

Wdh. : Arrays

- in Java sind Arrays auch Objekt, das bekannt
 1. Array-Variablen sind Referenzen
 2. Arrays besitzen Methoden
 3. Arrays werden zur Laufzeit erzeugt
- Zugriff über einen numerischen Index (Datentyp int)
 - => 1. Index stets 0
 - letzter Index stets Anzahl Elemente - 1

Erzeugung in 2 Schritten:

1. Deklaration `int [] coeffs;`
2. Erzeugung eines Array-Objektes und Zuweisung an die Array-Variable

`coeffs[0];` `coeffs = new int [3];`

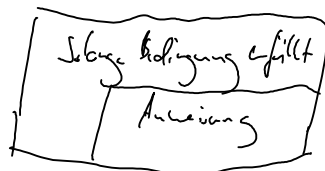


Mehr-Dim.-Array

`int [][] coeffs = new int [3][3];`

Wdh. : Kontrollstrukturen

- 1.) while Schleife: Wiederholte Ausführung einer Anweisung (Block) in Abhängigkeit einer Bedingung



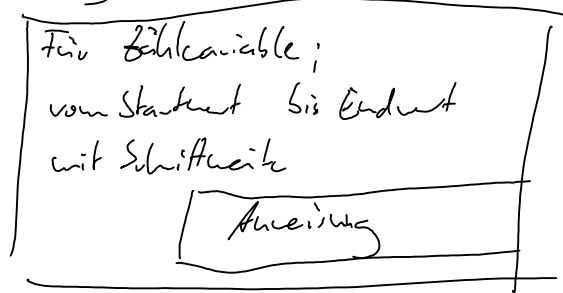
Gefahr?
Endlosschleife

Bsp.:

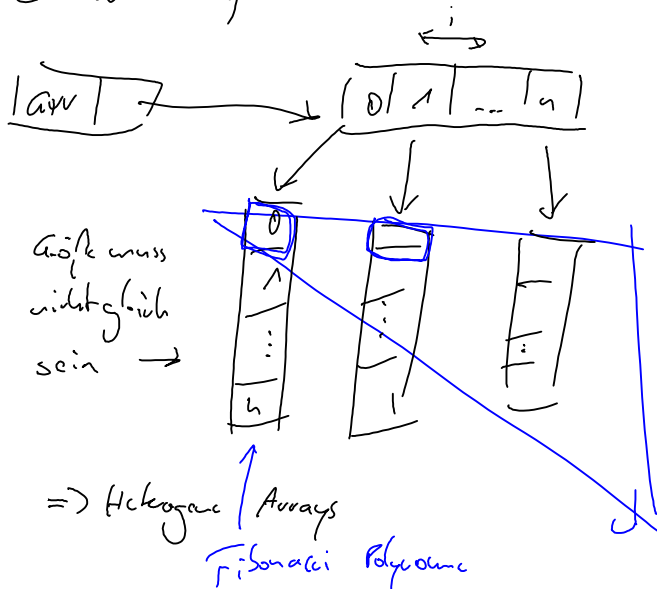
```
int i=0;
while (i < coeffs.length)
{
```

```
    Anweisung ;
    i++; hier Gefahr
}
```

2.) for-Schleife : Mehrfache Ausführung einer Anweisung unter Steuerung einer Zählvariable



Wdh. 2-dim Array



Übung 5Frage 1

class Ostkuh

{

String name;boolean isMale;int age;final boolean flies = false;

↳ Attribut mit konstanten Wert (Wert bleibt für die gesamte Lebenszeit des Objekts erhalten)

}

class Parrot

{

String name;boolean isMale;int age;final boolean flies = true;

boolean speaks;

}

class Dolphin

{

String name;boolean isMale;int age;final boolean livingInWater = false;

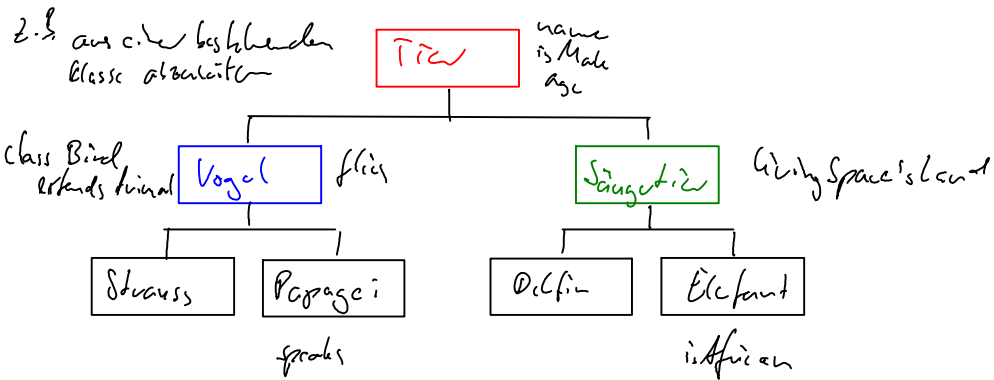
}

```

class Elefant
{
    String name;
    boolean isMale;
    int age;
    final boolean livingSpaceIsLand=true;
    boolean isAfrican;
}
    
```

Frage 2

Klassenhierarchie



Allg: Deklaration einer Variable in der Form

Typname Variablenname;

Bsp: Delfin d; Animal a;

z.B. jetzt: $d = \text{new Delfin}();$ // Initialisierung durch Objektzeugung und Zuweisung

aber auch: $a = \text{new Delfin}();$ // da Delfin ein Animal ist weist a hier auf ein "Animal"-Objekt (näherliegender auf ein "Delfin"-Objekt)

! jedoch nicht!
Bird ~~f~~new Elefant;

Zugriff auf Attribute am Beispiel eines "Elephants"-Objekt

```

name = "Hugo";
isMale = true;
age = 18;
livingSpaceIsland = true;
isAfrican = true;
    
```

Deklaration:

```

Animal a;
Mammal m;
Elephant e;
    
```

Initialisierung:

```

a = new Elephant();
m = - " - ;
e = - " - ;
    
```

```

a.age = 20; ✓ m.age; ✓
    
```

```

a.livingSpaceIsland = true;
    
```

```

a.isAfrican = true;
    
```

```

e.age = 20; ✓
    
```

```

e.livingSpaceIsland = true; ✓
    
```

```

e.isAfrican = true; ✓
    
```

Typumwandlung

```

((Elephant)a).isAfrican = true;
    
```

Problem: Laufzeitfehler möglich, da nur gültig wenn tatsächlich a auf "Elephant"-Objekt weist.

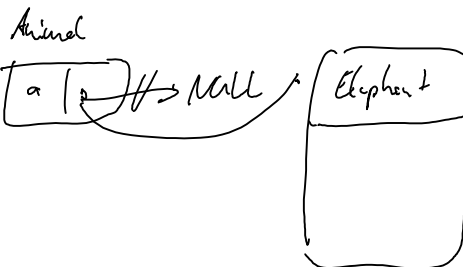
Instance-Operator

Prüfung zur Laufzeitmöglichkeit
 if (a instanceof Elephant)



```

((Elephant)a).age = 20;
disc System.out.println("...");
    
```



Frage 4

```

Animal[] animals = new Animal[10];
animals[0] = new Elephant();
animals[1] = new Parrot();

```

Beispiel inform()-Methode der Klasse Elefant

```

class Elefant extends Mammal
{
    private boolean isAfrican;
    ...
    public void inform()
    {
        if (isMale)
            System.out.print ("name ist männlich");
        else
            ... ("name ist weiblich");
        System.out.print ("and " + age + " Jahre alt");
        ... //Eigenschaften angeben
    }
}

```

Frage 6

```

for (int i=0; i<animals.length; i++)
{
    if (animals[i] instanceof Ostvuh)
        ((Ostvuh) animals[i]).inform();
    ...
    if (animals[i] instanceof Elefant)
        ((Elefant) animals[i]).inform();
}

```

Frage 8

```
for(int i=0; animals.length; i++)  
{  
    animals[i].inform();  
}
```

↳ Dynamische Bindung

Frage 9

```
abstract class Animal  
{  
    ..  
}
```

Übung 6 Interpreter

Teil 1 Zeile 0: z.B. Zuweisung eines Wertes an eine Variable
 Zeile 3: z.B. Ausgabe des Wertes

Datentypen in Java:

Ein Datentyp (oder einfacher Typ) ist charakterisiert durch

- eine Trägermenge

- auf diese Trägermengen definierten Operationen

z.B. int ganze Zahlen Operation
 ↑ (Zeichenbehaftet) +, -, *, /, %, =
 Typname

Wahl: Hash-Speicher

Idee: Speichern und Suchen von Elementen in einer Liste durch Ausrechnen der Listenposition aus einem (eindeutigen) Schlüsselwert.

Def. Hashing (allg.): Unter Verwendung einer Funktion h wird anhand eines seq. Schlüssel k die Position i (Index) des entsprechenden Datenwertes ermittelt.

Im Folgenden wird die Position i in der Hash-Tabelle über Division-Rest (modulo) ermittelt;

$$H(k) = k \bmod p, \quad p = (\text{i. d.}) \text{ Primzahl}$$

Beispiel: Kundennummer $(1, 6, 9, 10)$ $p=7$

0	1	2	3	4	5	6

$h(1) = 1$
 $h(6) = 6$
 $h(9) = 2$
 $h(10) = 3$

Im Vergleich: Seq. Durchsuchen benötigt im Mittel $\frac{n}{2}$ Zugriffe

Vgl. / Vergleich Array und Listen

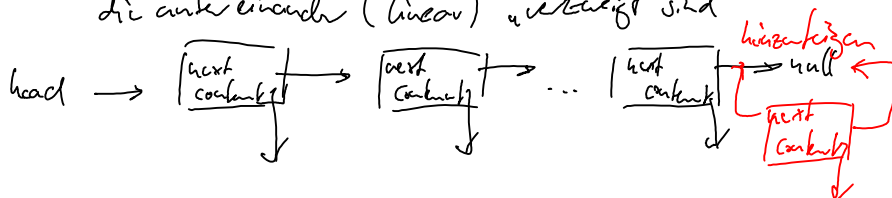
1. Array:

$\text{int}[] a = \text{new int}[100];$

Nachteil: Begrenzung der Elemente \rightarrow Wissen über die Anzahl muss vor dem Anlegen eines Arrays bekannt sein.

Vorteil: Direkter Zugriff auf Elemente
 \rightarrow schnell
 \rightarrow konstant

2. Listen: Eine dyn. Datenstruktur, die aus einer Menge von Knoten besteht, die untereinander (linear) "verknüpft" sind



Frage 1: Kollision \Rightarrow Mehrfache Abbildung

Bsp.: k.-Nr.: $H(1), H(3), H(8), H(15)$



$$H(k) = k \bmod 7$$

1. Überlauf mit
separaten Bereich

Vorteil: Beladungsfaktor
kann > 1 sein

Nachteil: Zus. Platz für
Zeiger und Überlauf

2. Überlauf ohne separaten Bereich



Wdh.: h k faces

- In Java ist keine echte "Mehrfachbelegung" möglich
- Bezeichnen h k faces
- Beschränkte Funktionalität, kein Überlauf

Ziel: Speichern und effizientes Suchen von Objekten im Speicher
(oder Datenbank)

Objekte unabh. klassenspezifisch gespeichert werden

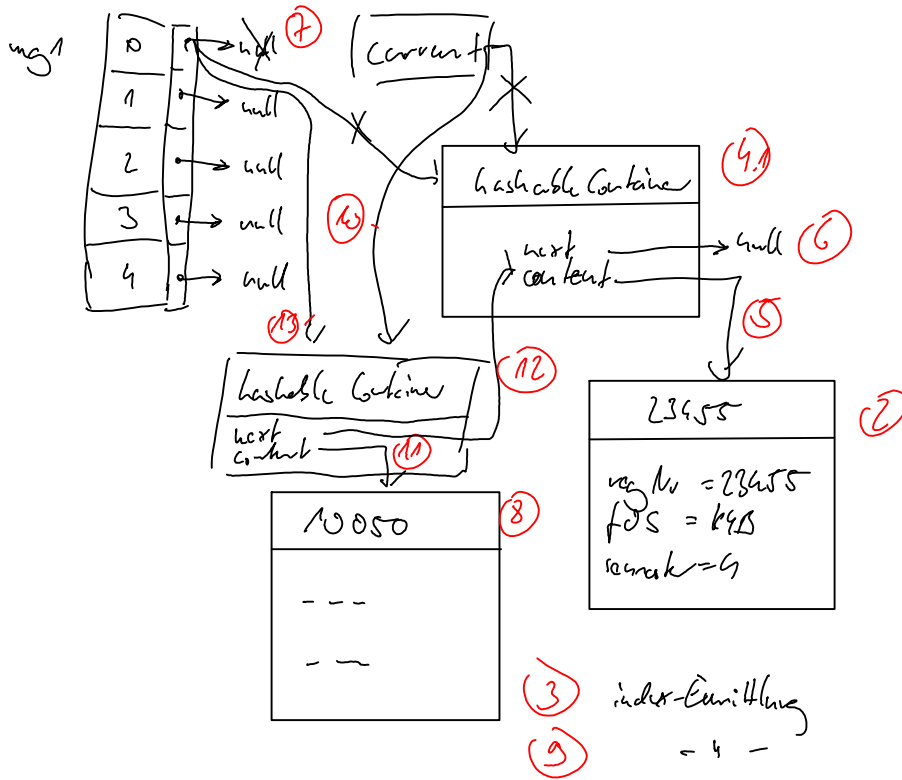
\rightarrow Lösung mit h k face Konzept

\rightarrow Objekte, deren Klassen das h k face hashable interface implementieren, besitzen die Fähigkeit im Speicher oder in DB gespeichert zu werden.

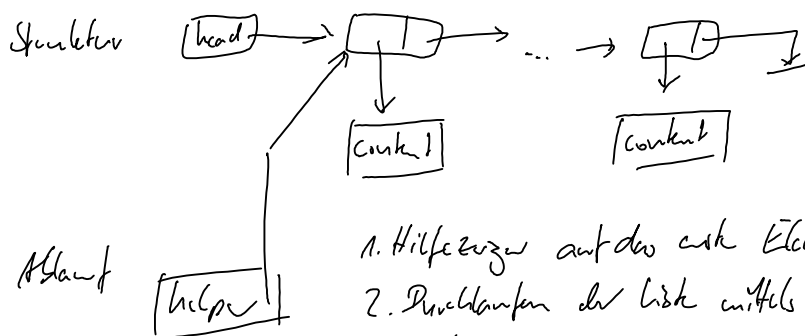
\Rightarrow Separierungsprinzip: Trennung Struktur und Daten

Beispiel Studypoker

Hash-Tabelle mit $p=5$ $h(k) = k \bmod 5$



Wdh: Linear, einfach verkettete Listen



1. Hilfszeiger auf das erste Element
2. Durchlaufen der Liste mittels des Hilfszeigers

Operationen = add, remove, search

in Java: Verwendung von Containerklassen

- Verkettete Liste \rightarrow LinkedList

- Merkmale:
 - bewahrt keine Ordnungrelation
 - kann Objekte beliebiger Klassen verwahren
- jedes Objekt in der Liste erhält einen Index $0, 1, \dots, n$

Methoden der Klasse LinkedList (Auswahl)

- | | | |
|--------------|--------------|-----------------|
| - addFirst() | - getFirst() | - removeFirst() |
| - addLast() | - getLast() | - removeLast() |
| - add() | - get() | - remove() |

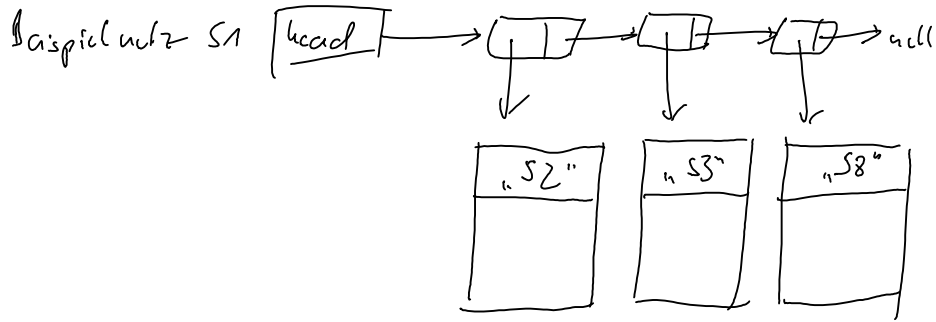
Alternativen zum Durchlaufen einer Liste

- Methoden: Konstruktor, hasNext(), next(), remove()

↑
darf nur aufgerufen werden
wenn hasNext() true zurückliefert

Klasse Node

LinkedList reachable Nodes



Beispiel: (Indizes) $\times (1)$

	Zeilensuche							
S1	0	1	1	0	0	0	0	1
S2	0	0	0	0	0	0	0	0
S3	0	1	0	0	0	0	0	1
S4	-	-	0	-	-	-	-	-
S5	-	-	0	-	-	-	-	-
S6	-	-	-	0	-	-	-	-
S7	-	-	-	-	0	-	-	-
S8	1	0	0	1	1	0	1	0

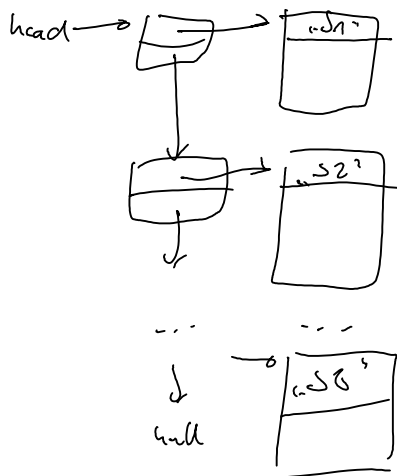
↓
Statt-
knoten

→ über die verkettete Liste
"reachable Nodes"

→ Invertieren

Klasse Invertiert

LinkedList nodes



Vergleich Tiefen- und Breitensuche in einem Grafen

-> Verfahren zum Durchsuchen bzw. Durchlaufen der Knoten in einem Grafen

1. **Breitensuche**

Zuerst wird ein Startknoten ausgewählt. Von diesem aus wird jeder direkte Nachfolger betrachtet und getestet ob der direkte Nachfolger schon untersucht wurde bzw. das gesuchte Element ist. Ist dies nicht der Fall, so wird der entsprechende Knoten in einer Liste gespeichert. Nachdem **alle direkten** Nachfolger der Startknoten betrachtet wurden, und das erste Element in der Liste der Startknoten ausgewählt und das Verfahren wiederholt.

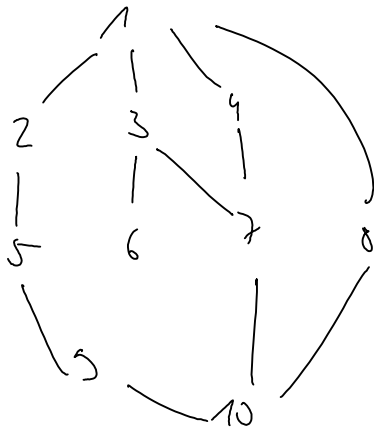
2. **Tiefensuche**

Zuerst wird ein Startknoten ausgewählt. Von diesem aus wird der erste direkt nachfolgende Knoten betrachtet und getestet, [...]. Ist dies nicht der Fall, so wird **rekursiv** fuer diesen Knoten die Tiefensuche aufgerufen, wodurch wieder der erste Nachfolger dieses Knotens ausgewählt wird. Dieses Verfahren wird solange fortgesetzt bis das gesuchte Element gefunden wurde oder alle Knoten untersucht wurden.

ToDo: Erstellen eines Struktogramms für die Tiefen- und die Breitensuche.

Beispiel = Graph mit 10 Knoten

Startknoten 1



=> Breitensuche (Reihenfolge beim Durchlaufen)

1, 2, 3, 4, 8, 5, 6, 7, 10, 9

=> Tiefensuche (Reihenfolge beim Durchlaufen)

1, 2, 5, 9, 10, 7, 3, 6, 4, 8