

Methoden und Klassen

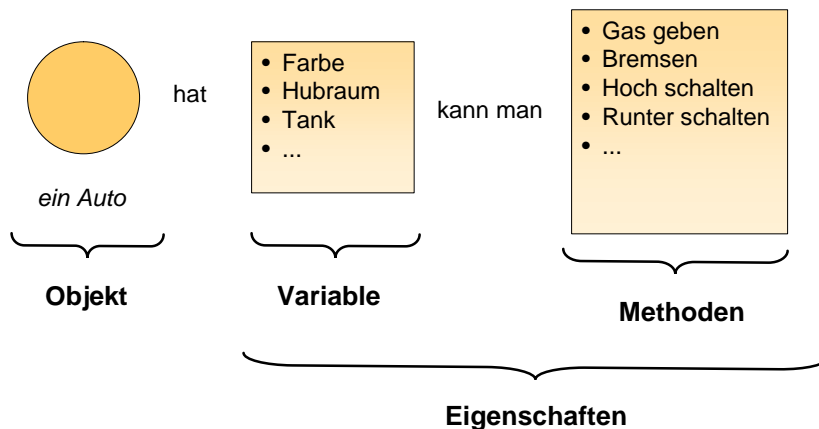
Silke Trißl

Wissensmanagement in der Bioinformatik



Objektorientierte Programmierung (OOP)

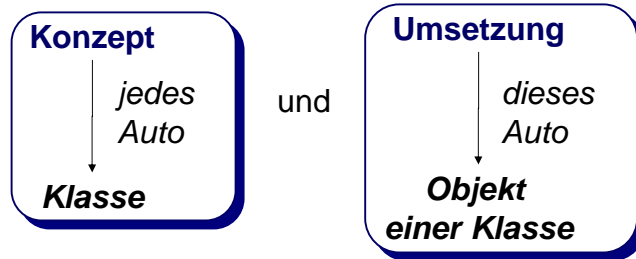
Vorstellung wie in der realen Welt:



Objektorientierte Programmierung

1. Abstraktion

→ klare Trennung zwischen :



oder auch :

alle Objekte mit gleichen Eigenschaften

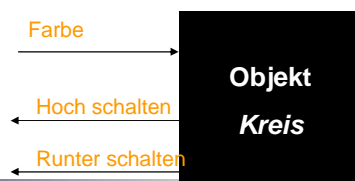
gehören zu einer Klasse → **Klassenbildung**

Objektorientierte Programmierung

1. Abstraktion

2. Kapselung

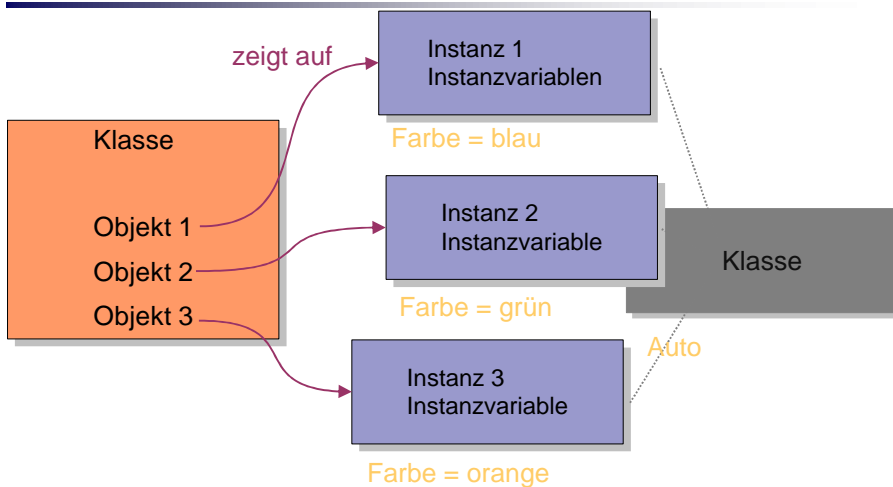
- Zugriff auf die Attribute eines Objekts i.d.R. nur über seine Methoden
- Verbergen unwichtiger Details (Black box)



Vokabular

- Klasse `class`
 - Vorlage von der das eigentliche Objekt erzeugt wird
- Objekt
 - wird aus einer Klasse konstruiert
 - ? Instanz(en) einer Klasse
 - besitzt Instanzvariablen und Methoden
 - hat einen aktuellen Zustand, der durch Methodenaufrufe verändert werden kann

Klassen und Objekte



Eigene Klassen

- werden für komplexe Java-Anwendungen benötigt
- komplexe Anwendungen
 - eine Klasse mit main-Methode
 - viele zusätzliche Klassen (eigene oder Java-Library)

Eigene Klassen - cont.

```
class NameDerKlasse {  
  
    Konstruktoren  
    ...  
  
    Methoden  
    ...  
  
    Instanzfelder  
    ...  
}
```

Konstruktor

```
class Auto {  
  
    // Konstruktor  
    public Auto() {  
  
    }  
  
    public Auto(String Auto_Marke) {  
        marke = Auto_Marke;  
    }  
}
```

Objekte instanzieren

```
public class MainProg {  
    public static void main(String[] args)  
    {  
        // Das Hauptprogramm  
  
        new Auto();  
    }  
}
```

```
public class Auto {  
    public Auto() {  
  
    }  
    ...  
    // Methoden  
    // Instanzvariablen  
}
```

Initialisierung von Objekten

- Konstruktor

- konstruiert und initialisiert neue Instanz
- gleicher Name wie Klasse
- immer in Verbindung mit **new**

```
new Klassenname(Variablen);
```

- Objektvariable (Zeiger)

- Objektvariable ist vom Typ des Objekts

```
Klassenname instanzvariable =  
    new Klassenname(Variablen);
```

```
Auto bmw = new Auto();
```

Methoden einer Klasse aufrufen

- Ausgeben oder Verändern eines Zustandes durch eine Methode

```
instanzvariable.methodenname(variablen);
```

- Beispiel

```
Auto bmw = new Auto();  
bmw.setFarbe("Blau");
```

Methoden – Klassen

- Methoden kommen in Klassen vor

```
class Klassenname {  
    public static void main(String[] args) {  
        // lokale Variable  
    }  
    public Rückgabetyt methode1() {  
        // lokale Variable  
    }  
    private Rückgabetyt methode2() {  
        // lokale Variable  
    }  
}
```

Methoden – Beispiel

```
class Auto {  
    ...  
    Gang hoch_schalten() {  
        ...  
    }  
    Gang runter_schalten() {  
        ...  
    }  
    Gang einlegen() {  
        ...  
    }  
    entkuppeln() {  
        ...  
    }  
}
```

Methoden – wo sollen sie sichtbar sein

- public
 - sichtbar für andere Klassen
- private
 - **nicht** sichtbar für andere Klassen, auch nicht der Subklasse
- protected
 - gleich wie private, aber Subklasse kann darauf zugreifen
- ohne Modifier
 - sichtbar für das Paket – das Standardverhalten

Methoden - Rückgabetyt

- void
 - ohne Rückgabewert
 - nur Berechnung
 - sofortige Ausgabe
- primitiver Typ
 - int, long, short, byte, float, double, boolean, char
 - benötigt einen entsprechenden Rückgabewert
- jeder Typ
 - String, Date

Methoden – Beispiel in Java

```
class Auto {  
    ...  
    public int hoch_schalten() {  
        ...  
    }  
    pu  
}  
pu  
}  
public boolean entkuppeln() {  
    ...  
}  
}
```

Jetzt machen die Methoden
noch nichts!

Variablen deklarieren

- In der Methode – nur die Methode kennt sie

Methoden – Beispiel in Java

```
class Auto {  
  
    public int hoch_schalten() {  
        int gang = gang + 1;  
    }  
    pu  
  
    }  
    pu  
  
    }  
    public boolean entkuppeln() {  
        int gang = null;  
    }  
  
}
```

Schlechte Idee für den Gang!

Variablen deklarieren

- In der Methode – nur die Methode kennt sie
- In der Klasse – alle Methoden kennen sie
 - public
 - private

Methoden – Beispiel in Java

```
class Auto {
    private int gang;
    public int hoch_schalten() {
        gang = gang + 1;
    }
    public int runter_schalten() {
        gang = gang - 1;
    }
    public boolean einlegen(_gang) {
        gang = _gang;
    }
    public boolean entkuppeln() {
        gang = null;
    }
}
```

Variablen deklarieren

- In der Methode – nur die Methode kennt sie
- In der Klasse – alle Methoden kennen sie
 - public
 - private
- Der Unterschied zwischen
 - Objektvariable: Deklariert mit `static`
 - Instanzvariable: Deklariert ohne `static`

Methoden – Beispiel in Java

```
class main {  
    public static main ... {  
        Auto bmw = new Auto();  
        Auto ente = new Auto();  
  
        bmw.einlegen(2);  
        ente.einlegen(4);  
  
        bmw.hochschalten();  
        ente.hochschalten();  
    }  
}
```

gang ist eine Instanzvariable
(ohne static)

bmw.get_gang = 2
ente.get_gang = 4

```
    public int hoch_schalten() {  
        gang = gang + 1;  
    }  
    public boolean einlegen(_gang) {  
        gang = _gang;  
    }  
    ...  
}
```

Silke Trißl: Bioinformatik für Biophysiker

Methoden – Beispiel in Java

```
class main {  
    public static main ... {  
        Auto bmw = new Auto();  
        Auto ente = new Auto();  
  
        bmw.einlegen(4);  
        ente.einlegen(5);  
  
        bmw.hochschalten();  
        ente.hochschalten();  
    }  
}
```

gang ist eine Klassenvariable
(mit static)

bmw.get_gang = 4
ente.get_gang = 5

```
    public int hoch_schalten() {  
        gang = gang + 1;  
    }  
    public boolean einlegen(_gang) {  
        gang = _gang;  
    }  
    ...  
}
```

Silke Trißl: Bioinformatik für Biophysiker

Objektorientierte Programmierung

1. Abstraktion

2. Kapselung

3. Vererbung

Rechteck

- Kantenlänge a
- Kantenlänge b
- Position X
- Position Y

Quadrat

- Kantenlänge
- Position X
- Position Y

Kreis

- Radius
- Position X
- Position Y

Objektorientierte Programmierung

1. Abstraktion

2. Kapselung

3. Vererbung

Zeichenobjekt

- Position X
- Position Y

Rechteck

- Kantenlänge a
- Kantenlänge b

Quadrat

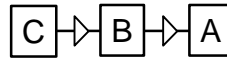
- Kantenlänge

Kreis

- Radius

Vererbung

- Konsequenz aus Spezialisierung
- Eine Unterklasse erbt **alle Eigenschaften** ihrer Oberklasse:
 - Eine Unterklasse **erbt alle Attribute** ihrer Oberklasse
 - Eine Unterklasse **erbt alle Methoden** ihrer Oberklasse
- Vererbungsbeziehung ist transitiv
 - eine Unterklasse erbt die Variablen und Methoden **aller** darüber liegenden Oberklassen
 - wenn C Unterklasse von B ist und B Unterklasse von A, dann erbt C auch von A



Vererbung

■ Syntax

```
class Unterklasse extends Oberklasse {  
...  
}
```

■ Beispiel

```
class Zeichenobjekt {  
...  
}
```

```
class Kreis extends Zeichenobjekt {  
...  
}
```



Aufgabe: Klassen erstellen

- Erstellt eine Klasse 'Kreis', die mit einem Radius instanziiert wird.
- Die Klasse soll
 - den Durchmesser zurückgeben
 - den Umfang berechnen ($2 * r * p$)
 - den Flächeninhalt berechnen ($r^2 * p$)
- Schreibt eine Main-Methode, mit der
 - von Kreis 1 ($r = 5$) der Durchmesser berechnet werden soll
 - von Kreis 2 ($r = 8$) der Flächeninhalt
 - von Kreis 3 ($r = 9$) der Umfang

Java Library - Vorhandene Klassen und Packages

- Jede Klasse gehört zu einem package.
- Das Package `java.lang` steht dem Programmierer ohne weitere Deklaration zur Verfügung. Zu diesem gehören die Klassen `String`, `Math` und `System`.

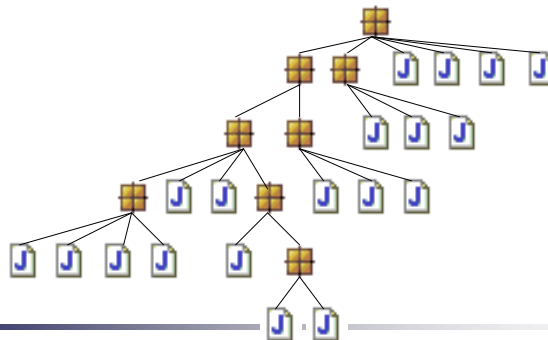
Class Hierarchy

- class java.lang.[Object](#)
- class java.lang.[Boolean](#) (implements java.io.[Serializable](#))
- class java.lang.[Character](#) (implements java.lang.[Comparable](#), java.io.[Serializable](#))
- class java.lang.[Character.Subset](#)
 - class java.lang.[Character.UnicodeBlock](#)
- class java.lang.[Class](#) (implements java.io.[Serializable](#))
- class java.lang.[ClassLoader](#)
- class java.lang.[Compiler](#)
- class java.lang.[Math](#)
- class java.lang.[Number](#) (implements java.io.[Serializable](#))
 - class java.lang.[Byte](#) (implements java.lang.[Comparable](#))
 - class java.lang.[Double](#) (implements java.lang.[Comparable](#))
 - class java.lang.[Float](#) (implements java.lang.[Comparable](#))
 - class java.lang.[Integer](#) (implements java.lang.[Comparable](#))
 - class java.lang.[Long](#) (implements java.lang.[Comparable](#))
 - class java.lang.[Short](#) (implements java.lang.[Comparable](#))
- class java.lang.[Package](#)
- class java.security.[Permission](#) (implements java.security.[Guard](#), java.io.[Serializable](#))
 - class java.security.[BasicPermission](#) (implements java.io.[Serializable](#))
 - class java.lang.[RuntimePermission](#)
- class java.lang.[Process](#)
- class java.lang.[Runtime](#)
- class java.lang.[SecurityManager](#)
- class java.lang.[StackTraceElement](#) (implements java.io.[Serializable](#))
- class java.lang.[StrictMath](#)
- class java.lang.[String](#) (implements java.lang.[CharSequence](#), java.lang.[Comparable](#), java.io.[Serializable](#))
- class java.lang.[StringBuffer](#) (implements java.lang.[CharSequence](#), java.io.[Serializable](#))
- class java.lang.[System](#)
- class java.lang.[Thread](#) (implements java.lang.[Runnable](#))
- class java.lang.[ThreadGroup](#)
- class java.lang.[ThreadLocal](#)
 - class java.lang.[InheritableThreadLocal](#)
- class java.lang.[Throwable](#) (implements java.io.[Serializable](#))
 - class java.lang.[Error](#)
 - class java.lang.[AssertionError](#)
 - class java.lang.[LinkageError](#)
 - class java.lang.[ClassCircularityError](#)
 - class java.lang.[ClassFormatError](#)
 - class java.lang.[UnsupportedClassVersionError](#)
 - class java.lang.[ExceptionInInitializerError](#)
 - class java.lang.[IncompatibleClassChangeError](#)
 - class java.lang.[AbstractMethodError](#)
 - class java.lang.[IllegalAccessError](#)

31

Organisation in Packages

- Der kanonische Name einer Klasse ist der package-Name, gefolgt von einem Punkt und dem Klassennamen. (Der package-Name kann weitere Punkte enthalten.)



Verwendung von Klassen

- Beispiel: Die Klasse String heißt mit vollem kanonischem Namen `java.lang.String`
- Die Deklarationen

```
String hallo = "Hallo";  
java.lang.String hallo = "Hallo";
```

sind äquivalent.

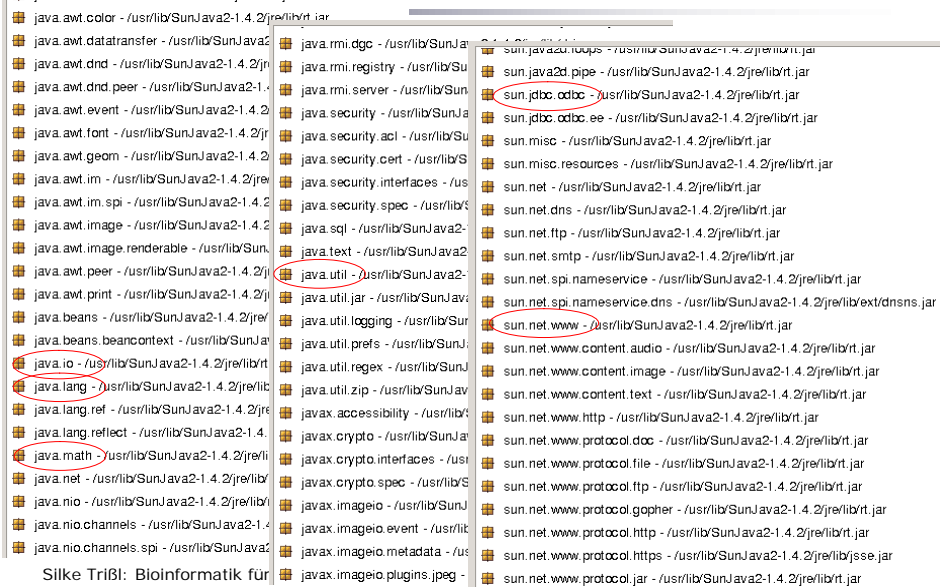
- Im ersten Fall sucht das Java-System in verschiedenen Packages nach der Klasse

Vorhandene Klassen und Packages

- Java hat eine Vielzahl von vorhandenen Klassen, organisiert in Packages

- | | |
|------------------------------|----------------------------------|
| - <code>java.lang.*</code> | String, Math, System, ... |
| - <code>java.Math.*</code> | BigInteger, BigDecimal, ... |
| - <code>java.IO.*</code> | File, DataInput, DataOutput, ... |
| - <code>java.util.*</code> | Random, GregorianCalendar, ... |
| - <code>javax.swing.*</code> | GUI, Box, JoptionPane, JFrame... |
| - ... | |

Vorhandene Klassen und Packages - cont



Importieren von Klassen

- Klassen importieren

```
import java.Package.class;
```

- *class* kann auch durch '*' ersetzt werden
- ? alle Klassen diese Packages werden importiert
- Vereinfacht die Programmierung mit Klassen
 - werden unter ihrem einfachen Namen verfügbar
 - müssen nicht mit dem kanonischen Namen bezeichnet werden

Beispiel

```
import java.util.*;

// Objektinitialisierung
new GregorianCalendar();

// Objektvariable (Zeiger)
GregorianCalendar c = new GregorianCalendar();

// Datum in Date-Format mit getTime()-Methode
Date birthday = c.getTime();

// Datum zu String mit toString()
String S = birthday.toString();
```