Informatik 1

Vorlesungsmitschrift

geschrieben von

Tihomir R. Todorov

SS 2010, Prof. Dr. Pape

Inhaltsverzeichnis

18.03.2010 (1 Vorlesung) Fehlt!

22.03.2010 (2 Vorlesung)

Variablen (zum Speichern von Rechnerergebnissen):
- Datentyp
- Name

müssen vor Verwendung deklariert werden.

Deklaration:

```
Datentyp    Name;              // ; (semicolon) schließt Anweisung ab. Dekl.
Datentyp    Name = Ausdruck;   // Deklaration + Initialisierung.
```

Anwendung:

```
int         wochentag;         // ; (semicolon) schließt Anweisung ab. Dekl.
int         monat = 3;         // Deklaration + Initialisierung.
```

Primitive Datentypen:

byte, short, int, long	float, double	boolean	char
2er-Komplement	Gleitkommazahl	Boolean	Character
Bsp.: -1, 123, 34, 1444	Bsp.: 20.70, 13.1	Bsp.: true, false	Bsp.: 'a'

Range:

byte: -128 bis 127	JVM
short: -32768 (2^{15}) bis 32767 (2^{15}-1)	2 bytes
int: ~-2Mio bis 2Mio	
long: -2^{63} bis 2^{63} - 1 (~10^{11})	8 bytes

Verwenden Sie: int, long Vergessen Sie: short
double statt float (in C double ist schneller als float!)

Beispielen:

```
int         wochentag;          // Wert 0
int         monat = 3;          //
boolean     sonning = true;     //
boolean     wolkig;             // false
char        c = 'a';            // unter Unicode
char        c;                  // nul = 0
double      pi = 3.1415;        //
long        l = 1L;             // L, damit der Compiler weißt, dass ein Long-Wert ist.
long        minusSieben = -7l;  // l, damit …
float       e = 2.7f;           // f, damit der Compiler weißt, dass ein Float-Wert ist,
                                // sonst ist immer ein Double-Wert.
double      hundert = 10.0e2;   // 1000 = 1*10^3
double      einhalb = 50E-2;    //      = 10*10^2
                                // Note: Wissenschaftliche Notation (obere 2 Bsp.) nur
                                // bei sehr großen oder sehr kleinen Zahl verwenden.
char        dal = '\u062F';     // 062F sind 4 Ziffern
char        schraegstrich = '\\'; //
char        neueZeile = '\n';   // new line
char        tabulator = '\t';   // Tabulator
```

Beispiel (Temperaturumrechnung):

Formel: `f = c * 1.8 + 32` // where 1.8 = 9/5

Zuweisung: `Variable = Ausdruck;` // Ausdruck wird ausgewertet und dann in // die Speichervariable geschrieben.

Source (Temperaturumrechnung):

```
public class CelciusInFarenheit {
	public static void main(String [] argv) {
		double celsius = 23.0;
		double fahrenheit;

		fahrenheit = celsius * (9.0 / 5.0) + 32;
		System.out.println(fahrenheit);
	}
}
```

Operatoren:
+, -, *, /, % (Gelten für ganze Zahlen sowie Gleitkommazahlen)

Mathematik:	vs.	Java:
7 : 5 = 1 Rest 2		7 / 5 = 1
		7 % 5 = 2

Falls eine Seite einer Operation im Ergebnis double (float), dann ist die Operation eine double-Operation (float).
Konvention: Möglichst gleiche Datentypen in Teil eines Ausdrucks verwenden.

Note: Tutorien, Rechnerübungen ab 29. März:

Beispiel:

```
byte b = 127;
b = b + 1;      // Überlauf, d.h. es wird einfach binär weiter gerechnet.
                // 0111 1111 + 0000 0001 -> 1000 0000 im Techn. Info. = -1
                // (-1 ist im 2er-komplement)
b = b / 0;      // Programm bricht mit Fehler ab (Schwerwiegender Fehler)!
```

Note: (float/double) Gleitkommazahlen:

Formel: $(-1)^{[\text{Vorzeichnis(s)}]} * \text{Mantisse(m)} * \text{Basis(b)}^{[+\text{-Exponent(e)}]}$

0.5 + 1 - 1 $+1*2^{-1} = 0.5$
0.1 hat periodische Darstellung binär => nicht genau als Gleitkommazahl darstellbar.

Fazit:
Rechenergebnisse bei Gleitkommaoperatoren sind (meist) mathematisch nicht genau!

25.03.2010 (3 Vorlesung)

Tutorien (nächste Woche):
Mo: 14:00 E303
Di: 9:50 LI137

Widening Conversion (von klein zu groß)
Konvertierung Datentypen geringer Genauigkeit zu Datentypen höherer Genauigkeit (Compiler) bei Ausdrücken, Zuweisungen, Initialisierungen

byte < short < int < long < float < double

Bsp.:
```
long l = 17;
int i = 17L;                    // Compiler meldet Fehler.
float f = 2147483642;           // 5 weniger als max. int Wert.
```

Konvertierung
int -> float
long -> float, double
kann ungenauer sein.

Narrowing Conversion (von groß zu klein)
Konvertierung von Datentype höherer Genauigkeit zu Datentypen geringer Genauigkeit.
cast-Operator vor Zahl-Ausdrück nötig.

Bsp.:
```
int i = (int) 17.5
```

1. Nachkommaanteil wird abgeschnitten
2. Ganzzahliger Anteil wird in int (oder long) konvertiert. Falls Zahl zu groß für int (oder long) ist, dann wird maximale int (long) Wert genommen.
3. Falls Zieldatentyp int (long) war, dann ist Wert aus 2. das Ergebnis.
4. Falls Zieldatentyp byte, short, char ist, dann werden die unteren 8, 16 Bits aus Wert von Schritt 2 genommen.

```
byte b = (byte) 255.552;        // -1.
int i = (int) 500 000 000L;     // i ist maximaler int Wert.
```

Weiter mit Operatoren
Zuweisungsoperator: =
Linke Seite muss eine Variable sein, rechte Seite muss ein Typkompatibler Ausdruck Sein (in der Regel gleiche Datentyp wie die Variable).

Ergebnis der Zuweisung ist Wert der rechten Seite.

Bsp.:
```
int a = 1;
    a = 2;
    a = (a = 1);
    a = (a = a + 1) - 1;
int b = 2;
```

```
a = b = 2;   <=>   a = (b = 2);          // Wird von rechts nach links ausgewertet.
```

Varianten von =

```
A Op = B          Op = { +, -, *, /, %, . . . }
```

Bsp.:

```
a += 1;       <=>   a = a + 1;          // A = (A) Op (B)
double d = 1 - 0;
d *= 7.0;     <=>   d = d * (7.0);
```

Inkrement, Dekrement

```
a++;   <=>   a = a + 1;
a--;   <=>   a = a - 1;
```

	a++	++a	a--	--a
Ergebnis	A	a + 1	a	a - 1

```
int a = 6;
a += (a -= 5);       <=>   a = a + (a = (a - 5))          // Lösung: 7
```

Seiteneffekte:

Zustandsänderungen von Programmen Werteänderungen einer Variablen ist ein Seiteneffekt.

Konvention: Seiteneffekte in Ausdrücken vermeiden!

Boolesche Ausdrücke

boolean

Literales true, false

&	Und
|	Oder
^	Entweder oder(XOR)
!	Negation

Bsp.:

```
boolean esRegnet = true;
boolean fensterSindOffen = true;
boolean wohnungIstNass = esRegnet & fensterSindOffen;

boolean bodenFrischGewisch = false;
boolean rutschGefahr = (esRegnet & fensterSindOffen) | bodenFrischGewisch;

boolean sonnigesWetter = !esRegnet;
```

Operatorenbindung

& stärker als ^, ^ stärker als |

! stärker als &, ^, |

Relationaloperatoren (für alle Datentypen)

= = (Identitätsoperator) != Bsp.: a != b <=> !(a = = b)

Bsp.:
1 = = 2 -> false
1 = = (2 - 1) -> true
1 != 2 -> true
true = = false -> false

Vergleichsoperatoren (nur für Zahltypen)
>, <, >=, <= Bsp.: 7 > 5 -> true, 5 <= -2 -> false

Übung:
double a = 1.0;
double b = 0.0;
10 mal 0.1 auf b addieren
a == b -> false (b nicht genau 1.0)

Gleitkommazahlen Nie mit = = direkt vergleichen!
|a - b| < 0.000000001
Java: Math.abs(a - b) < 0.000000001 (hängt von Anwendungszweck ab)

Zeichenketten
"Dies ist eine Zeichenkette"
Datentyp String

29.03.2010 (4 Vorlesung)

Tutorien
Mo 14:00 UE02
Mo 14:00 E303
Di 9:50 LI137

Boolesche Ausdrücke
&, ^, |, !
&&, || -> Kurzschlussoperatoren
true && false -> false
true || false -> true

Bei binären Operatoren (+, *, &. |.) wird erst linke Seite des Ausdrucks ausgewertet, dann die rechte Seite.

Bei && und ||| wird rechte Seite nur ausgeweführt, wenn Ergebnis noch nicht feststeht.

boolean b = true;	b && a	(rechte Seite wird ausgeführt)
boolean a;	(!b) && a	(rechte Seite wird nicht ausgewertet, wenn (!b) = false)
	(!b) || a	(rechte Seite wird ausgeführt)
	b | (!a)	(rechte Seite wird nicht ausgeführt, wenn b = true)

Operatorenbindung

! bindet am stärksten	!a & b	<==>	(!a) & b
& bindet stärker als ^	a | ((b & (!a)) ^ (a = = b))		
^ bindet stärker als |	(a | b) && (!a)	Erweiterung währ	a | (b && (!a))

Konvention:
- &, | nicht mit &&, || in einem Ausdruck mischen.
- Ausdrücke möglichst vollständig klammern.
- Nicht über sichtbares Zeilenende hinaus schreiben (Umbruch)

Formatierung vom Ausdrücken
- Vor und nach einem (binären) Operator ein Leerzeichen setzen (Bsp.: a | b)
- Vor einem Operator mit schwacher Bindung (Klammern) umbrechen
- Einrächen bis unter zugehörigen linken Zeilenausdruck

Bsp.:
int a, b; a * a * a + 3 * a * a * b + 3 * a * b * b
+ b * b * b

Kontrollanweisungen
steurn Kontrollfluß eines Programms

Einfachauswahl
Bei Kauf von Alkohl: ab 16 Bier/Wein
ab 18 alles
nicht zwischen 22:00 – 05:00 Uhr (BaWü)

Bsp.:

```
int  alter;
int  uhrzeit;          // angetangene volle Stunde
boolean  alkohol;      // kauft (irgendwas) alkoholische Getränke
boolean  bier          // darf ab 16 verkauft werden.
```

```
(bier | alkohol) & !(22 <= uhrzeit & uhrzeit < 5) & (alter >= 18 | bier & alter >= 16)
^ !(bier | alkohl))
```

```
if ((bier | alkohl) & ((!bier & alter < 16) ^ alter < 18)) {
        System.out.println("ALARM");
        System.out.println("Person festhalten");
}
```

1. Boolesche Ausdruck wird ausgewertet
2. Anweisungsblock wird ausgeführt, falls Ausdruck true ergab
3. Ansonsten: if-Anweisung ist beendet.

if (B) {
A1;
A2;
…
An;
}
	// Wenn nur ein Ausdruck ist, es ist möglich
	// auch ohne { }.
	if (B)
	A1;

Empfehlung:
Immer { } bei allen Kontrollanweisungen verwenden.

<u>Zweifachauswahl</u>

```
if (B) {
        // Wird ausgeführt, falls B true ist.
} else {
        // Wird ausgeführt, falls B false ist.
}
```

<u>Code Style:</u>

if (B) {
// …
}
	if (B)
	{
	// …
	}
		if (B)
		{
		// …
		}

Nach {, } kein Anweisung schreiben
Pro Zeile max. eine Anweisung
Bei (a), (b) Anweisungen einrücken.

If (B1) {
// …
} else {
if (B2) {
// …
} else {
if (B3) {
// …
}
}
}
	if (B1) {
	// …
	} else if (B2) {
	// …
	} else if (B3) {
	} else {
	// …
	}
	
	Ausnahme!!!

01.04.2010 (5 Vorlesung)

Gegeben: 3 Zahlen a,b,c	Gesucht: Sortierung von a, b, c a <= b <= c (mit gegebene Werte)

1. Vertauschen von Werten.
2. Zwei Werte sortieren.
3. Drei Werte sortieren.

1. Vertauschen von Werten (Java-Source).

```
int a, b, tmp;
tmp = a;
a = b;
b = tmp;
```

2. Zwei Werte Sortieren (Java-Source).

```
int a, b;
if (a < b) {
        // Do nothing.
} else {
        int tmp = a;
        a = b;
        b = tmp;
}
```

3. Drei Werte Sortieren (Überlegung).

a	b	c
1	2	3
2	1	3
1	3	2
2	3	1
3	1	2
3	2	1

3. Drei Werte Sortieren (Java-Source).

```
if (a > b) {
        // Vertausche a und b
} if (b > c)
        // vertausche b mit c
}
// => max. ist in c
if (a > b) {
}
```

while-Schleife (Kopfgesteuerte-Schleife, da die Bedingung am Anfang ist.)

```
while (boolescher Ausdruck) {
        // Do something …
}
```

1. Boolescher-Ausdruck wird ausgewertet.
2. Falls Ergebnis false ist, dann wird das while abgebrochen.
3. Falls Ergebnis true ist, dann wird der Block ausgeführt und danach geht es weiter mit 1.

Beispiel:

Zahlen von 1…10 ausgeben

```
int i = 1;
while (i < 10) {
        System.out.println(i);
        i++;
}
```

Eulerscher Algorithmus (für ggT - den **g**rößten **g**emeinsamen **T**eiler)
Solange zwei Zahlen verschieden sind, ziehe die kleinere von der größeren ab.

Überlegung:	Java-Source:	

a | b
---------|----------
21 | 9
2 | 9
3 | 9
3 | 6
3 | 3
---------|----------
	int a = 21, b = 9;
	while (a != b) {
	  if (a > b) {
	  a = a - b;
	  } else {
	  b = b - a;
	  }
	}
		ggT (a, b) = ggT(a, b-a) b >a
		ggT(a-b, b) a > b
		a

Testen:
Anhand von Beispieldaten, die Funktionsweise eines Programms überprüfen.

Korrektheit eines Programms (Algorithmus).
Programm liefert zu jeder beschriebenen Eingabe (gegeben) immer das gewünschte Ergebnis (gesucht).

Terminieren
Algorithmus endet bei den in gegeben beschriebenen Eingaben.

do-while-Schleife

do {
  // Do something …
} while (Boolescher-Ausdruck);
	1. Anweisung des Blocks werden ausgeführt.
	2. Boolescher-Ausdruck wird ausgewertet.
	3. Falls true: zurück zu 1. ansonsten ist Schleife beendet.

Beispiel
Durch 7 die teilbare Zufallszahl finden.
In Java: Math.random() gibt Zufallszahl zurück zwischen 0 und 1 (double).

do-while	gegen	while (Nachteil: Redondants)

int z;
do {
  z = (int) (Math.random()*100.0);
} while (z % 7 != 0);
		int z = (int) (Math.random()*100.0);
		while (z % 7 != 0) {
		  z = (int) (Math.random()*100.0);
		}

08.04.2010 (6 Vorlesung)

for-Schleife

selbst nachlesen …

Konvention:

- deklarierte Variablen der Initialisierung sollen nur im Anweisungskeit geändert werden
- nur im Zusammenhang mit Lokalvariablen verwenden

for (; ;) { } }	for (; true;) { } }	Endlose-Schlafen Vermeiden!

Primzahl-Beispiel 1:

Gegeben: Positive ganze Zahl z.
Gesucht: Ist z eine Primzahl (nur durch 1 oder z teilbar)?
Verzahlen: Alle potentiellen Teiler von 2 bis z-1 ausprobieren.

Primzahl-Beispiel 1 (Source):

```
public class Primzahl {
        public static void main(String[] args) {
                int z = 41;     // int z = 101 * 41 * 41;
                                // int z = Integer.MAX_VALUE – 1;
                boolean primzahl = true;

                // for (int teiler = 2; teiler < z & primzahl; teiler++) {
                for (int teiler = 2; teiler < z; teiler++) {
                        if (z % teiler == 0) {
                                primzahl = false;
                        }
                }

                if (primzahl) {
                        System.out.println(z + “ist eine Primzahl.“);
                } else {;
                        System.out.println(z + “ist keine Primzahl.“)
                }
        }
}
```

Schlüsselwort break:

- bricht die umschlossene Kontrollanweisung (außer if-else) ab.
- Vermeiden break bei Schleifen (Unübersichtlich)

Primzahl-Beispiel 2:

Gegeben: Positive Zahl z.
Gesucht: Nächste Primzahl nach (und einschließlich) z.

Primzahl-Beispiel 2 (Source):

```
public class Primzahl {
        public static void main(String[] args) {
                int z = Integer.MAX_VALUE – 1;
                boolean primzahl = true;

                primzahlSuche: for ( ; ; ) {
                        for (int teiler = 2; ; teiler++;) {
                                if (teiler = = z) {
                                        break primzahlSuche;
                                } else if (z % teiler = = 0) {
                                        z++;
                                        break;
                                }
                        }
                }

                System.out.println(z);
        }
}
```

switch-Anweisung (für eine Mehrfachauswahl)

```
switch (Arithmetischer Ausdruck) {
        case Konstanter Ausdruck:    Anweisung_1;
                                     Anweisung_2;
                                     Anweisung_3;
                                        ...
                                     Anweisung_N;
        case Konstanter Ausdruck:    Anweisung_N+1;
                                        ...
                                     Anweisung_M;
}
```

```
int i = 2;
switch (i) {
        case 0:     System.out.println(“0“);
        case 2:     System.out.println(“2“);
                    break;
        case 1:     System.out.println(“1“);
        case 3:     System.out.println(“3“);
}
```

i	Ausgabe
0	“0“
	“2“
1	“1“
	“3“
2	“2“
3	“3“
7	Keine Ausgabe

Beispiel:

```
switch (i) {
        case 7:     System.out.println(”7”);
                    break;
        default:    System.out.println(”X”);
}
```

Jeder Fall höchstens einmal + höchstens ein default.

Methoden (Klassenmethoden)

- Fasst Anweisungen unter einen pro Klasse eindeutigen Namen zusammen.
- Kann Parameter zur Wertenübergaben besitzen (Bsp.: “Math.abs(d);“, wobei d Parameter ist).
- Parametername eindeutig pro Methode.
- Parameter verhalten sich wie lokale Variablen.
- return bricht Methode ab.
- Bei Funktionen muss return ein Ausdruck folgen, der Typkompatibel zum deklarierter Typ der Funktion.

Primzahl-Beispiel 3 mit Methode (Source):

```
package prime_number;

class Primnumber {
	// static Method:
	// Use always the class name to call a class method / a static method!
	static boolean isPrimNumber(int nr) {
		for (int factor = 2; factor < nr; factor++) {
			if (nr % factor == 0) {
				return false;
			}
		}

		return true;
	}

	public static void main(String[] args) {
		for (int nr = 2; nr <= 100; nr++) {
			if (Primnumber.isPrimNumber(nr)) {
				System.out.println(nr);
			}
		}
	}
}
```

Aufruf eine Klassenmethode von einer anderen Klasse:

```
package prime_number;

class PrimnumberTest {

	public static void main(String[] args) {
		for (int nr = 2; nr <= 100; nr++) {
			if (Primnumber.isPrimNumber(nr)) {
				System.out.println(nr);
			}
		}
	}

}
```

12.04.2010 (7 Vorlesung)

- Klassen-Methoden (mit Schlüsselwort static)
- Normalen Methoden
- Methoden mit Rückgabewert.

Beispiel:

```
public class PrimzahlTest {
        public static void main(String[] args) {
                for (int z = 1; z <= 100; z++) {
                        if (Primzahl.isPrimzahl(z)) {
                                System.out.println(z);
                        }
                }
                return primzahl;
        }

        public static void berechnePrimzahlen(int von, int bis) {
                for (int z = von; z <= bis; z++) {
                        if (Primzahl.istPrimzahl(z)) {
                                System.out.println(z)
                        }
                }
        }
}
```

Konvention:

Immer die Klassenname beim Aufruf eine Klassenmethode / Variable verwenden.

Aufrufreihenfolge bei der berechnePrimzahlen(int, int)-Klassenmethode:

1. Parameter werden von links nach rechts ausgewertet. (Es geht nach Strengregeln).
2. Dann wird zu der Methode-Implementierung "gesprungen".
3. Parameter werden im Speicher angelegt und werden mit den Werte aus [1] belegt.
4. Methode bricht ab, falls:
    a. ein return ausgeführt wird,
    b. letzte Anweisung der Methode ausgeführt wurde.
5. Bei Funktionen: Wert des Ausdrucks von return wird an die aufrufende Stelle zurückgegeben
6. Nach Ende einer Methode existieren Parameter und lokale Variablen nicht mehr

Methodennamen sind immer eindeutig (pro Klasse) + unterschiedliche Datentypen / Anzahl Parameter!

Polymorphie (Methoden mit der gleichen Namen, aber mit unterschiedlicher Parameter):
Bei Methoden gleichen Namens wird die "best" passende genommen.

1. Anzahl Parameter muss gleich sein.
2. Widening conversions werden berücksichtigt.

Erde-Beispiel:

Erdumfang (Äquator, Pol)
Bahngeschwindigkeit

Erde-Beispiel (Source):

```
public class Erde {
        public static double bahngeschwindigkeit = 29.703; // in m/s
        public static final double UMFANG_AEQUATOR = 40075.003;
        public static final double UMFANG_POL = 39949.630; // in km

        public static void main(String[] argv) {
                System.out.println("Durchschnittliche Bahngeschwindigkiet: "
                                + Erde.bahngeschwindigkeit);


                Erde.bahngeschwindigkeit += 0.1;
                System.out.println("Durchschnittliche Bahngeschwindigkiet: "
                                + Erde.bahngeschwindigkeit);
        }

        public static void bahngeschwindigkeitAendern() {
                Erde.bahngeschwindigkeit = Erde.bahngeschwindigkeit
                                + Erde.bahngeschwindigkeit
                                + Erde.bahngeschwindigkeit + 0.1;
        }
}
```

Schlüsselwort **static**:

Klassenvariablen (Bsp.: bahngeschwindigkeit):
Klassenmethoden (Bsp.: bahngeschwindigkeitAendern())

Schlüsselwort **final**:

Nur bei Initialisierung kann der Wert der Variablen angegeben werde.
Variable kann nicht mit Zuweisung geändert werden
Geht auch bei Parametern, lokale Variablen, etc …
Bei "**static final**":

- Großbuchstaben verwenden
- _: für Teilwörter.
- Bsp.: UMFANG_AEQUATOR, UMFANG_POL

Mischen:

Es ist möglich Klassenvariablen mit Methodenvariablen mit den gleichen Namen zu mischen.

Initialisierung von Klassenvariablen

- Jede Klassenvariable bekommt ihren default-Wert.
- Von oben nach unten werden Klassenvariablen initialisiert.
- Wird dabei auf eine Klasse zugegriffen, die noch nicht im Zustand der Initialisierung ist, dann wird sie vollständig initialisiert.

Beispiel (zwei voneinander abhängige Klassen):

```
public class A {
        public static int x = 1;
        public static int y = B.u;
}
```

```
public class B {
        public static int u = A.x;
}
```

=> Problem bei der Initialisierung.

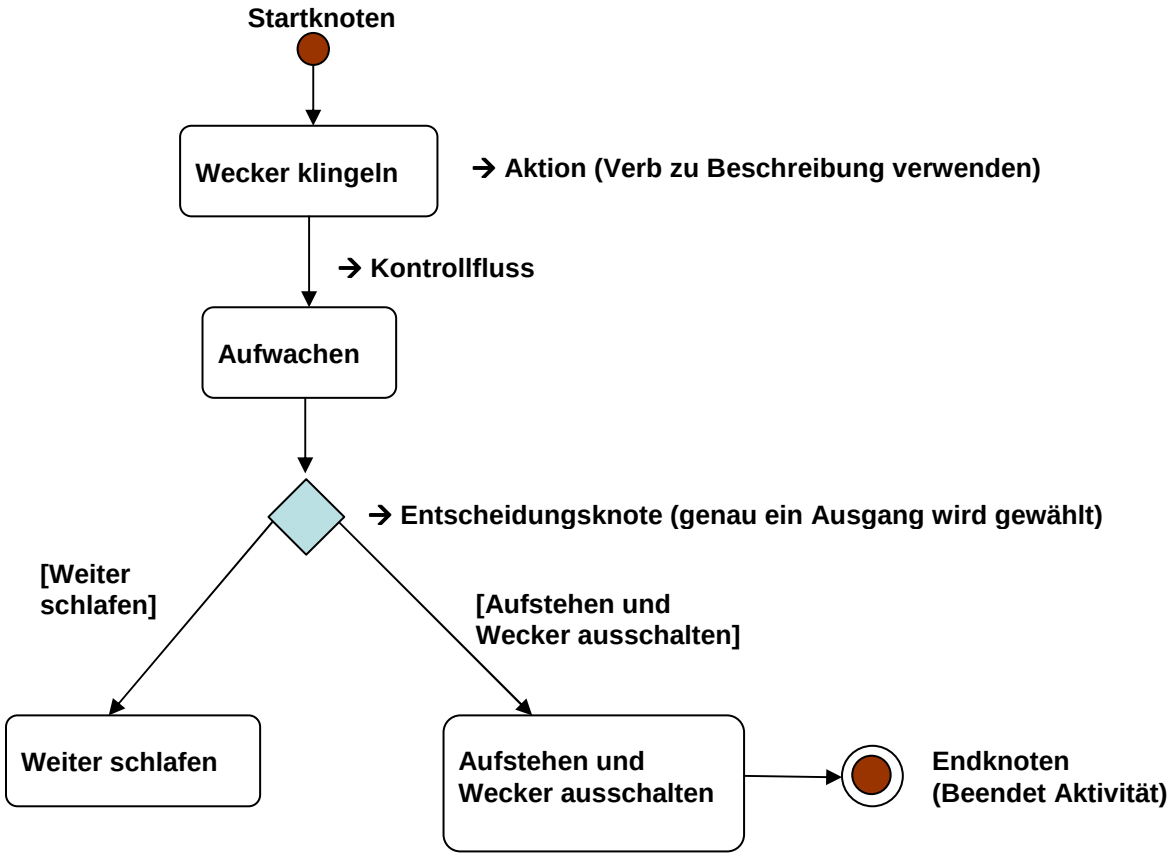
Fazit:

Initialisierung zweier Klassen nicht voneinander (beidseitig) abhängig machen!

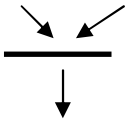
15.04.2010 (8 Vorlesung)

Abgabe Lösungen 1. Übungsblatt bis 27.4 beim Tutor

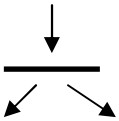
Beispiel: (Aktivitätsdiagramm)



Andere Elemente des Aktivitätsdiagramms

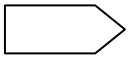


Join: Fügt unabhängige Kontrollflüsse zusammen

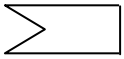


Fork: Kontrollfluss verzweigt in unabhängige Kontrollflüsse

Erzeugt Marken

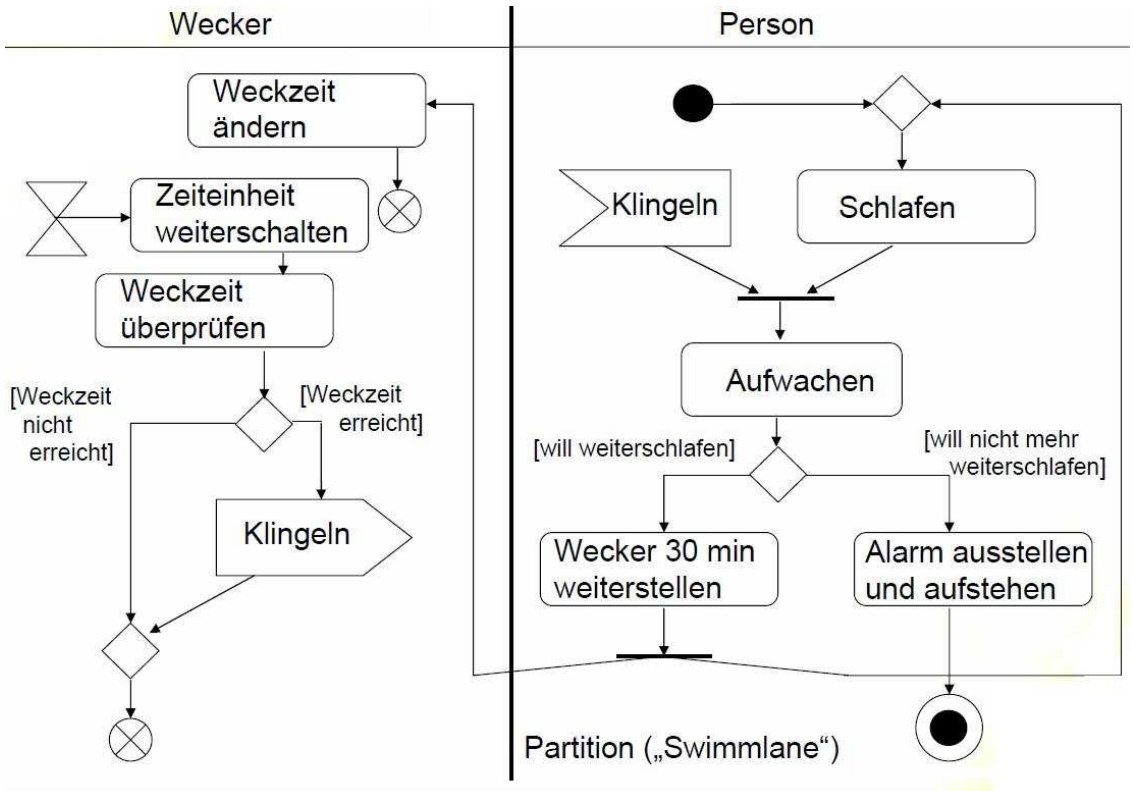


Auslösendes Ereignis



Verwickelt eine Marke (Ablaufendknoten - nur einen Ablauf stoppen)

Beispiel:



Aufgabe (Ablauf einer Vorlesung)

Vorlesungsbeginn erreicht
Vorlesungsende erreicht

Dozent kommt in Raum	Student kommt in Raum
Dozent begrüßt Studenten	Student hört zu
Dozent spricht	Student antwortet
Dozent fragt	Student fragt
Dozent antwortet	Student geht aus Raum
Dozent geht aus Raum	Student klopfen auf den Tisch
Dozent schließt Vorlesung	

15.04.2010 (9 Vorlesung)

22.04.2010 (10 Vorlesung)

UML Modellierung

Aufgabe

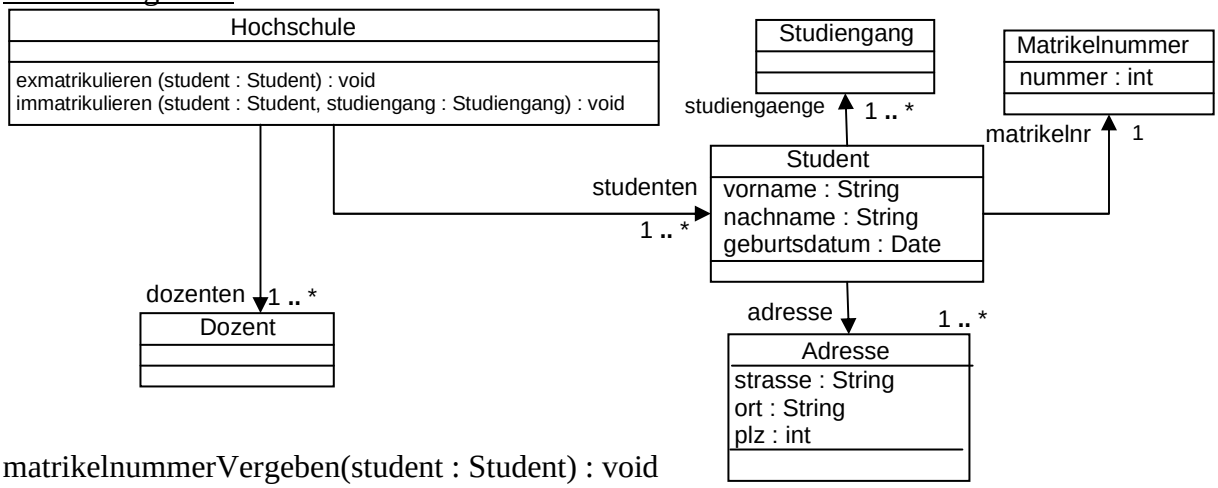
Eine Hochschule hat Studenten und Dozenten.

- Studenten haben einen Vor-/Nachname, ein Matrikelnummer und ein Geburtsjahr.
- Die Wohnadressen mit Strasse, Ort und Postleitzahl aller Studenten sollen vorhanden sein.
- Die Matrikelnummer soll von der Hochschule für einen Student vergeben werden.
- Studenten sollen sich für einen Studiengang an der Hochschule immatrikulieren oder exmatrikulieren können.

Klassen

Student (S)
Hochschule (H)
Adresse
Studiengang
Matrikelnummer
Dozent

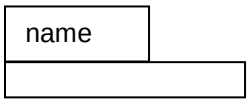
Klassendiagramm



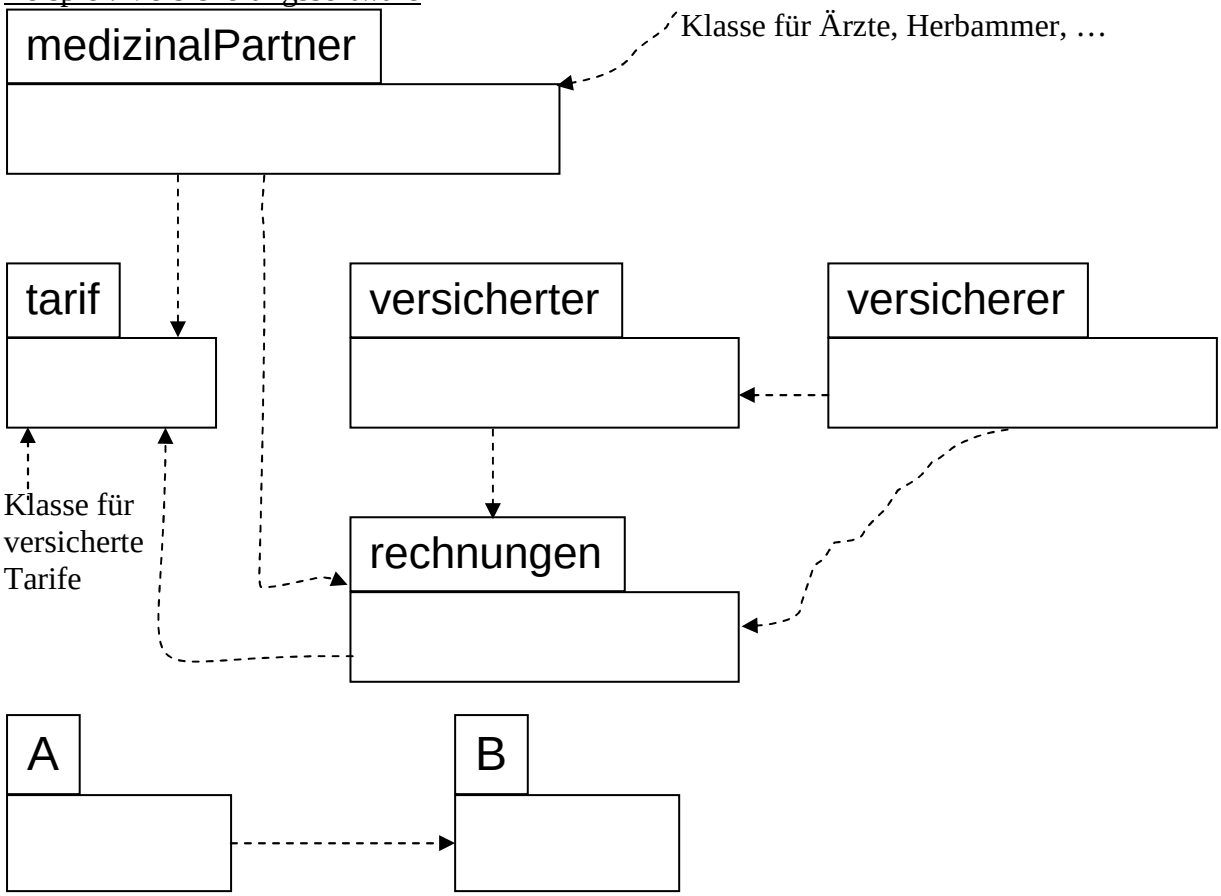
matrikelnummerVergeben(student : Student) : void

Pakete / Paketdiagramme

Paket (package): Menge von Klassen und Paketen

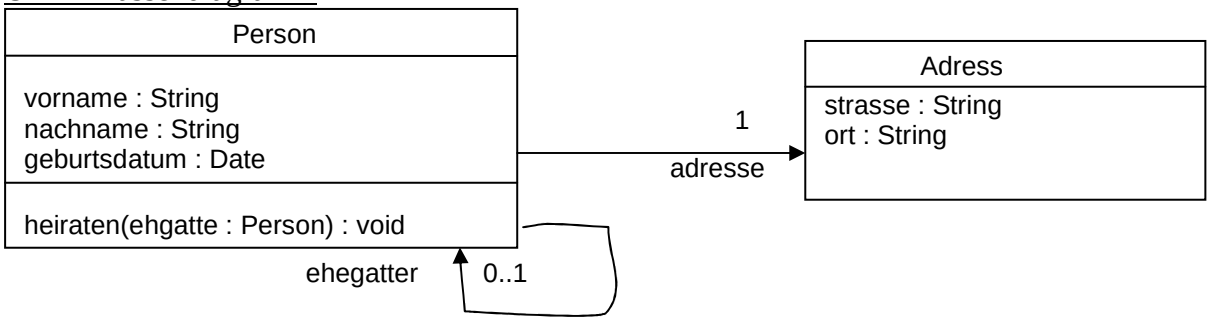


Beispiel: Versicherungssoftware



Klassen aus A dürfen auf Klassen von B zugreifen.

UML-Klassendiagramm



- Klassen definieren Wertemenge + Operationen (Methoden). Sie sind Datentypen.
- Konstruktor erzeugt ein Objekt einer Klasse
  z.B.: new Person(); erzeugt bei Ausführung ein neues Objekt von Typ Person

- Zugriff auf Objektattribute / -methoden mit • – Operator.

Implementierung eines UML-Klassendiagramms (Source)

```java
package standesamt_v1;

public class Person {

	public String vorname; //Object attribute.
	public String nachname;
	public int alter;
	public Person ehegatte;
	public Adresse adresse;

	public void heiraten(Person ehegatte) {
		ehegatte.nachname = this.nachname;
	}
}
```

```java
package standesamt_v1;

public class Adresse {
	public String ort;
	public String strasse;
}
```

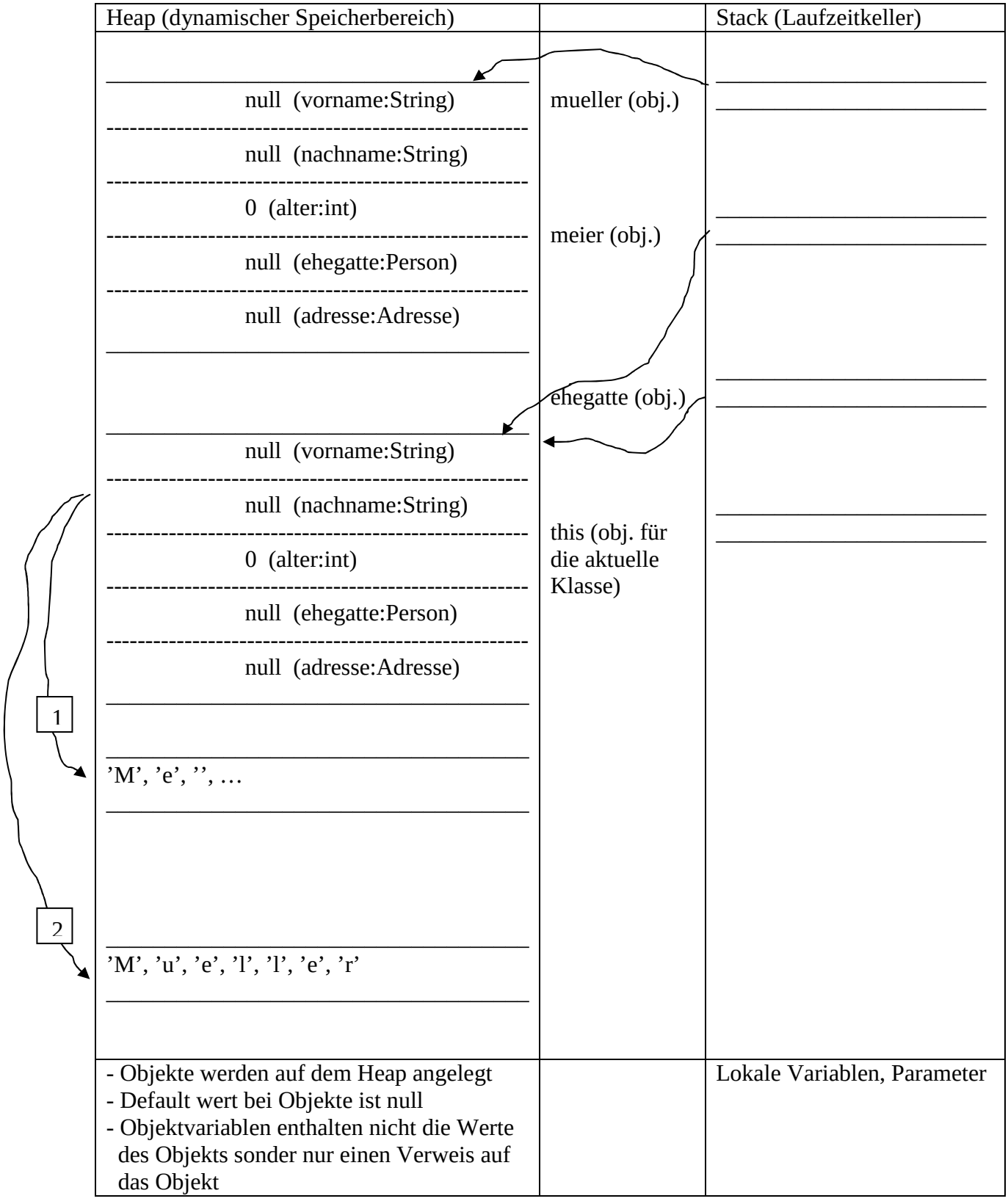
```java
package standesamt_v1;

public class Main {
	public static void main(String[] args) {

		Person mueller = new Person();
		mueller.nachname = "Müller";
		mueller.alter = 24;

		Adresse a = new Adresse();
		a.ort = "Karlsruhe";
		a.strasse = "Moltkestr.";

		mueller.adresse = a;
		System.out.println(mueller.adresse.ort);

		Person meier = new Person();
		meier.nachname = "Meier";
		mueller.heiraten(meier);
		System.out.println(meier.nachname);

		mueller.adresse = null;
		a = null;
	}
}
```

Heap (dynamischer Speicherbereich)		Stack (laufzeitkeller)
_____ null (vorname:String) ---------- null (nachname:String) ---------- 0 (alter:int) ---------- null (ehegatte:Person) ---------- null (adresse:Adresse) ----------		
- Objekte werden auf dem Heap angelegt - Default wert bei Objekte ist null - Objektvariablen enthalten nicht die Werte des Objekts sonder nur einen Verweis auf das Objekt		Lokale Variablen, Parameter

Im Java, .Net wird nicht benötigter Speicher (Heap) automatisch freigegeben (garbage Collector), z.B. falls kein Verweis mehr auf ein Objekt existiert.

26.04.2010 (11 Vorlesung)

Beispiel Person und Heiraten weiter machen.



Konvention:

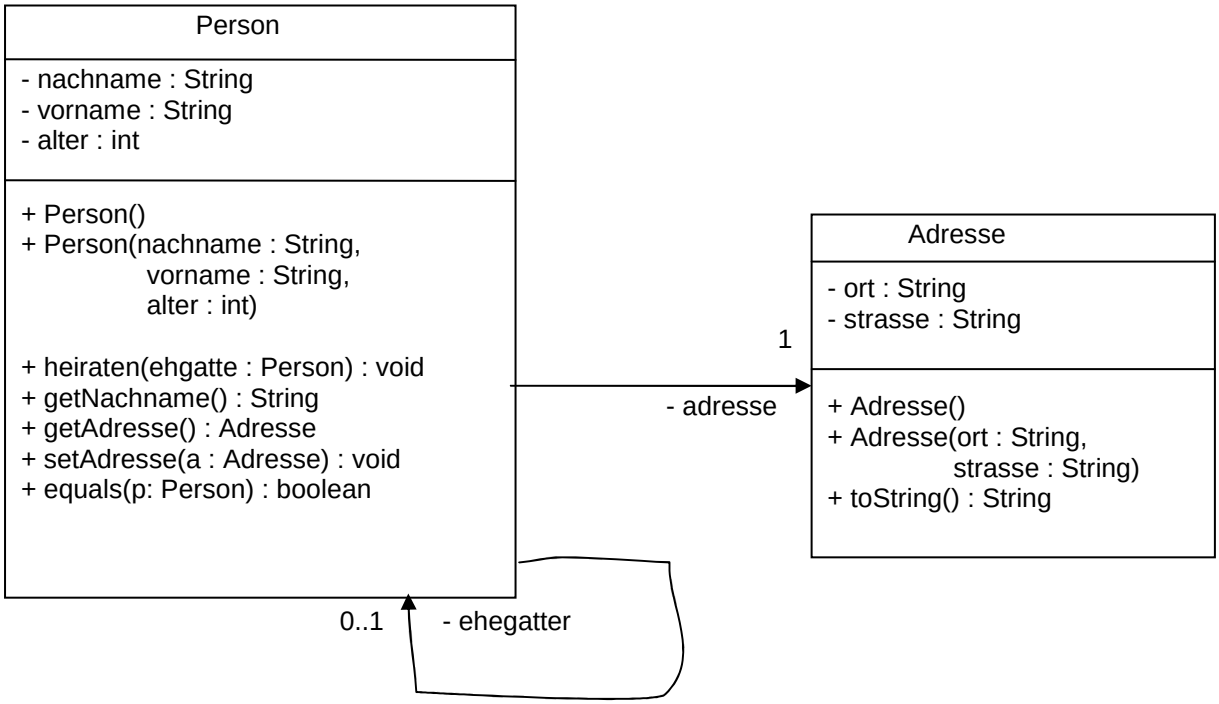
- Immer **this** innerhalb einer Objektmethode vor Objektattribute von **this** schreiben.
- = =, ! = Identitätsoperator vergleich nur Verweise (bei Datentypen, Klasse)
- equals(…) vergleichen auch nur die Verweise. Der Vergleich muss selbst programmiert werden.

Geheimprinzip:

- Datenkapslung (Zugriff auf Objektattribute einer Klasse nur in der Klasse selbst)
- In Java: Objektattribute mit **private** deklarieren

Default-Konstruktor existiert nur, wenn kein eigener programmiert wird.

Neues Klassendiagramm:



Neues Source-Code:

```java
package standesamt_v2;

public class Person {

  private String vorname; // Object attribute.
  private String nachname;
  private int alter;
  private Person ehegatte;
  private Adresse adresse;

  public Person() {
  }

  public Person(String nachname, String vorname, int alter) {
     this.nachname = nachname;
     this.vorname = vorname;
     this.alter = alter;
  }

  public void heiraten(Person ehegatte) { // Objectmethod
     this.nachname = (this.nachname + "-")
           + ehegatte.nachname;
     ehegatte.nachname = this.nachname;
     this.ehegatte = ehegatte;
     ehegatte.ehegatte = this;
  }

  public String getNachname() {
     return this.nachname;
  }

  public void setAdresse(Adresse a) {
     this.adresse = a;
  }

  public Adresse getAdresse() {
     return adresse;
  }

  @Override
  public boolean equals(Object person) {
     if (person instanceof Person) {
        Person p = (Person) person;
        if (this == p) {
           return true;
        } else {
           return this.nachname.equals(p.nachname)
                 & this.vorname.equals(p.vorname);
        }
     }

     return false;
  }
}
```

```java
package standesamt_v2;

public class Adresse {
        private String ort, strasse;

        public Adresse() {
        }

        public Adresse(String ort, String strasse) {
                this.ort = ort;
                this.strasse = strasse;
        }

        @Override
        public String toString() {
                return strasse + ", " + ort;
        }
}
```

```java
package standesamt_v2;

public class Main {
   public static void main(String[] args) {

      Person mueller = new Person("Müller", "Cyprian", 69);

      Adresse a = new Adresse("Karlsruhe", "Moltkerstr. 20");
      mueller.setAdresse(a);
      System.out.println(mueller.getAdresse());

      Person meier = new Person("Meier", "Anne", 19);
      System.out.println("Bevor: " + meier.getNachname());

      meier.heiraten(mueller);
      System.out.println("Danach: " + meier.getNachname());

      // Check of equals.
      Person p1 = new Person("Müller", "Cyprian", 69);
      Person p2 = new Person("Müller", "Cyprian", 69);
      System.out.println(p1.equals(p2));
   }
}
```

public - Zugriff von jeder Klasse aus (In UML: +)
private - Zugriff nur in der Klasse (In UML: -)
(nicht) - Zugriff von allen Klassen im gleichen Paket (In UML: ~)
protected - Zugriff von allen Klassen im gleichen Paket und in jeder Unterklasse (In UML: #)

Java Beans Design Pattern

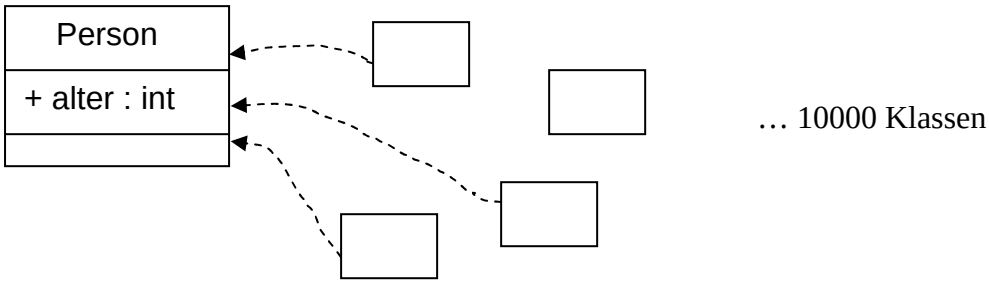
getAttribut(): Rückgabewert des Attributs (Bei boolean statt get, is => isSchaltjahr())
getAttribut(… Attributty …): void
Auch bei anderen Zustandsänderungen

29.04.2010 (12 Vorlesung)

Geheimnisprinzip

- Verbergen von Implementierungsdetails einer Klasse (Datenkapselung unterstützt dies Prinzip)

Falls alter public:



Änderung: Statt alter, soll Geburtdatum gespeichert werden
=> alle von alter direkt abhängigen Klassen müssen geändert werden (hoher Aufwand)

Bei Datenkapslung:
Chance Änderungen nur in einer Klasse implementieren zu müssen

```
Person p = new Person(…);
Date geburtsdataum = person.getGeburtsdatum();
geburtsdatum.setJahr(-1000);
Date geburtsDatum = new Date(…);
Person p2 = new Person (“Mueller”, “Heinz”, geburtsDatum);
geburtsDatum.setJahr(0);
```

Testen mit JUnit
Unit = Modul
Modultest
In Java: Klasse = Modul
Für jede Klasse wird eine Testklasse programmiert.

Jede Testmethode beginnt mit test.

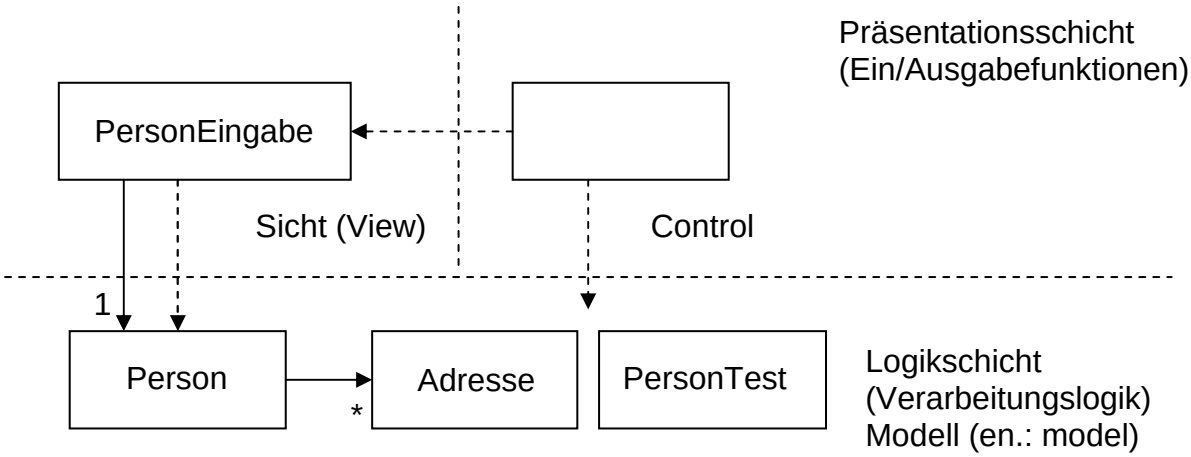
1.Testdaten erzeugen
2. Zu überprüfende Methode aufrufe
3. Ergebnis der Methode überprüfen
assertEquals(erwarteter Wert, tatsächlicher Wert)

Testfälle sollten viel kleiner als die zu überprüfende Implementierung sein
Testfälle müssen reproduzierbar sein, d.h. keine Zufälligen Daten erzeugen.

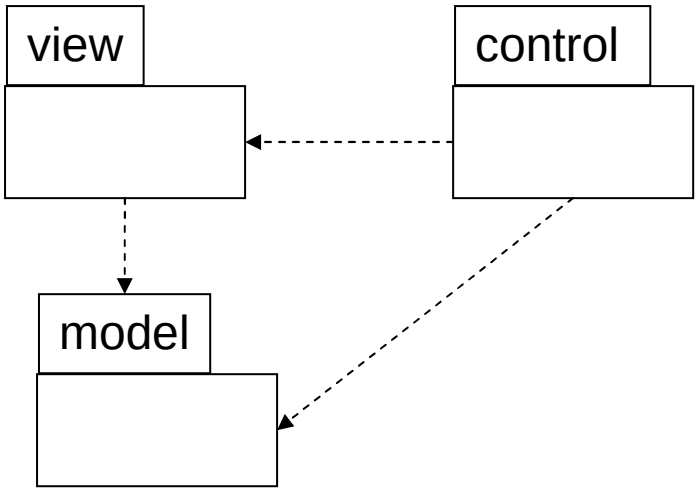
NullpointerException
Programm bricht ab, weil versucht wird bei einem null-Verweis einer Variablen auf Objektattribute oder – Methoden zuzugreifen

03.05.2010 (13 Vorlesung)

MVC – Model-View-Control



Paketdiagramm bei komplexe Entwürfen oder wenn wir viele Klassen haben.



Klassenkommentar: (Begleitende Material: Der Beispiel im Java-Projekt von Tutorium 5)

- Erster Satz erscheint in der Klassenübersicht.
- Beschreibe um welche (realen) Objekte es sich handelt.

Methoden:

- Die Methoden-Kommentare sollen auf die folgende Frage antworten:
  Was macht die Methode?
- Verb oder Methode im Präsens verwenden.
- @param erlaubten Werte beschreiben.

Konstruktoren: (wie bei einer Methode)

- Verb erzeugt

Kommentare müssen:

a) kurz sein und
b) spezifisch sein (zusätzliche relevante Information enthält)

06.05.2010 (14 Vorlesung) Fehlt!

10.05.2010 (15 Vorlesung)

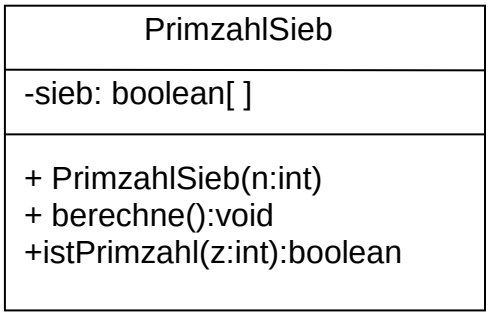
Abgabe: 2.Übungsblatt bis 8.6.

Beispiel: Primzahlsieb
Gegeben: eine ganze Zahl n > 2
Gesucht: Alle Primzahlen zw. 2 und n
1 18 (n = 18)
|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|

Schritt 1: Fange mit 2 an und das Rest +2 addieren und streichen…
| 0 | 1 | 2 | 3 |~~ 4 ~~| 5 |~~ 6 ~~| 7 | 8 | 9 |10|11|12|13|14|15|16|17|18|
/\------/\-----/\------/\-----/\ …
+2 +2 +2 +2 …

Schritt 2: Fange mit 3 an und das Rest +3 addieren und streichen…
| 0 | 1 | 2 | 3 |~~ 4 ~~| 5 |~~ 6 ~~| 7 |~~ 8 ~~|~~ 9 ~~|~~10~~|11|~~12~~|13|14|15|16|17|18|
/\---------/\---------/\----------/\ …
+3 +3 +3 …

Entwurf:



Source:
Sehen Sie das beigefügte Java-Projekt mit Eclipse: tt.tutorial-6.primzahl

Junit-Test-Klasse:
Sehen Sie das beigefügte Java-Projekt mit Eclipse: tt.tutorial-6.primzahl

Zweidimensionale Felder:

$$\begin{pmatrix} 5 & 3 & 8 & 2 & 1 \\ 2 & 1 & -5 & 4 & 3 \\ -5 & 7 & 4 & 8 & -6 \end{pmatrix}$$

```
                [ ] -> | 1 | 2 |
a     ->        [ ] ->
                [ ]
```

```
int [] [] matrix;
matrix = new int [3] [5];
```

Symetrisches Feld :

```
int [] [] matrix = {    {5, 3, 8, 2, 1},
                        {...},
                        {-5, 7, 4, 8, -6}       }
```

Asymmetrisches Feld:

```
int [] [] a = {   {1, 2},
                  {1, 2, 3},
                  null              }
```

```
a [2] [1] = 7; //NullPointerException
```

Beispiel :

```
                                1
                            1       1
                        1       2       1
                    1       3       3       1
                1       4       6       4       1
            1       5       10      10      5       1
```

$(a+b)^0 = 1$
$(a+b)^1 = 1*a + 1*b$
$(a+b)^2 = 1*a^2 + 2*a*b + 1*b^2$
$(a+b)^3 = 1*a^3 + 3*a^2*b + 3*a*b^2 + 1*b^3$
$(a+b)^5 = 1*a^5 + 5*a^4*b + 10*a^3*b^2 + 10*a^2*b^3 + 5*a*b^4 + 1*b^5$
$(a+b)^n = \binom{n}{0}*a^n + \binom{n}{1}*a^{n-1}*b + \; + \; \binom{n}{n-1}*a*b^{n-1} \; + \; \binom{n}{n}*b^n$

17.05.2010 (16 Vorlesung) - Rekursion

Praxisforum Mi. 19.5., 11:30

Pflichtaufgabe Sudoku
Abgabe: bis Mo. 28.6 (Pape)
Bis Di 29.6. (Körner)

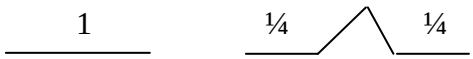
S

Z										
	1	3	7							
	6	2	4		1	9	5			
	5	9	8					6		
	8					6				
	4						3			1
			W			2				
		6						2	8	
			4		4	1	9			5
									7	

Model
- Konstruktor mit patiell ausgefullten Spiel
- Wert an Zeile, Spalte Wert ermitteln
- Ist Zug an Stelle z, s mit w erlaubt?
- Wert setzen, falls erlaubt

Züge brauchen nicht rückgängig gemacht werden.
„Sackgassen“ brauchen nicht erkannt werden.

Rekursion:



n! := 1.2.3.4 … (n-1).n

$$n! := \begin{cases} 1 & n = 0 \\ (n-1)!.n & n > 0 \end{cases}$$ rekursive Definition

Bsp.: 7! = 6!*7=5!*6*7=4!*5*6*7=…=0!*1*2*…*7= 1*…*7

$$ggT(a, b) := \begin{cases} a & \text{falls } a = b \\ ggT(a-b, b) & a > b \\ ggT(a, b-a) & a < b \end{cases}$$

ggT(6, 18) = ggT(6, 12) = ggT(6, 6) = 6

Bsp.:

$$(n|k) := \begin{cases} 1 & k = 0 \text{ oder } n = k \\ (n-1|k-1) + (n-1|k) & \text{falls } 0 < k < n \end{cases}$$

Fakultät von 4 als Bild: Sehen Sie Tutorium 7

Erster rekursiver Aufruf hat Rekursionsebene (-tiefe) 0

Lineare Rekursion:

Ein Aufruf einer Methode hat höchstens einen return Aufruf zur Folge

Bsp.: FibonacciV1

1 Paar Kaninchen (m+n)

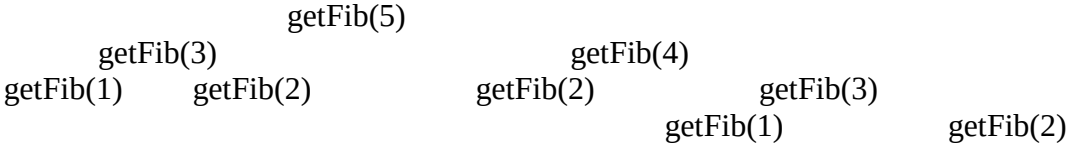
2. Generationen Zur Geschlechtsreite

1 Paar wirkt ein weiteres Paar, Kaninchen sterrben nicht.

N	Paare
1	1
2	1
3	2
4	3
5	5
6	8
7	13

$$fib(n) := \begin{cases} 1 & , n <= 2 \\ fib(n-2) + fib(n-1) & , n > 2 \end{cases}$$

Baumaufruf:



Rekursionstiefe 3

Verzweigende Rekursion:

Ein Methodenaufruf kann mehr als einen rekursiven Aufruf zu Folge haben

Dynamisches Programmieren:

Ergebnisse berechnen indem Teilergebnisse wiederverwendet werden.

Teilergebnisse in Feld speichern

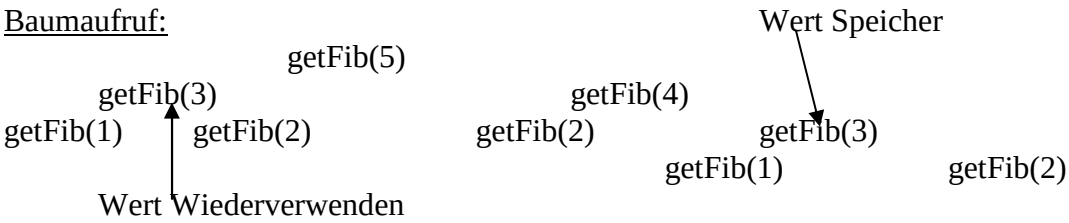
20.05.2010 (17 Vorlesung) - Rekursion

Dynamisches Programmieren

Bsp.: FibonacciV2

$$fib(n) := \begin{cases} 1 & , n <= 2 \\ fib(n-2) + fib(n-1) & , n > 2 \end{cases}$$

Baumaufruf:



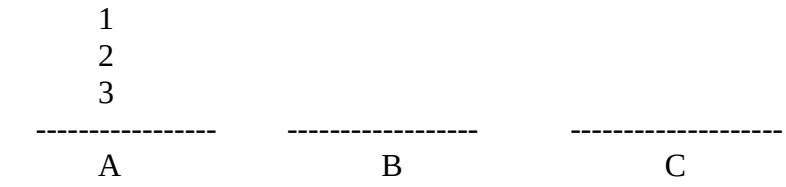
Wert in ein Feld speichern

fib[n] = fib(n)
fib[n] = falls wert noch nicht berechnet

	1	2	3	4	5	6
	1	1	2	3	5	8

Also Fibonacci-Zahl von 4 ist die Summe von vorherigen zwei Zahlen (2 + 3)
Fibonacci-Zahl von 5: 3 + 2
Fibonacci-Zahl von 6: 5 + 3 usw.

Bsp.: Türme von Hanoi



Immer kleinere Zahl auf eine größere Zahl
A -> B oder A -> C

1. A(1) -> C(1)
2. A(2) -> B(2)
3. C(1) -> B(1)
4. A(3) -> C(3)
5. B(1) -> A(1)
6. B(2) -> C(2)
7. A(1) -> C(1)
8.

1 Parameter: Anzahl-Scheibe
2. Parameter: von
3. Parameter: zu
4. Parameter: Hilf-Stab

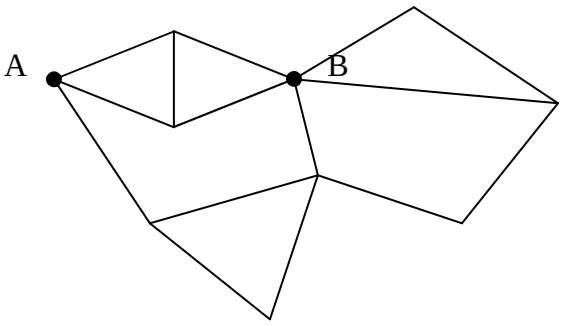
hanoi (4, 1, 3, 2)

hanoi(3, 1, 2, 3) hanoi(3, 2, 3, 1)

. .
. .
. .

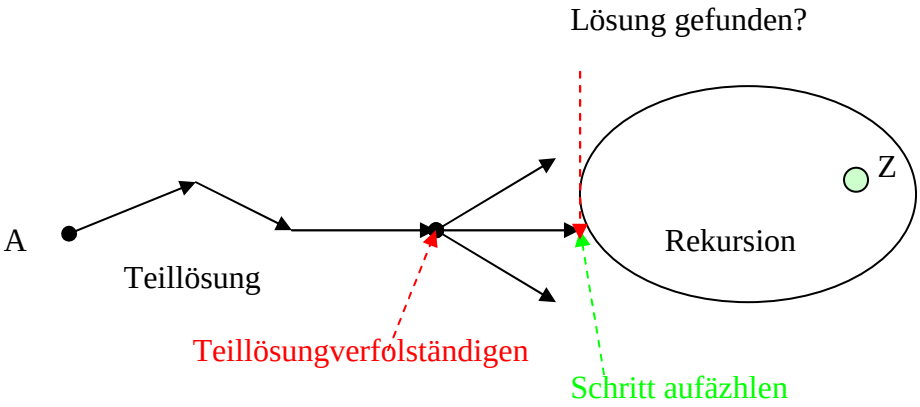
```
public void hanoi(int n, int von, int zu, int hilf) {
        if (n == 1) {
                Syso(von + "->" + zu);
        } else {
                hanoi(n-1, von, hilf, zu);
                Syso(von + "->" +  zu);
                hanoi(n-1, hilf, zu, von)
        }
}
```

Aufrufe:
1, 2, 4, 8, …
$2^n - 1$
2^{65}- 1 +~ $2^{65} = 2^{6*10} * 2^5 \sim 1000^6 * 2^5$
Mehr als 1 Billion Aufrufe $(10^3)^6$

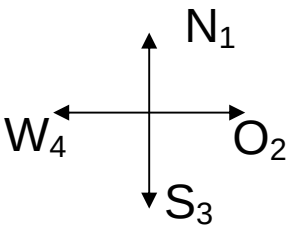


Versuch und Irturm / trial and errov
Zurückgehen : backtracking

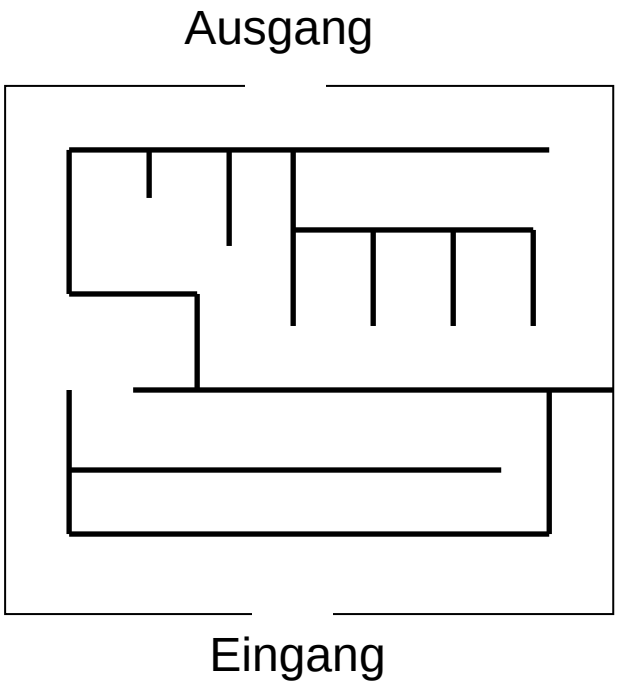
31.05.2010 (18 Vorlesung) - Backtracking



Alle Schritte aufzählen



Labyrinth Bild von der Tafel:



Fazit:

- Lsg. Gefunden
- Algorithmus ist sehr schnell
- Findet der kurzeste Weg

Springerproblem
Ab ein Feld von 4x4, bei 4x4 ist das Feld zu klein und es geht nicht.

	18	3	8	13
		12	17	4
19	2	7	14	9
	11	16	5	
1	6		10	15

Bei 8x8 Feld dauert es zu lange, kann sein Tagen oder Stunden ☺.

10.06.2010 (19 Vorlesung) Fehlt!

14.16.2010 (20 Vorlesung)

$T(n) = 2*T(n/2) + c*n$
$= 2*[2*T(n/4) + c*n/2] + c*n$
$= 2^2*T(n/2^2) + 2*c*n$
$= 2^2*[2*T(n/2^2) + c*n/2v]\ 2*c*n$
$= 2^3*T(n/2^3) + 3*c*n$
. . . k mal
$= 2^k*T(o) + k*c*n = 2^k*c + k*c*n = 2*n*c + \log_2(2*n)*c*n$
$= O(\log_2 (2*n)*n) = O(\log_2 (n)*n)$

	n^2	$(\log_2 n)*n$
1.000	$(1000^2)10$ = 10ms	10*10*1.000, 100 microsececunds (µs).
1.000.000	$(1000^4)10$ = 10.000ms	10*20*1.000.000, 200 (µs).

Mergesort (Zeitaufwand) $O(n*\log_2 n)$ im schlimmsten, bestens, durchschnittlichen Fall

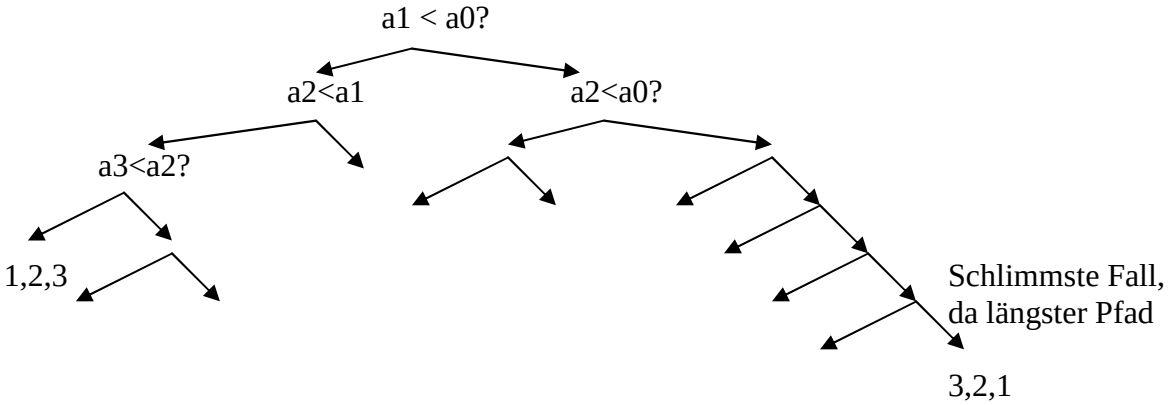
Untere Schranke für ein Problem der Größe n:
Zeitaufwand im schlimmsten Fall, den jeder Algorithmus mindestens benötigt:
$O(\log_2 *n!) = O(n*\log_2 *n)$

Zählen alle Vergleiche von Werten der Folge

n=3

a_0	a_1	a_2
1	2	3
2	1	3
1	3	2
3	2	1
2	3	1
3	1	2

Entscheidungsbaum:



n Elemente, n! verschiedene Anordnungen

Längste Pfad $\log_2(n!)$

$\log_2(n!)$ >= $\log_2(n/2 * (n/2+1) * (n/2+2) \ldots n)$
>= $\log_2(n/2*n/2 \ldots n/2* n/2)$
>= $\log_2((n/2)^{n/2}) = n/2* \log_2(n/2) = O(n*\log_2 n)$

Runden beliebiggroßer Zahlen {1, . . ., n}

Bubblesort:

|=> Benachbarte Werte vertauschen, falls linke wert größer als rechte Wert ist.

5	**3**	**8**	**7**	**2**	**1**	**4**	**6**
5	**3**	**8**	**7**	**1**	**2**	**4**	**6**
5	**3**	**8**	**1**	**7**	**2**	**4**	**6**
5	**3**	**1**	**8**	**7**	**2**	**4**	**6**
5	**1**	**3**	**8**	**7**	**2**	**4**	**6**
1	**5**	**3**	**8**	**7**	**2**	**4**	**6**
1	**2**	**5**	**3**	**8**	**7**	**4**	**6**
1	**2**	**3**	**5**	**4**	**8**	**7**	**6**

(n-1) Vergleich
(n-2)
(n-3)

1/$O(n^2)$ Vergleich
$O(n^2)$ Vertauschen im schlimmsten Fall

Warum Bubble?

|=> die kleinere Elemente sehen so aus, als sie wandern nach oben (wie kleine blasen)

Direktes Einfügen:

3	**5**	**2**	**8**	**1**	**7**	**4**	**6**
3	**5**	**2**	**8**	**1**	**7**	**4**	**6**
2	**3**	**5**	**8**	**1**	**7**	**4**	**6**
2	**3**	**5**	**8**	**1**	**7**	**4**	**6**
1	**2**	**3**	**5**	**8**	**7**	**4**	**6**
1	**2**	**3**	**5**	**7**	**8**	**4**	**6**
1	**2**	**3**	**4**	**5**	**7**	**8**	**6**
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**

a| | | | | |
i j

Für jeden i = 1, 2, …,n
Finde Pos. j an der a[i] „hingehört“ a[j] . . .a[i-1] nach rechts verschieben a[i]a_n Stelle a[j] schreiben

Schlimmste Fall: Folge ist absteigend sortiert: 1+2+3+4+. . . +(n+1) = $O(n^2)$ Vergleiche

Binärsuche $O(m\log_2 n)$ => $O(n^2)$ Verschiebeoperationen

Bester Fall. Folge bereits sortiert = O(m) Zeitaufwand