":" + t.s);  
    }  
}
```

Kombination von Klassen mit Arrays



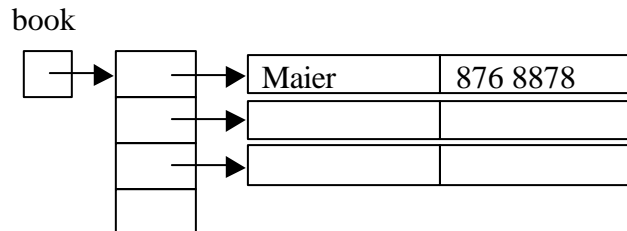
Beispiel: Telefonbuch

	name	phone
0	Maier	876 8878
	Mayr	543 2343
	Meier	656 2332
99		

zweidimensionales Array
kann hier nicht verwendet werden

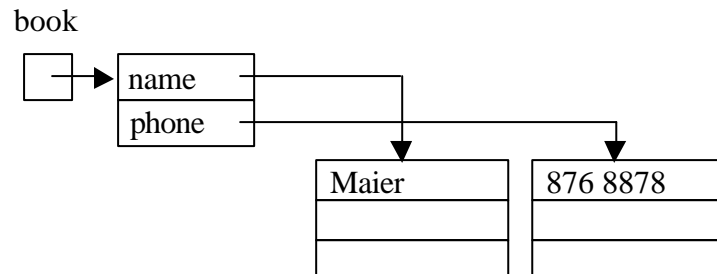
Array von Objekten

```
class Person {  
    String name;  
    int phone;  
}  
Person[] book = new Person[100];
```



Objekt bestehend aus 2 Arrays

```
class PhoneBook {  
    String[] name;  
    int[] phone;  
}  
PhoneBook book = new PhoneBook();  
book.name = new String[100];  
book.phone = new int[100];
```



Implementierung



```
class Person {  
    String name;  
    int phone;  
}
```

```
class PhoneBookSample {  
    static Person[] book;  
    static int nEntries = 0; // current number of entries in book
```

```
    static void enter (String name, int phone) {  
        if (nEntries >= book.length) Out.println("--- phone book full");  
        else {  
            book[nEntries] = new Person();  
            book[nEntries].name = name;  
            book[nEntries].phone = phone;  
            nEntries++;  
        }  
    }  
}
```

```
    static int lookup (String name) {  
        int i = 0;  
        while (i < nEntries && !name.equals(book[i].name)) i++;  
        // i >= nEntries || name.equals(book[i].name)  
        if (i < nEntries) return book[i].phone; else return -1;  
    }  
}
```

Implementierung (Fortsetzung)



```
...
public static void main (String[] arg) {
    book = new Person[1000];
    //----- read the phone book from a file
    In.open("phonebook.txt");
    String name = In.readName();
    int phone;
    while (In.done()) {
        phone = In.readInt();
        enter(name, phone);
        name = In.readName();
    }
    In.close();
    //----- search in the phone book
    for (;;) {
        Out.print("Name: "); name = In.readName();
        if (!In.done()) break;
        phone = lookup(name);
        if (phone >= 0) Out.println("phone number = " + phone);
        else Out.println(name + " unknown");
    }
}
} // end PhoneBookSample
```

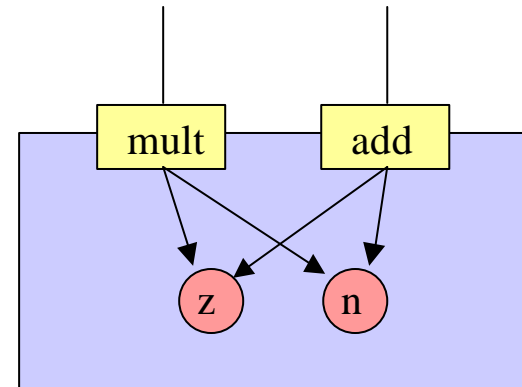
Klasse = Daten + Methoden



Beispiel: Bruchzahlenklasse

```
class Fraction {  
    int z; // Zähler  
    int n; // Nenner  
  
    void mult (Fraction f) {  
        this.z = this.z * f.z;  
        this.n = this.n * f.n;  
    }  
  
    void add (Fraction f) {  
        this.z = this.z * f.n + f.z * this.n;  
        this.n = this.n * f.n;  
    }  
}
```

f1.mult(f2);



abgeschlossener Baustein

- *mult* und *add* sind lokal zu *Fraction* (können auf *Fraction*-Objekte angewendet werden)
- *this* bezeichnet "dieses Objekt", auf das die Operation angewendet wird
- Methoden hier ohne *static* deklariert (siehe später)

Aufruf von Methoden



```
Fraction a = new Fraction(); a.z = 1; a.n = 2;    // a == 1/2  
Fraction b = new Fraction(); b.z = 3; b.n = 5;    // b == 3/5
```

a.mult(b);

Auf das Objekt *a* wird die Operation *mult* angewendet (mit Parameter *b*)

Man sagt:

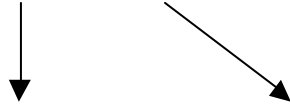
- *a* bekommt die Nachricht (message) *mult*
- *a* ist der Empfänger der Nachricht *mult*

Was passiert dabei?

Aufruf von Methoden



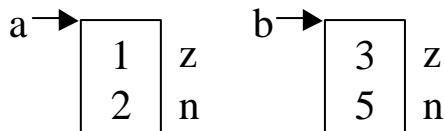
a.mult(b);



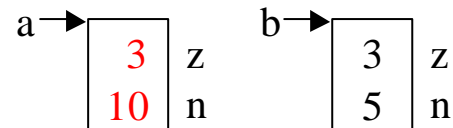
```
void mult (/*Fraction this, */ Fraction f) {  
    this.z = this.z * f.z;  
    this.n = this.n * f.n;  
}
```

Was passiert?

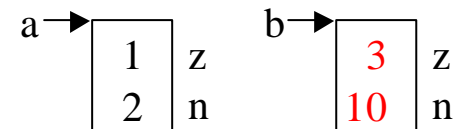
- Parameterübergabe:
this = *a*; (*this* ist ein versteckter Parameter jeder Methode)
f = *b*;
- Es wird die *mult*-Methode der Klasse von *a* aufgerufen



a.mult(b);



b.mult(a);



Weglassen von *this*

```
class Fraction {
  int z, n;

  void mult (Fraction f) {
    z = z * f.z;
    n = n * f.n;
  }

  void add (Fraction n) {
    z = z * n.n + this.n * n.z;
    this.n = this.n * n.n;
  }
}
```

z und *n* sind eindeutig.
Compiler fügt *this* automatisch ein

n wäre nicht eindeutig.
Qualifikation mit *this* nötig

this kann weggelassen werden, wenn der restliche Name eindeutig ist

Grafische Notation für Klassen



UML-Notation (Unified Modeling Language)

Fraction	<i>Klassenname</i>
int z int n	<i>Felder</i>
void mult (Fraction f) void add (Fraction f)	<i>Methoden</i>

Vereinfachte Form

Fraction	<i>falls weniger Details gewünscht oder nötig</i>
z n	
mult(f) add(f)	

Konstruktoren



Spezielle Methoden, die beim Erzeugen eines Objekts automatisch aufgerufen werden

```
class Fraction {  
    int z, n;  
  
    Fraction (int z, int n) {  
        this.z = z; this.n = n;  
    }  
  
    Fraction () {  
        z = 0; n = 1;  
    }  
  
    void mult (Fraction f) {...}  
    void add (Fraction f) {...}  
}
```

- dienen zur Initialisierung eines Objekts
- heißen wie die Klasse
- ohne Funktionstyp und ohne void
- können Parameter haben
- können überladen werden

Aufruf

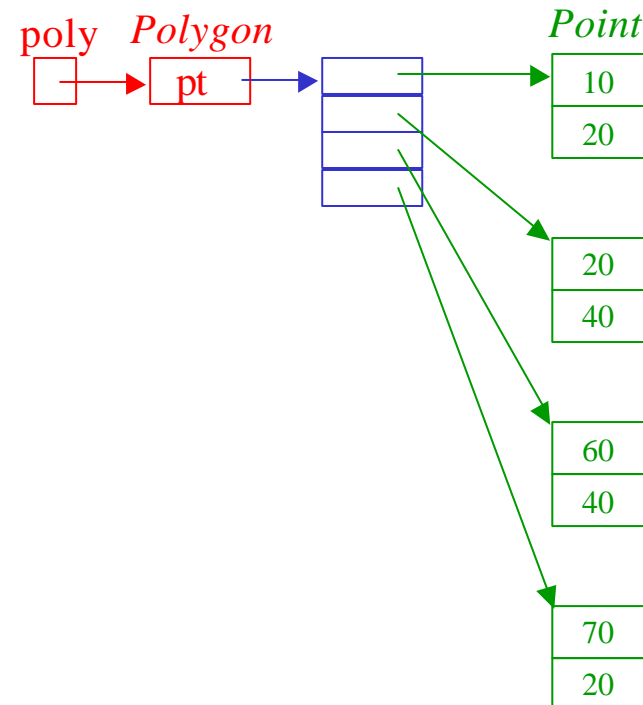
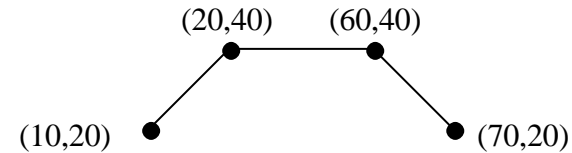
```
Fraction f = new Fraction(3, 5);  
Fraction g = new Fraction();
```

- legt neues *Fraction*-Objekt an
- ruft für dieses Objekt den Konstruktor auf

Bsp: Polygon-Aufbau mit Konstruktoren



```
class Point {  
    int x, y;  
    Point (int x, int y) { this.x = x; this.y = y; }  
}  
class Polygon {  
    Point[] pt;  
    Polygon (Point[] pt) { this.pt = pt; }  
}
```



```
class Program {  
    Polygon poly = new Polygon(  
        new Point[] {  
            new Point(10, 20),  
            new Point(20, 40),  
            new Point(60, 40),  
            new Point(70, 20)  
        }  
    );  
    ...  
}
```

static



```
class Window {  
    int x, y, w, h;           // Objektfelder (in jedem Window-Objekt vorhanden)  
    static int border;       // Klassenfeld (nur einmal pro Klasse vorhanden)  
  
    void redraw () {...}     // Objektmethode (auf Objekte anwendbar)  
    static void setBorder (int n) {border = n;} // Klassenmethode (auf Klasse Window anwendbar)  
  
    Window(int w, int h) {...} // Objektkonstruktor (zur Initialisierung von Objekten)  
    static { ... }           // Klassenkonstruktor (zur Initialisierung der Klasse)  
}
```

Klasse Window

border
setBorder() Klassenkonstruktor

Window-Objekt

x y w h
redraw() Window()

Window-Objekt

x y w h
redraw() Window()

Window-Objekt

x y w h
redraw() Window()

- Objektmethoden haben Zugriff auf Klassenfelder (*redraw* kann auf *border* zugreifen)
- Klassenmethoden haben keinen Zugriff auf Objektfelder (*setBorder* kann nicht auf *x* zugreifen)

static (Forts.)

Was geschieht wann?

Beim Laden der Klasse *Window*

- Klassenfelder werden angelegt (*border*)
- Klassenkonstruktor wird aufgerufen

Beim Erzeugen eines *Window*-Objekts

- Objektfelder werden angelegt (*x, y, w, h*)
- Objektkonstruktor wird aufgerufen

Zugriffe

Zugriff auf static-Elemente über den Klassennamen

- *Window.border* = ...; *Window.setBorder*(3);
- Methoden der Klasse *Window* können Klassennamen weglassen (*border* = ...; *setBorder*(3);)
- Klassenkonstruktor wird nie explizit aufgerufen

Zugriff auf nonstatic-Elemente über einen Objektnamen

- *Window win* = new *Window*(100, 50);
win.x = ...; *win.redraw*();
- Methoden der Klasse *Window* können auf eigene Elemente direkt zugreifen (*x* = ...; *redraw*();)

Bitoperationen

- $\&$ bitweises Und $10 \& 3 = 2$ ($1010 \& 0011 = 0010$)
- $|$ bitweises Oder $10 | 3 = 11$ ($1010 | 0011 = 1011$)
- \sim bitw. Negation ~ 10 ($\sim 1010 = 0101$)

– Gelegentlich benutzt:

- \wedge bitw. Exkl. Oder $10 \wedge 3 = 9$ ($1010 \wedge 0011 = 1001$)

– Übersicht

• a	b	a&b	a b	~a	a^b
• 0	0	0	0	1	0
• 0	1	0	1	1	1
• 1	0	0	1	0	1
• 1	1	1	1	0	0

Aufgabe

- Implementieren Sie eine Klasse Set, die Mengen von Zahlen $[0;31]$ speichert. Nutzen Sie dazu ein normales 32bit int, dessen Bits 0..31 jeweils genau dann 1 sind, wenn die Zahl Element der Menge ist. Implementieren Sie die Operationen
 - void insert (int), //Element einfügen
 - bool contains (int), //ist ein Element im Set?
 - void delete (int), //Element löschen
 - void union (Set) und //Vereinigungsmenge
 - void intersect (Set). //Schnittmenge(Dafür verwenden Sie bitweises $\&$, bitweises $|$, bitweise Negation \sim und Bitshifts \gg und \ll .)