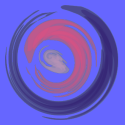
Java in 2 Tagen

Helge Janicke
Karsten Wolke
Niels-Peter de Witt



Inhalt

1. Zeitplan

9:00 Begrüßung und Gruppen bilden.

9:15 Klassen und Modellierung.

10:00 Pause.

10:15 Objekte und Referenzen.

10:45 Aufgabe.

12:00 Pause.

13:00 Vererbung.

13:45 Aufgabe.

14:30 Pause.

14:45 Objekterzeugung, Zugriff, static.

15:45 Pause.

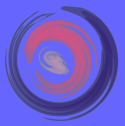
16:00 Strukturgramme & Aufgaben.

16:45 Pause.

17:00 Fehlerbehandlung.

18:00 Schluss.

9:00 4 Gewinnt.



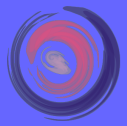
Inhalt

2. Was sind Klassen

Objektorientierte Sprachen fassen

Daten (*Attribute*)
und
Algorithmen (*Methoden*)

zu logischen Einheiten zusammen.



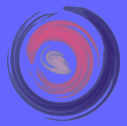
2. Was sind Klassen

Objektorientierte Sprachen fassen

Daten (*Attribute*)
und
Algorithmen (*Methoden*)

zu logischen Einheiten zusammen.

Diese logischen Einheiten nennt man Klassen.



2. Was sind Klassen

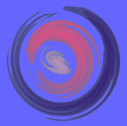
Objektorientierte Sprachen fassen

Daten (*Attribute*)
und
Algorithmen (*Methoden*)

zu logischen Einheiten zusammen.

Diese logischen Einheiten nennt man Klassen.

Klassen werden (abstrakten) Gegenständen der realen Welt nachempfunden.



2. Was sind Klassen

Objektorientierte Sprachen fassen

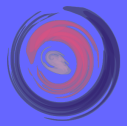
Daten (*Attribute*)
und
Algorithmen (*Methoden*)

zu logischen Einheiten zusammen.

Diese logischen Einheiten nennt man Klassen.

Klassen werden (abstrakten) Gegenständen der realen Welt nachempfunden.

Eine Klasse ist eine Beschreibung der Merkmale und Fähigkeiten eines solchen Gegenstandes.



2.1. Ein Jabberwok

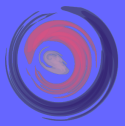
Ein Jabberwok ist ein urzeitliches drachenähnliches Monster, daß mit 100 Jahren erwachsen ist.

Geschützt wird ein Jabberwok durch eine ungerade Anzahl von Schuppen.

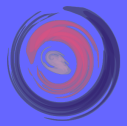
Jabberwoks schlüpfen aus einem Ei, daß von einem erwachsenen weiblichen Jabberwok gelegt wurde. Mit dem Legen des Eies ist der neue Jabberwok „geboren“.

Nur der Jabberwok kennt seine Mama.

Jeder Jabberwok kennt alle anderen Jabberwoks.



Es müssen für die Modellierung geeignete Klassen identifiziert werden.



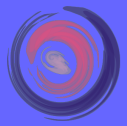
Inhalt

Es müssen für die Modellierung geeignete Klassen identifiziert werden.

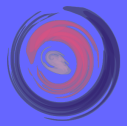
Es soll nur eine Klasse entworfen werden, die einen Jabberwok repräsentiert.

Merkmale und Fähigkeiten für einen Jabberwok identifizieren:

Bezeichnung: `Jabberwok`	
Merkmale	Schuppen Mama Alter Geschlecht andere Jabberwoks
Fähigkeiten	Eier legen

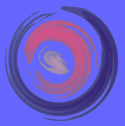


Dieses Wissen wird in eine Klasse umgesetzt und deren Attribute und Methoden näher spezifiziert:

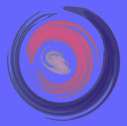


Dieses Wissen wird in eine Klasse umgesetzt und deren Attribute und Methoden näher spezifiziert:

Klasse: Jabberwok		
Attribut	Typ	Bedingungen
bekannte	`Menge`	nur Jabberwoks
schuppen	`int`	gerade, > 0
alter	`long`	in Jahren
geschlecht	`boolean`	weiblich oder männlich
mama	`Jabberwok`	weiblich, anderen unbekannt
Methode	**Typ**	**Bedingungen**
istErwachsen()	`boolean`	Wahr wenn Alter > 100
eierLegen()	`void`	istErwachsen(), weiblich



Für die Methoden muß nun ein Algorithmus festgelegt werden.



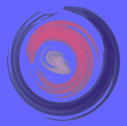
Für die Methoden muß nun ein Algorithmus festgelegt werden.

istErwachsen — Rückgabe Typ: boolean

alter $\geq$ 100 ?	
Y	N
istErwachsen = true	istErwachsen = false

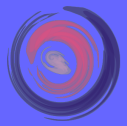
eierLegen — Rückgabe Typ: void

istErwachsen AND geschlecht == weiblich ?	
Y	N
neuen Jabberwok erzeugen	Fehler: kann keine Eier legen
zu bekannten hinzufügen	



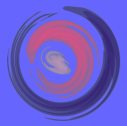
Im ersten Ansatz werden nur **Attribute** und **Methodendefinitionen** der Klasse übertragen.

```
// benötigt für Menge aller Jabberwoks
import java.util.Set

// Klassendefinition
public class Jabberwok {
    // Attribute
    int       schuppen;
    int       alter;
    boolean   geschlecht;
    Jabberwok mama;
    Set       bekannte;
    // Methoden
    boolean istErwachsen() {
        return false;
    }

    void eierLegen() {
    }
}
```

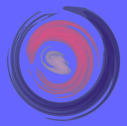


Fangen wir an die bekannten Informationen in die Klasse einzubringen.

```
boolean istErwachsen() {
  return (alter >= 100);
}

void eierLegen() {
  if (istErwachsen() && (geschlecht == true)) {
    Jabberwok kind = new Jabberwok();
    bekannte.add(kind);
  }
}
```



Das Statement `geschlecht == true` ist nicht besonders aussagekräftig. Zwar legt man fest, daß `true` für weiblich steht, dies ist aber in einem größerem Programm nicht mehr zu überblicken.



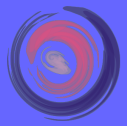
Das Statement `geschlecht == true` ist nicht besonders aussagekräftig. Zwar legt man fest, daß `true` für weiblich steht, dies ist aber in einem größerem Programm nicht mehr zu überblicken.

Führen wir deshalb zwei allgemein zugreifbare Konstanten ein:

```
public final static boolean MAENNLICH = false;
public final static boolean WEIBLICH  = true;
```

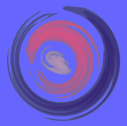
Nun läßt sich auch besser lesbarer Code schreiben:

```
geschlecht == WEIBLICH
```



3. Was ist ein Objekt

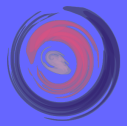
Ein Objekt ist ein Exemplar einer Klasse.



3. Was ist ein Objekt

Ein Objekt ist ein Exemplar einer Klasse.

Alle Objekte einer Klasse verfügen über die gleichen Attribute und Methoden.

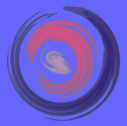


3. Was ist ein Objekt

Ein Objekt ist ein Exemplar einer Klasse.

Alle Objekte einer Klasse verfügen über die gleichen Attribute und Methoden.

Die Attributwerte eines jeden Objekts können verschieden sein.



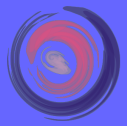
3. Was ist ein Objekt

Ein Objekt ist ein Exemplar einer Klasse.

Alle Objekte einer Klasse verfügen über die gleichen Attribute und Methoden.

Die Attributwerte eines jeden Objekts können verschieden sein.

Ein Objekt wird über den `new` - Operator erzeugt.



3. Was ist ein Objekt

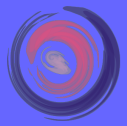
Ein Objekt ist ein Exemplar einer Klasse.

Alle Objekte einer Klasse verfügen über die gleichen Attribute und Methoden.

Die Attributwerte eines jeden Objekts können verschieden sein.

Ein Objekt wird über den `new` - Operator erzeugt.

Objekte werden über Referenzen angesprochen.



3. Was ist ein Objekt

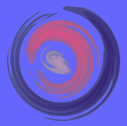
Ein Objekt ist ein Exemplar einer Klasse.

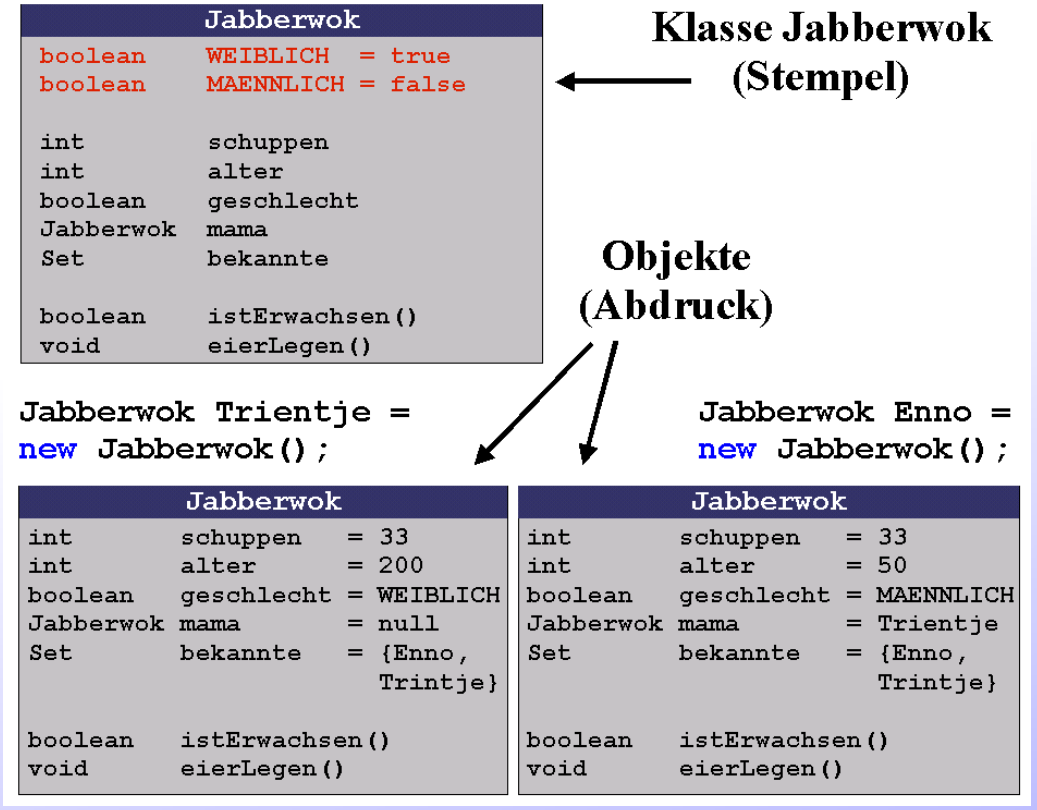
Alle Objekte einer Klasse verfügen über die gleichen Attribute und Methoden.

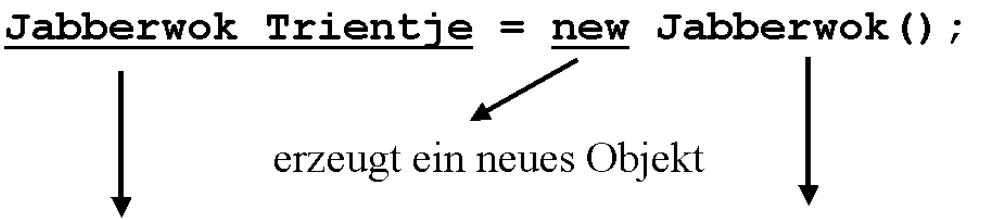
Die Attributwerte eines jeden Objekts können verschieden sein.

Ein Objekt wird über den `new` - Operator erzeugt.

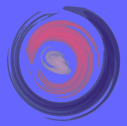
Objekte werden über Referenzen angesprochen.



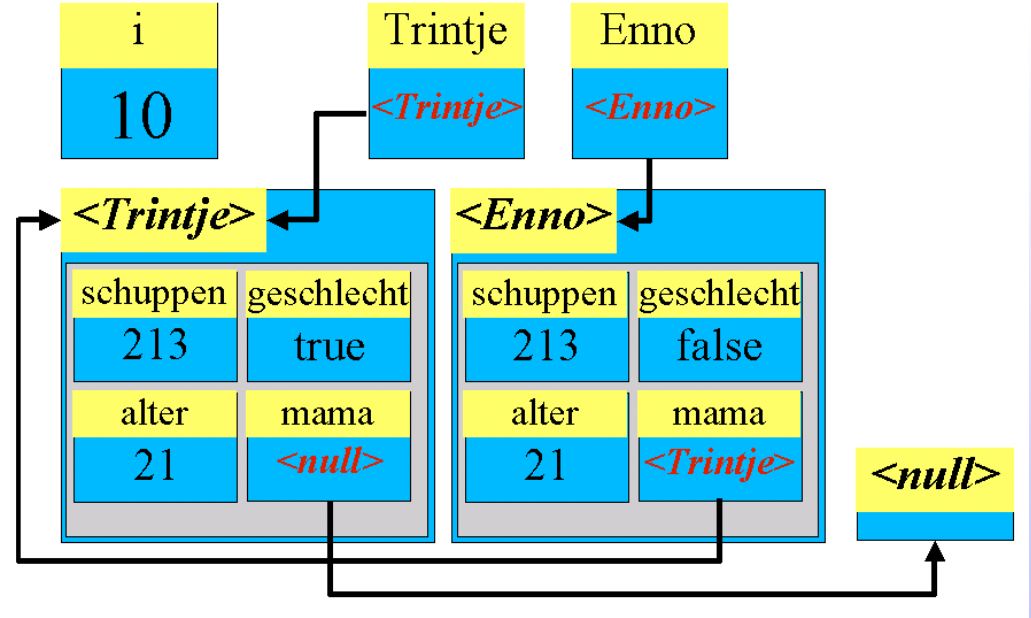




Von nun an kann der Jabberwok über die Referenz Trientje angesprochen werden.



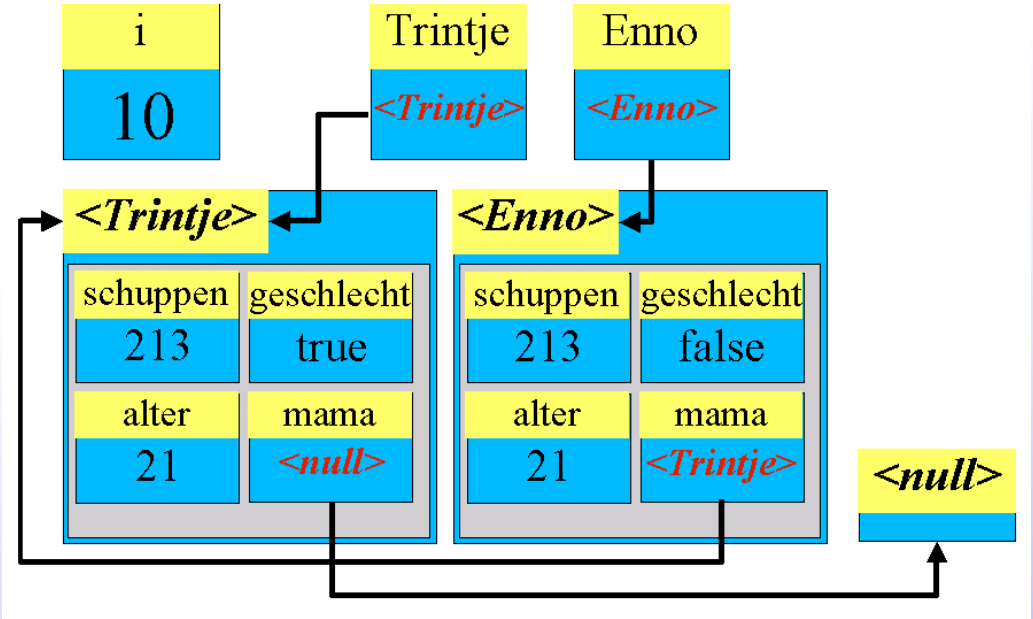
Wie kann man sich Variablen bzw. Referenzen vorstellen?



Wenn nun folgendes Statement ausgeführt wird:

```
Trintje = Enno;
```

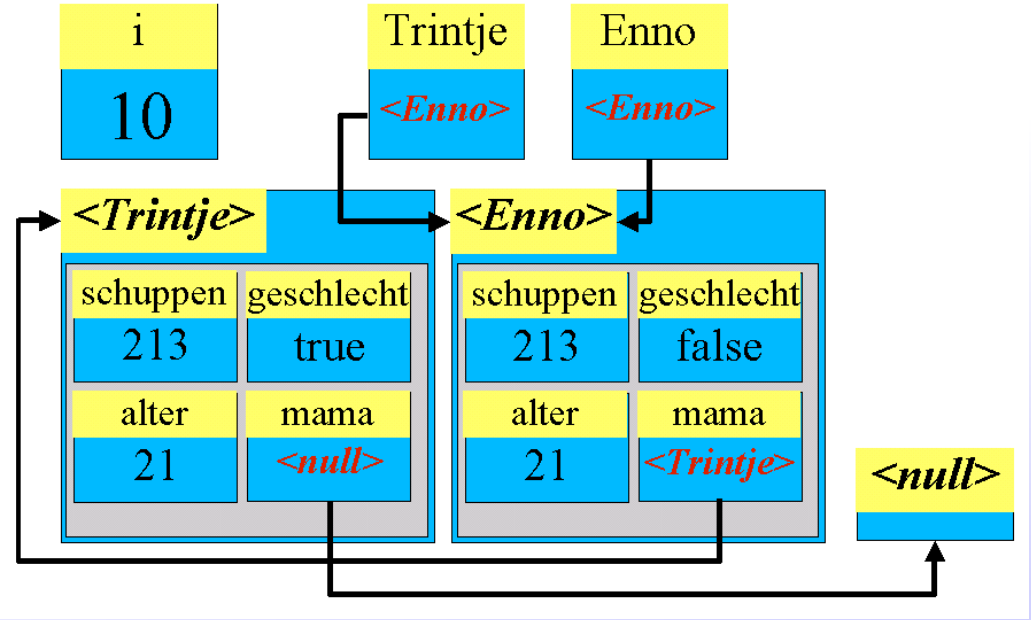
Wie kann man sich Variablen bzw. Referenzen vorstellen?



Wenn nun folgendes Statement ausgeführt wird:

```
Trintje = Enno;
```

Ändern sich nicht die Objekte, sondern nur die Referenz der Variablen Trientje wird „umgebogen“.



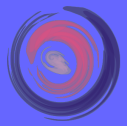
Existiert keine Referenz (Pfeil) mehr auf ein Objekt, so wird dieses Objekt entfernt (Garbage Collection).

z.B bei `Enno = Trientje`

Aufgabe:

Für ein einstöckiges Gebäude in Plattenbauweise soll eine Klassenstruktur entworfen werden.
Es sind folgende Fragen zu beantworten:

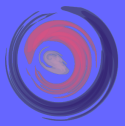
1. Sind zwei gegebene Zimmer benachbart?



Aufgabe:

Für ein einstöckiges Gebäude in Plattenbauweise soll eine Klassenstruktur entworfen werden.
Es sind folgende Fragen zu beantworten:

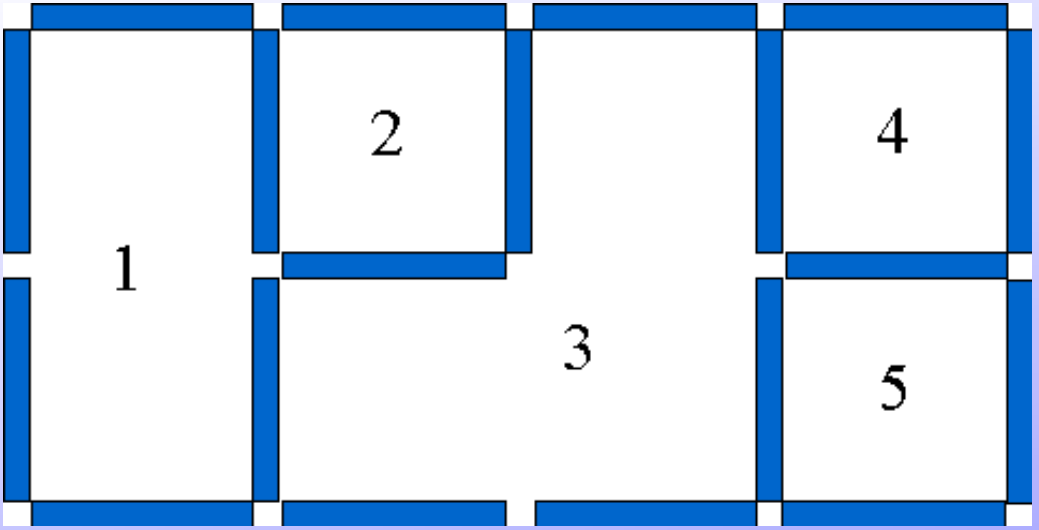
1. Sind zwei gegebene Zimmer benachbart?
2. Welche Zimmer sind zu einem gegebenen Zimmer benachbart?



Aufgabe:

Für ein einstöckiges Gebäude in Plattenbauweise soll eine Klassenstruktur entworfen werden.
Es sind folgende Fragen zu beantworten:

1. Sind zwei gegebene Zimmer benachbart?
2. Welche Zimmer sind zu einem gegebenen Zimmer benachbart?



4. Vererbung

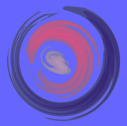
Vererbung in der Objektorientierung ist mit der evolutionären Entwicklung von Lebewesen vergleichbar.

Nehmen wir an unser Jabberwok entwickelt sich im Laufe der Zeit weiter:

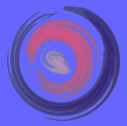
Diese Evolutionäre Stufe des Jabberwoks nennen wir einen Flabberwurm.

Dieser besitzt nun noch folgende zusätzliche Eigenschaften:

- kann feuerspeien.
- kann fliegen.
- ist mit 50 Jahren erwachsen.

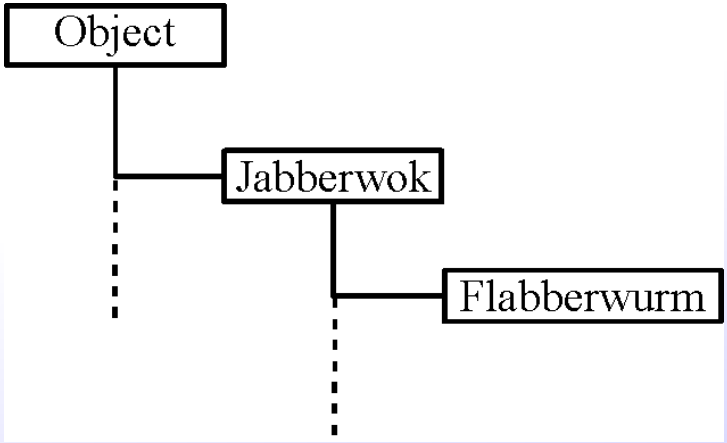


Die Vererbungsstruktur läßt sich in einem Vererbungsbaum darstellen.



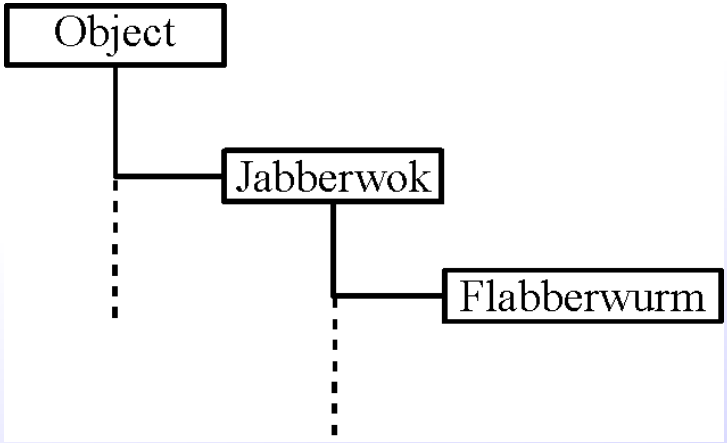
Inhalt

Die Vererbungsstruktur läßt sich in einem Vererbungsbaum darstellen.



Die Vererbungs-Beziehung zwischen Klassen wird mit dem Schlüsselwort

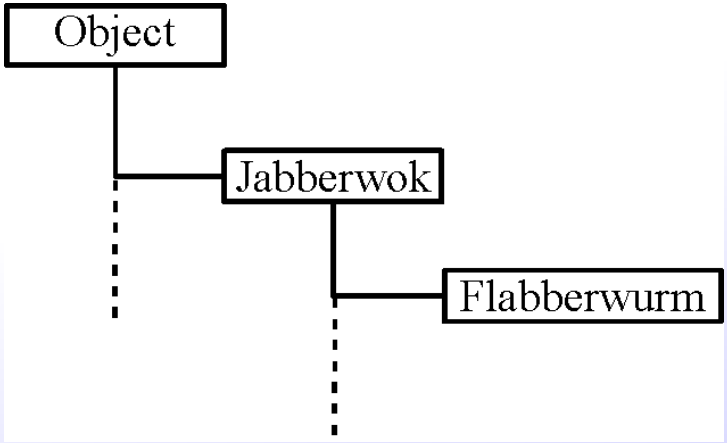
Die Vererbungsstruktur läßt sich in einem Vererbungsbaum darstellen.



Die Vererbungs-Beziehung zwischen Klassen wird mit dem Schlüsselwort extends angegeben.

Die Klassendefinition des Flabberwurms lautet dann:

Die Vererbungsstruktur läßt sich in einem Vererbungsbaum darstellen.



Die Vererbungs-Beziehung zwischen Klassen wird mit dem Schlüsselwort extends angegeben.

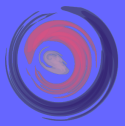
Die Klassendefinition des Flabberwurms lautet dann:

```
public class
Flabberwurm extends Jabberwok {
```

Der Flabberwurm kann auf alle Attribute und Methoden eines Jabberwoks zugreifen.

Man kann diese Beziehung auch so ausdrücken:

Ein Flabberwurm

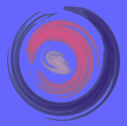


Der Flabberwurm kann auf alle Attribute und Methoden eines Jabberwoks zugreifen.

Man kann diese Beziehung auch so ausdrücken:

Ein Flabberwurm ist ein Jabberwok

Die weiteren Eigenschaften werden nun in der Klasse Flabberwurm definiert.



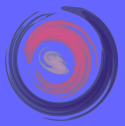
Der Flabberwurm kann auf alle Attribute und Methoden eines Jabberwoks zugreifen.

Man kann diese Beziehung auch so ausdrücken:

Ein Flabberwurm ist ein Jabberwok

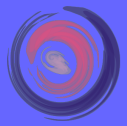
Die weiteren Eigenschaften werden nun in der Klasse Flabberwurm definiert.

Klasse	ist eine	Klasse
Jabberwok	Oberklasse superclass Generalisierung	Flabberwurm
Flabberwurm	Unterklasse subclass Spezialisierung	Jabberwok



```
public class Flabberwurm extends Jabberwok {
  // neue Methoden
  void fliegen() {
  }

  void feuerspeien() {
    if (istErwachsen()) {
      // speie Feuer
    }
  }

  // überschriebene Methode
  boolean istErwachsen() {
    return (alter >= 50);
  }
}
```

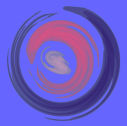
Die Methode `istErwachsen()` ist in der Oberklasse Jabberwok bereits definiert.



```
public class Flabberwurm extends Jabberwok {
  // neue Methoden
  void fliegen() {
  }

  void feuerspeien() {
    if (istErwachsen()) {
      // speie Feuer
    }
  }

  // überschriebene Methode
  boolean istErwachsen() {
    return (alter >= 50);
  }
}
```

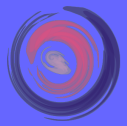
Die Methode `istErwachsen()` ist in der Oberklasse Jabberwok bereits definiert.

Sie wird hier neu definiert und überschreibt die Methode aus der Klasse Jabberwok.



Bei der Evolution kann es durchaus sein, daß sich die Eigenschaften zweier Wesen weitervererben.

Diese Mehrfachvererbung (Das Erben von zwei Klassen) wird von Java nicht unterstützt.



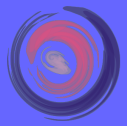
Inhalt

Bei der Evolution kann es durchaus sein, daß sich die Eigenschaften zweier Wesen weitervererben.

Diese Mehrfachvererbung (Das Erben von zwei Klassen) wird von Java nicht unterstützt.

Als Ersatz gibt es das Konzept der Schnittstellenvererbung.

Schnittstellen können von Klassen erfüllt werden.



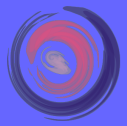
Bei der Evolution kann es durchaus sein, daß sich die Eigenschaften zweier Wesen weitervererben.

Diese Mehrfachvererbung (Das Erben von zwei Klassen) wird von Java nicht unterstützt.

Als Ersatz gibt es das Konzept der Schnittstellenvererbung.

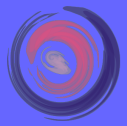
Schnittstellen können von Klassen erfüllt werden.

Eine Klasse kann mehrere Schnittstellen erfüllen.



4.1. Schnittstelle - Interface

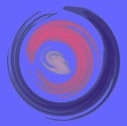
Eine Schnittstelle ist die Deklaration von Methoden.



4.1. Schnittstelle - Interface

Eine Schnittstelle ist die Deklaration von Methoden.

Die Methoden einer Schnittstelle enthalten keinen Rumpf.

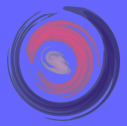


4.1. Schnittstelle - Interface

Eine Schnittstelle ist die Deklaration von Methoden.

Die Methoden einer Schnittstelle enthalten keinen Rumpf.

In einer Schnittstelle können keine Variablen definiert werden.



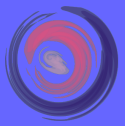
4.1. Schnittstelle - Interface

Eine Schnittstelle ist die Deklaration von Methoden.

Die Methoden einer Schnittstelle enthalten keinen Rumpf.

In einer Schnittstelle können keine Variablen definiert werden.

Da Schnittstellen keine Funktionalität haben kann auch kein Exemplar einer Schnittstelle erzeugt werden.



4.1. Schnittstelle - Interface

Eine Schnittstelle ist die Deklaration von Methoden.

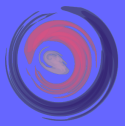
Die Methoden einer Schnittstelle enthalten keinen Rumpf.

In einer Schnittstelle können keine Variablen definiert werden.

Da Schnittstellen keine Funktionalität haben kann auch kein Exemplar einer Schnittstelle erzeugt werden.

Hier eine Schnittstelle.

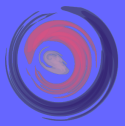
```
public interface Fortplanzungsfaehig {

    // Methoden Deklaration
    void fortpflanzen();
}
```



Nun erweitern wir den Jabberwok um diese Schnittstelle:

```
public class Jabberwok implements
                Fortplanzungsfaehig {
    ...
    public void fortpflanzen() {
        eierlegen();
    }
}
```

Bezüglich des Fortpflanzens können wir nun einen Jabberwok genauso behandeln, wie jedes andere Wesen, daß die Schnittstelle `Fortplanzungsfaehig` erfüllt.



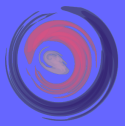
Nun erweitern wir den Jabberwok um diese Schnittstelle:

```
public class Jabberwok implements
                Fortplanzungsfaehig {
    ...
    public void fortpflanzen() {
        eierlegen();
    }
}
```

Bezüglich des Fortpflanzens können wir nun einen Jabberwok genauso behandeln, wie jedes andere Wesen, daß die Schnittstelle `Fortplanzungsfaehig` erfüllt.

ist ein

Eine Klasse kann von genau einer anderen Klasse erben (deren Oberklasse). Wird keine Klasse angegeben so leitet sich die Klasse direkt von der Klasse `Object` ab.



Nun erweitern wir den Jabberwok um diese Schnittstelle:

```
public class Jabberwok implements
                Fortplanzungsfaehig {
    ...
    public void fortpflanzen() {
        eierlegen();
    }
}
```

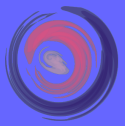
Bezüglich des Fortpflanzens können wir nun einen Jabberwok genauso behandeln, wie jedes andere Wesen, daß die Schnittstelle `Fortplanzungsfaehig` erfüllt.

ist ein

Eine Klasse kann von genau einer anderen Klasse erben (deren Oberklasse). Wird keine Klasse angegeben so leitet sich die Klasse direkt von der Klasse `Object` ab.

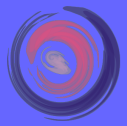
verhält sich wie

Eine Klasse kann beliebig viele Schnittstellen erfüllen. Damit kann eine bestimmte Funktionaltät der Klasse beschrieben werden.



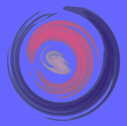
Es ist möglich folgende Referenzen zu erzeugen:

```
// alfred ein neuer Flabberwurm
Flabberwurm alfred = new Flabberwurm();
```



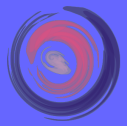
Es ist möglich folgende Referenzen zu erzeugen:

```
// alfred ein neuer Flabberwurm
Flabberwurm alfred = new Flabberwurm();

// alfred etwas allgemeiner
Jabberwok hugo = alfred;
```



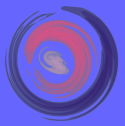
Es ist möglich folgende Referenzen zu erzeugen:

```
// alfred ein neuer Flabberwurm
Flabberwurm alfred = new Flabberwurm();

// alfred etwas allgemeiner
Jabberwok hugo = alfred;

// alfred nur Fortpflanzungsfaehig
Fortplanzungsfaehig egon = alfred;
```



Es ist möglich folgende Referenzen zu erzeugen:

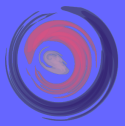
```
// alfred ein neuer Flabberwurm
Flabberwurm alfred = new Flabberwurm();

// alfred etwas allgemeiner
Jabberwok hugo = alfred;

// alfred nur Fortpflanzungsfaehig
Fortplanzungsfaehig egon = alfred;

// wie oben aber über Umweg
Fortpflanzungsfaehig jean = hugo;
```



Es ist möglich folgende Referenzen zu erzeugen:

```
// alfred ein neuer Flabberwurm
Flabberwurm alfred = new Flabberwurm();

// alfred etwas allgemeiner
Jabberwok hugo = alfred;

// alfred nur Fortpflanzungsfaehig
Fortplanzungsfaehig egon = alfred;

// wie oben aber über Umweg
Fortpflanzungsfaehig jean = hugo;
```

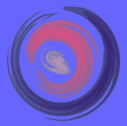
Diese Zuweisungen sind vergleichbar mit einem *widening cast*



Aufgabe:

Stellt die folgende Vererbungstruktur grafisch dar.

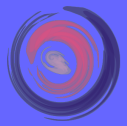
```
class A
interface E
class B extends A
class C implements E
interface D extends E
class Weired extends B implements D
```



Aufgabe:
Stellt die folgende Vererbungstruktur grafisch dar.

```
class A
interface E
class B extends A
class C implements E
interface D extends E
class Weired extends B implements D

public D m1(A a, C c);
public B m2(Weired w, E e);
public Weired m3(A a, E e);
```

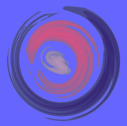
Die Methodendeklarationen werden für die folgende Aufgabe benötigt.



Sind folgende Ausdrücke syntaktisch korrekt?
Wenn Nein, warum?

```
1)  D d = m1(a, c);
2)  E e = m1(a, c);
3)  D d = m1(weired, c);
4)  E e = m1(weired, weired);
5)  D d = m1(m2(weired, e), c);
6)  E e = m1(m2(weired, weired), c);
7)  B b = m3(m2(weired, d), weired);
8)  E e = m2(m3(b, d), m1(a, m3(a, e)));
```

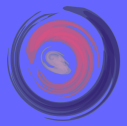
Alle Kleinbuchstaben stehen für Variablen, die ein Objekt der entsprechenden Klasse referenzieren.



5. Objekterzeugung - Konstruktor

5.1. Konstruktor

Ein Konstruktor wird zum Initialisieren eines Objekts verwendet.

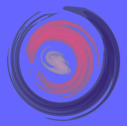


5. Objekterzeugung - Konstruktor

5.1. Konstruktor

Ein Konstruktor wird zum Initialisieren eines Objekts verwendet.

Ein Konstruktor kann, wie eine Methode, Parameter übergeben bekommen.



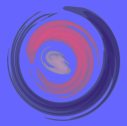
5. Objekterzeugung - Konstruktor

5.1. Konstruktor

Ein Konstruktor wird zum Initialisieren eines Objekts verwendet.

Ein Konstruktor kann, wie eine Methode, Parameter übergeben bekommen.

Wird kein Konstruktor in der Klasse definiert, wird ein sogenannter Standardkonstruktor (leere Parameterliste) verwendet.



5. Objekterzeugung - Konstruktor

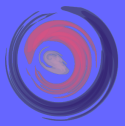
5.1. Konstruktor

Ein Konstruktor wird zum Initialisieren eines Objekts verwendet.

Ein Konstruktor kann, wie eine Methode, Parameter übergeben bekommen.

Wird kein Konstruktor in der Klasse definiert, wird ein sogenannter Standardkonstruktor (leere Parameterliste) verwendet.

Wird ein Konstruktor definiert, kann der Standardkonstruktor nicht mehr verwendet werden.



5. Objekterzeugung - Konstruktor

5.1. Konstruktor

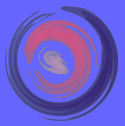
Ein Konstruktor wird zum Initialisieren eines Objekts verwendet.

Ein Konstruktor kann, wie eine Methode, Parameter übergeben bekommen.

Wird kein Konstruktor in der Klasse definiert, wird ein sogenannter Standardkonstruktor (leere Parameterliste) verwendet.

Wird ein Konstruktor definiert, kann der Standardkonstruktor nicht mehr verwendet werden.

Es können beliebig viele Konstruktoren in einer Klasse definiert werden.



5. Objekterzeugung - Konstruktor

5.1. Konstruktor

Ein Konstruktor wird zum Initialisieren eines Objekts verwendet.

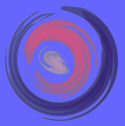
Ein Konstruktor kann, wie eine Methode, Parameter übergeben bekommen.

Wird kein Konstruktor in der Klasse definiert, wird ein sogenannter Standardkonstruktor (leere Parameterliste) verwendet.

Wird ein Konstruktor definiert, kann der Standardkonstruktor nicht mehr verwendet werden.

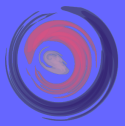
Es können beliebig viele Konstruktoren in einer Klasse definiert werden.

Sie müssen sich durch ihre Parameterlisten unterscheiden.



Die Initialisierung eines Objekts findet in folgender Reihenfolge statt:

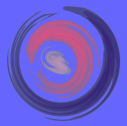
1. Anlegen der Variablen und setzten der dort angegebenen Werte.
   ```
   private int a = 1;
   ```



Die Initialisierung eines Objekts findet in folgender Reihenfolge statt:

1. Anlegen der Variablen und setzten der dort angegebenen Werte.
   ```
   private int a = 1;
   ```

2. Ausführen eines Initialisierungsblocks.
   ```
   { a = 2; }
   ```



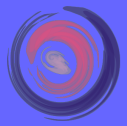
Die Initialisierung eines Objekts findet in folgender Reihenfolge statt:

1. Anlegen der Variablen und setzten der dort angegebenen Werte.
   ```
   private int a = 1;
   ```

2. Ausführen eines Initialisierungsblocks.
   ```
   { a = 2; }
   ```

3. Ausführen des Konstruktors.

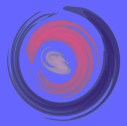
Beim Initialisieren eines Objektes werden zuerst die Oberklassen Initialisierungen durchgeführt.



Soll ein anderer Konstruktor als der Standardkonstruktor der Oberklasse ausgeführt werden, so kann dieser mit dem Aufruf:

`super(`*`parameterliste`*`)`

angegeben werden. (Stichwort: Constructor chaining)



Soll ein anderer Konstruktor als der Standardkonstruktor der Oberklasse ausgeführt werden, so kann dieser mit dem Aufruf:

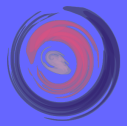
`super(`*`parameterliste`*`)`

angegeben werden. (Stichwort: Constructor chaining)

Soll ein anderer Konstruktor dieser Klasse mitaufgerufen werden, so dies mit

`this(`*`parameterliste`*`)`

tun.



Soll ein anderer Konstruktor als der Standardkonstruktor der Oberklasse ausgeführt werden, so kann dieser mit dem Aufruf:

`super(`*`parameterliste`*`)`

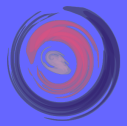
angegeben werden. (Stichwort: Constructor chaining)

Soll ein anderer Konstruktor dieser Klasse mitaufgerufen werden, so dies mit

`this(`*`parameterliste`*`)`

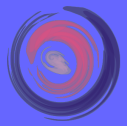
tun.

`super(...)` und `this(...)` dürfen nur als erstes statement im Konstruktor verwendet werden.



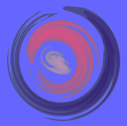
Beispiel:

```
public class Jabberwok ... {

  private Jabberwok() {
  }

  private Jabberwok(Jabberwok mama) {
    this.mama = mama;
  }

  private Jabberwok(Jabberwok mama,
                    int schuppen,
                    boolean geschlecht) {
    this(mama);
    this.setSchuppen(schuppen);
    this.geschlecht = geschlecht;
  }
}
```



6. Zugriff: private, protected, public

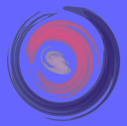
Über sogenannte Zugriffsrestriktionen kann man den Variablen/ Methodenzugriff, von anderen Objekten aus, einschränken.



6. Zugriff: private, protected, public

Über sogenannte Zugriffsrestriktionen kann man den Variablen/ Methodenzugriff, von anderen Objekten aus, einschränken.

`public boolean getGeschlecht()`
Jeder kann das Geschlecht eines Jabberwoks erkennen. (z.B. Männchen haben eine rote Nase).

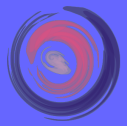


6. Zugriff: private, protected, public

Über sogenannte Zugriffsrestriktionen kann man den Variablen/ Methodenzugriff, von anderen Objekten aus, einschränken.

`public boolean getGeschlecht()`
Jeder kann das Geschlecht eines Jabberwoks erkennen. (z.B. Männchen haben eine rote Nase).

`protected int getAlter()`
Nur ein Jabberwok (oder eine Spezialisierung) kann das Alter eines Jabberwoks wissen.



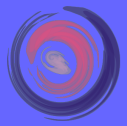
6. Zugriff: private, protected, public

Über sogenannte Zugriffsrestriktionen kann man den Variablen/ Methodenzugriff, von anderen Objekten aus, einschränken.

`public boolean getGeschlecht()`
Jeder kann das Geschlecht eines Jabberwoks erkennen. (z.B. Männchen haben eine rote Nase).

`protected int getAlter()`
Nur ein Jabberwok (oder eine Spezialisierung) kann das Alter eines Jabberwoks wissen.

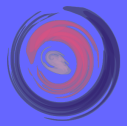
`private Jabberwok getMama()`
Nur ein reinrassiger Jabberwok kann die Mutter des Jabberwoks in Erfahrung bringen.



7. static

Alle Jabberwoks sind allen anderen Jabberwoks bekannt.

Das bedeutet, daß sobald ein neuer Jabberwok geboren wird dies allen anderen Jabberwoks mitgeteilt werden muß.

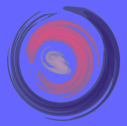


7. static

Alle Jabberwoks sind allen anderen Jabberwoks bekannt.

Das bedeutet, daß sobald ein neuer Jabberwok geboren wird dies allen anderen Jabberwoks mitgeteilt werden muß.

Es liegt nahe eine Liste aller Jabberwoks an einem zentralen Ort zu speichern.



7. static

Alle Jabberwoks sind allen anderen Jabberwoks bekannt.

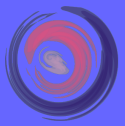
Das bedeutet, daß sobald ein neuer Jabberwok geboren wird dies allen anderen Jabberwoks mitgeteilt werden muß.

Es liegt nahe eine Liste aller Jabberwoks an einem zentralen Ort zu speichern.

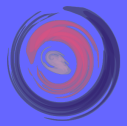
Ein solcher zentraler Ort wäre die Klasse Jabberwok.

```
public static Set bekannte = new HashSet();
```

Es wird nur ein mal Speicherplatz angelegt, auf den alle Zugreifen können.

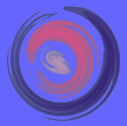


Es ist nicht sinnvoll Jabberwoks einfach so zu erzeugen, da ein Jabberwok von seiner Mutter geboren wird.



Es ist nicht sinnvoll Jabberwoks einfach so zu erzeugen, da ein Jabberwok von seiner Mutter geboren wird.

Dies bedeutet, die Klasse Jabberwok darf keinen öffentlichen Konstruktor haben.

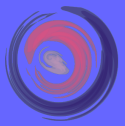


Es ist nicht sinnvoll Jabberwoks einfach so zu erzeugen, da ein Jabberwok von seiner Mutter geboren wird.

Dies bedeutet, die Klasse Jabberwok darf keinen öffentlichen Konstruktor haben.

Wenn man aber Jabberwoks nur über die Mutter erzeugen kann, stellt sich die bekannte Frage:

Was war zuerst: Jabberwok oder Ei?



Es ist nicht sinnvoll Jabberwoks einfach so zu erzeugen, da ein Jabberwok von seiner Mutter geboren wird.

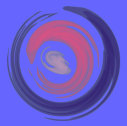
Dies bedeutet, die Klasse Jabberwok darf keinen öffentlichen Konstruktor haben.

Wenn man aber Jabberwoks nur über die Mutter erzeugen kann, stellt sich die bekannte Frage:

Was war zuerst: Jabberwok oder Ei?

Diese Schwierigkeit können wir mit einer statischen Initialisierung umgehen, in der wir einen Ur-Jabberwok erzeugen.

```
static {
  bekannte.add(
      new Jabberwok(null, 3, WEIBLICH));
}
```



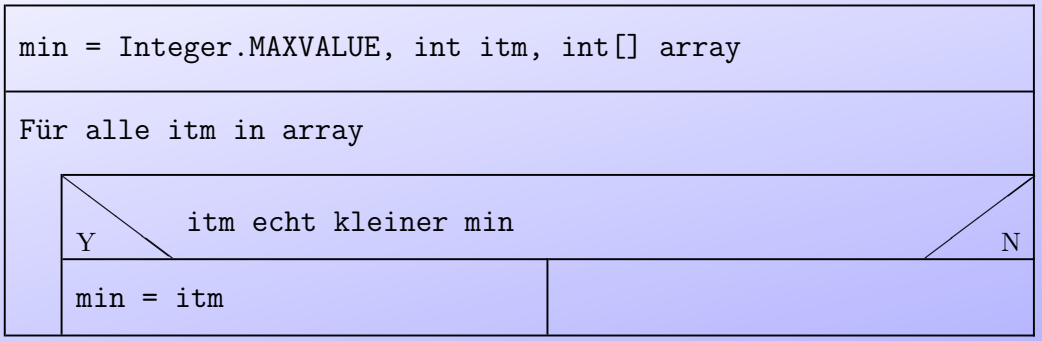
8. Struktogramme

Die folgenden vier Struktogramme sollen in korrekten Java code umgesetzt werden.

Tragt bei Name: einen Euch sinnvoll erscheinenden Namen für die Methode ein.

Welche Daten müssen als Parameter übergeben werden?

Name: — `Rückgabe: min`



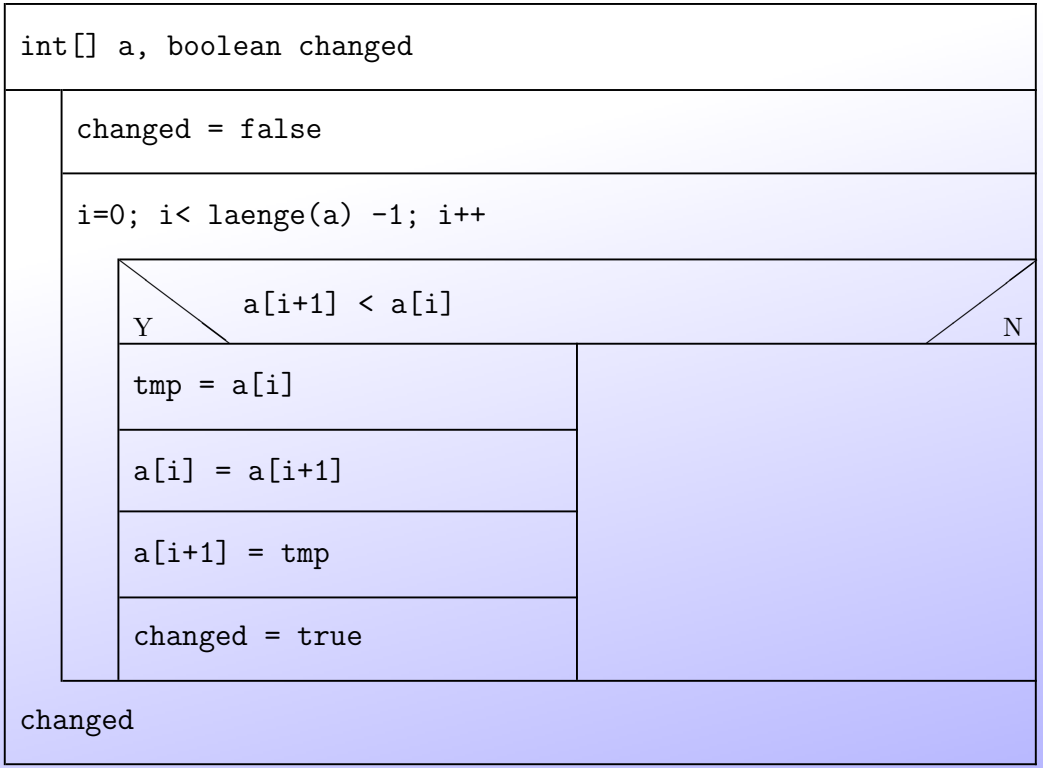
Name: — Rückgabe : min

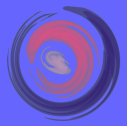
```
List lst, Object itm, Object min = null, Comparator cmp
Für alle itm in lst
    itm echt kleiner min
    Y: min = itm
    N:
```

Name: — Rückgabe : m

```
int[] a, int itm, int s = 0
Für alle itm in a
    s += itm
m = s / laenge(a)
```

Name: — Rückgabe : void





Inhalt

9. Fehlerbehandlung unter Java

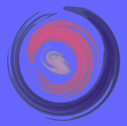
Normalerweiser sagt der Rückgabewert einer Methode etwas über das Gelingen oder den aufgetretenen Fehler aus.

In Java gibt es das Konzept der Ausnahmen-Behandlung (exception handling).

Dies basiert darauf einem try & error Prinzip:

Versuche eine fehleranfällige Aktion auszuführen und versuche im Fehlerfall angemessen zu reagieren.

Entweder wird die Ausnahme an der Stelle behandelt, an der sie entsteht, oder zu einer aufrufenden Methode *„weitergeworfen“*.



9.1. Behandeln in aktueller Methode

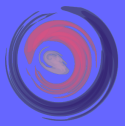
```
try {
  fehlerVerursachendeMethode();
} catch (Exception exc) {
  // Auf Ausnahmen reagieren
}
```

In einem `try` Block stehen alle Methodenaufrufe, die zu Ausnahmezuständen führen können.

In dem `catch` Block werden aufgetretene Ausnahmen behandelt.

Ausnahmen können anhand ihrer Klasse unterschieden werden.

Ein `catch` Block behandelt eine Klasse von Ausnahmen. Es können mehrere `catch` Blöcke für verschiedene Klassen von Ausnahmen stehen.

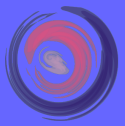


Ein Beispiel für eine komplexere Ausnahmenbehandlung:

```
try {
  fehlerVerursachendeMethode1();
  fehlerVerursachendeMethode2();
  fehlerVerursachendeMethode3();
} catch (InvalidArgumentException exc) {
  // Auf Ausnahme reagieren
} catch (ArrayIndexOutOfBoundsException exc) {
  // Auf Ausnahme reagieren
} catch (Exception exc) {
  // Auf Ausnahme reagieren
} finally {
  // aufräumen!
}
```

finally

Dieser Block wird immer ausgeführt, unabhängig davon, ob eine Ausnahme auftrat oder nicht.



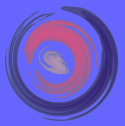
Eine Ausnahme kann eine beliebige Unterklasse von `Throwable` sein.

Eine Methode kann eine Ausnahme erzeugen und diese auswerfen:

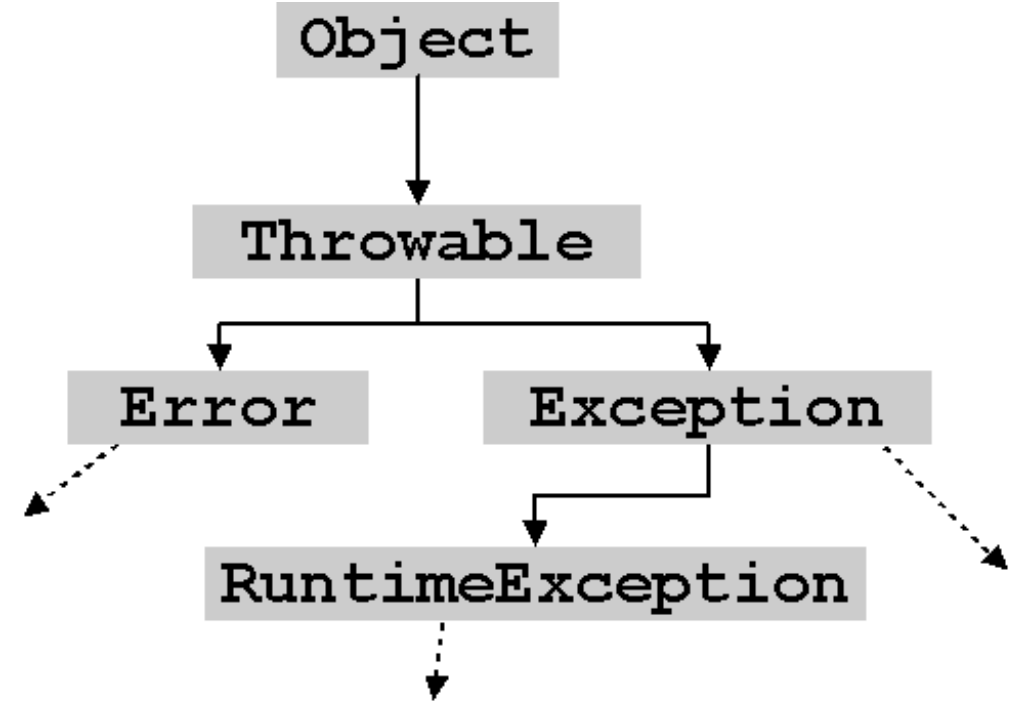
```
Exception exc = new Exception();
throw exc;
```

Handelt es sich bei der Ausnahme nicht um eine Unterklasse von `RuntimeException`, so muß in der Methodendeklaration angezeigt werden, daß die Methode diese Ausnahme verursachen kann.

```
public void m1() throws Exception {
```



Der Vererbungsbaum



Throwable Alle Exemplare von Throwable können über den `try-catch-finally` Mechanismus behandelt werden. Methoden können diese erzeugen und auswerfen. Methoden die Ausnahmen erzeugen, müssen dies in der Methodendeklaration anzeigen.

Error Exemplare dieser Klasse könne erzeugt und ausgeworfen werden. Man sollte keine Error Ausnahmen abfangen.

Exception Normale Ausnahmen, sie können erzeugt, geworfen und abgefangen werden.

RuntimeException Wie Exception, aber diese Ausnahmen müssen nicht in der Methodendeklaration angezeigt werden.