Wiederholung und Grundwissen für Q11

Darstellung im Klassendiagramm bzw. Objektdiagramm

Klassenkarte

RECHTECK
Oben Links Breite Höhe Füllfarbe Randfarbe ...
Drehen(Winkel) Verschieben(dx,dy) ...

Objektkarte

Tisch1:RECHTECK
Breite = 50 Höhe = 20 Oben = 100 Links = 75 Randfarbe = rot

Objektorientierung

Objektorientierung beschreibt das Lösen von Problemstellungen durch Analyse der beteiligten Objekte und deren Zusammenspiel.

Punktnotation Methodenaufrufs ohne oder mit Parameter

```
Objektname.Methode(Parameter);
Objektname.Attribut = Attributswert;
```

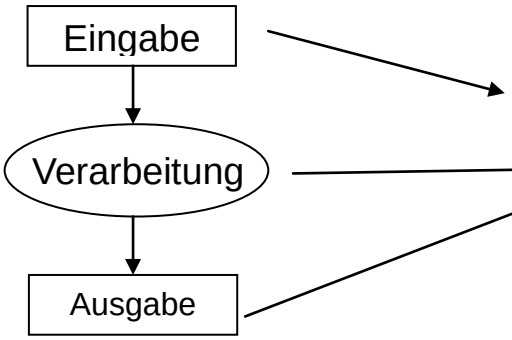
Beispiel:

```
Tisch1.Breite = 50;
Tisch1.verschieben(10,10);
Tisch1.Löschen( );
Karol.Schritt();
Karol.Hinlegen();
Karol.Linksdrehen();
```

Datentypen

Datentyp	Beschreibung	Werte	Speicherbedarf
int	ganze Zahl	-2.147.483.648 bis 2.147.483.647	4 Bytes
long	ganze Zahl	-2^{63} bis $2^{63}-1$	8 Bytes
float	Dezimalzahl	±1,4E-45 bis ±3,4E+38	4 Bytes
double	Dezimalzahl	±4,9E-324 bis ±1,7E+308	8 Bytes
char	ein Zeichen	0 bis 65.535	2 Byte
boolean	Wahrheitswert	true / false	1 Bit
String	Zeichenfolge	“Zeichenfolge variabler länge …“	2 Byte*Zeichen+Overhead

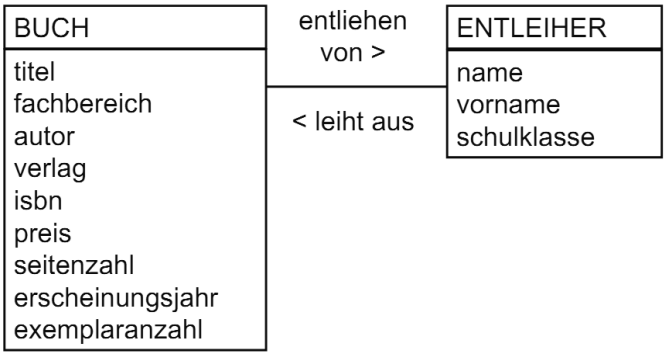
E-V-A Prinzip




```
public bmi(float gewicht, float groesse)
{
  float bmi = gewicht/(groesse^2);
  System.out.println("BMI "+bmi);
}
```

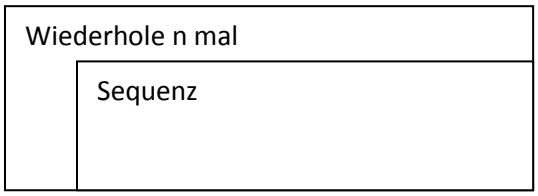
Klassen-Beziehungsdiagramm

Das Klassen-Beziehungsdiagramm ist ein Strukturdiagramm zur grafischen Darstellung (**Modellierung**) von Klassen mit ihren Attributen und Methoden sowie deren Beziehungen. Das Ziel ist die Verringerung der Komplexität der entstehenden Programme.



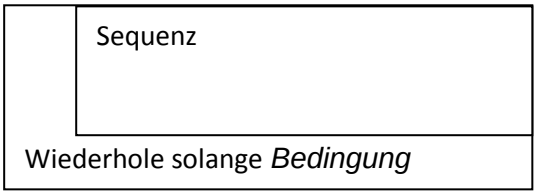
Kontrollstrukturen

Wiederholung mit fester Anzahl



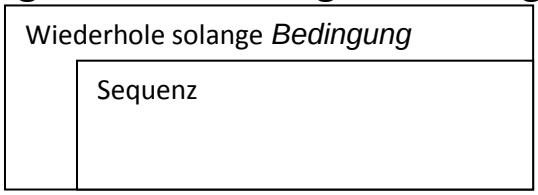

```
for(int i=1; i<=n; i++)
{
    Sequenz;
}
```

bedingte Wiederholung mit Endbedingung



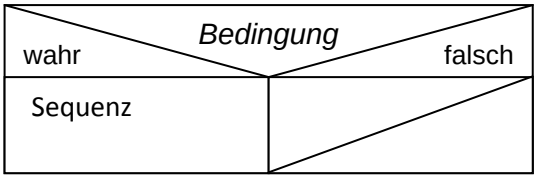

```
do
{
    Sequenz;
} while(Bedingung);
```

bedingte Wiederholung mit Anfangsbedingung




```
while(Bedingung)
{
    Sequenz;
}
```

Bedingte Anweisung (1-seitig)




```
if (Bedingung)
{
    Sequenz;
}
```

Bedingte Anweisung (2-seitig)

Bedingung: wahr	Bedingung: falsch
Sequenz1	Sequenz2

```
if (Bedingung)
{
    Sequenz1;
}
else
{
    Sequenz2;
}
```

Bedingungen (Beispiele)

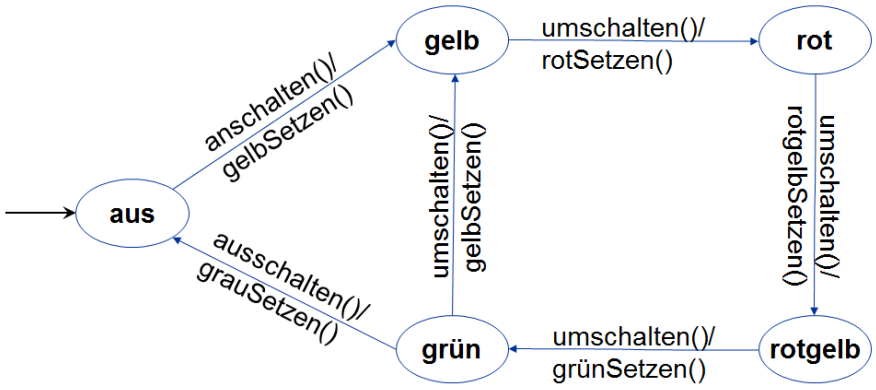
```
x > 0
b == 5
c != 20
(a > 5 && a < 10)
(a > 6 || a <-6)
karol.IstWand()
!karol.IstZiegel()
```

Mehrfachauswahl

Variable = ?				
1	**2**	**3**	**…**	**sonst**
Sequenz1	Sequenz2	Sequenz3	…	Sequenz

```
switch (Variable)
{
case 1:
    Sequenz1;
    break;
case 2:
    Sequenz2;
    break;
case 3:
    Sequenz3;
    break;
...
default:
    Sequenz;
    break;
}
```

Zustandsdiagramme

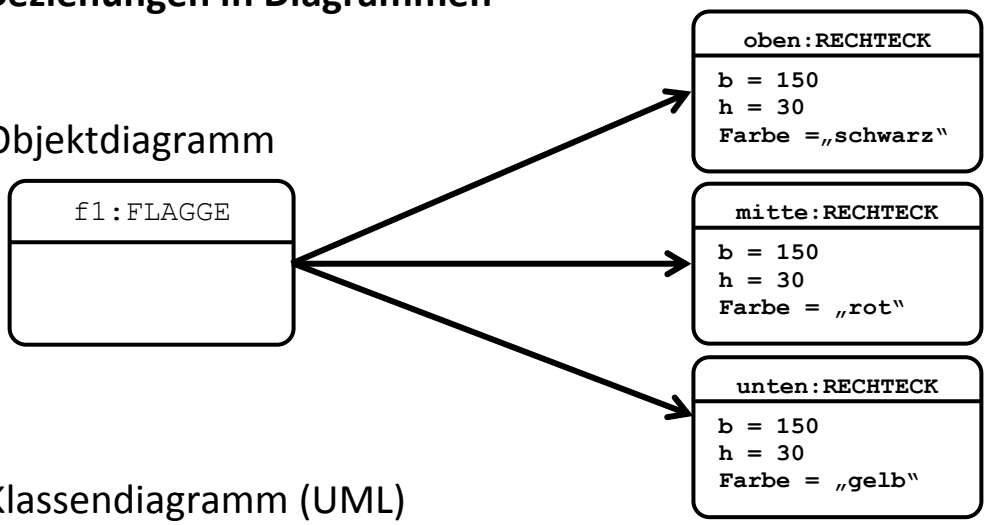


Schreibweise bei Zustandsdiagrammen: Ereignis [Bedingung] /Aktion

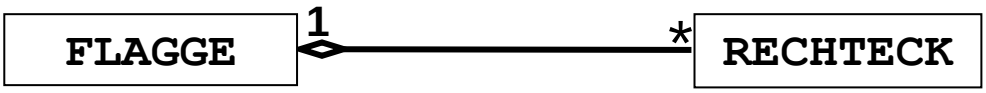
Viele **Informatiksysteme** (**IS**) wie z.B. Mobilfunktelefone, Waschmaschinen, Aufzüge oder Steuerungen können durch **endliche Automaten** (**EA**) beschrieben werden.

Beziehungen in Diagrammen

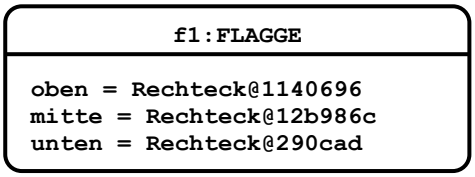
Objektdiagramm



Klassendiagramm (UML)



Die **Aggregation** („**Enthält-Beziehung**“) wird im enthaltenen Objekt als Attribut implementiert. Dessen Wert ist eine **Referenz** (Verweis auf den Speicherplatz) auf das enthaltene Objekt.
UML : Unify Modelling Language Beispiel: Flagge



Arten von Beziehungen

Kardinalität		Multiplizität
1 : 1	oder	1 .. 1
1 : n		1 .. *
n : m		* .. *

Feld (Array)

- Zusammenfassung mehrerer Variablen oder Objekte der gleichen Klasse in einer **zusammenhängenden Folge fester Länge**.
- Die einzelnen Feldelemente sind durchnummeriert. Diese Nummern nennt man **Index**.
- Zugriff auf einzelne Elemente über Indizes (Nummerierung beginnt bei 0).

```
int zahl[];
zahl = new int[1000];
zahl[5] = 10;
```

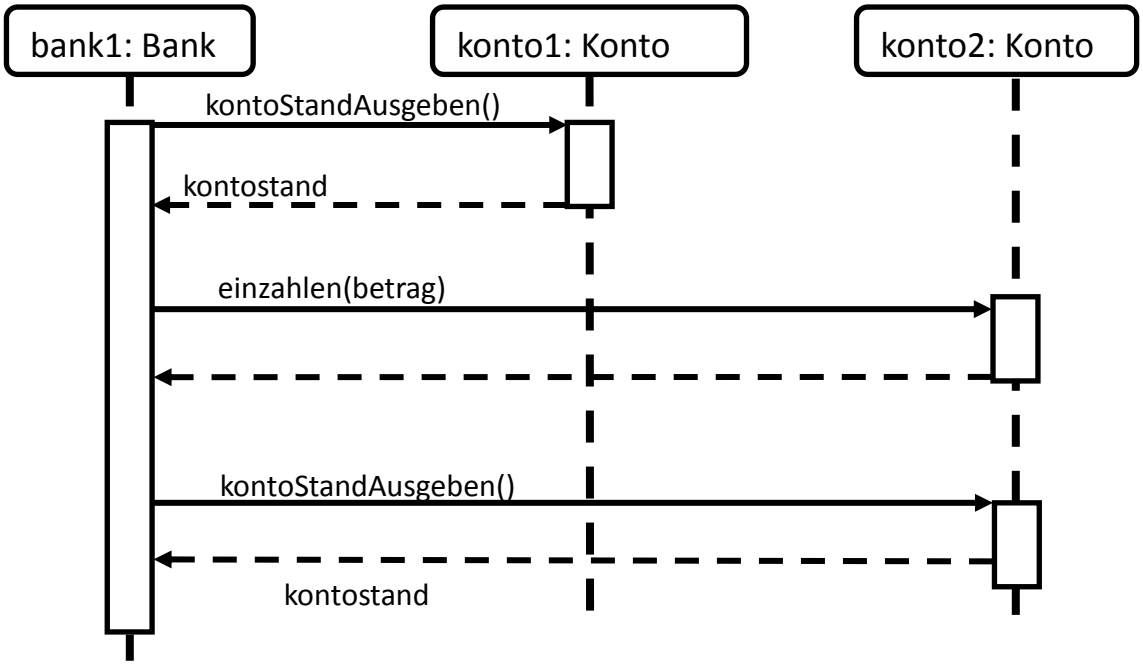
Deklaration
Initialisierung
Wertzuweisung

Hinweis: Instanziierung bedeutet die Erzeugung eines neuen Objekts (Anwendung des Konstruktors)

Kommunikation zwischen Objekten

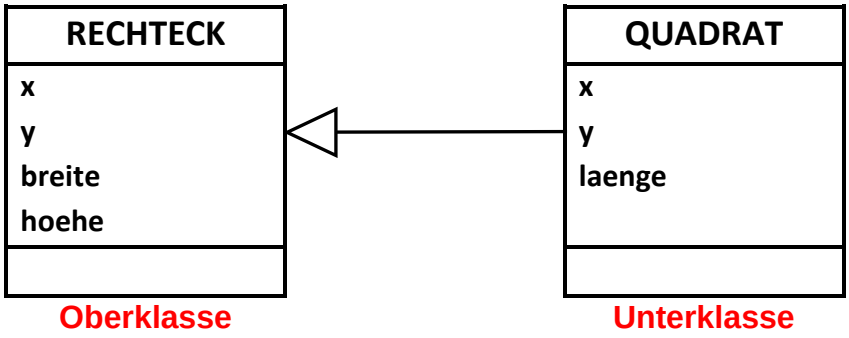
Objekte können mit einander kommunizieren, durch…
…lesen/verändern der öffentliche Attributwerte eines anderen Objektes lesen
…Aufruf der öffentliche Methoden eines anderen Objektes. (zu bevorzugen!!)
Hinweis: Attribute sollten nicht öffentlich sein (**Datenkapselung**), **Getter**- & **Setter**-Methoden verwenden.

Sequenzdiagramme



Vererbung

Klassenbeziehungen sehr „ähnlicher“ Klassen
→ bei hierarchischen Strukturen (**Vererbung**)
→ Ziel: Codevereinfachung




```
public class QUADRAT extends RECHTECK {
  // …
  public QUADRAT(int x, int y, int laenge) {
     super(x, y, laenge, laenge);
  //…
  }
}
```

super	Aufruf des Konstruktors der Oberklasse.
extends	Gibt die Oberklasse an, nach der die Unterklasse erstellt wird.
protected	Attribute der Oberklasse werden **protected** (statt **private**), damit sie vererbt werden können

Hinweis: Methoden der Oberklasse können in der Unterklasse überschrieben werden (**Polymorphismus**).