**Objektorientierte Programmierung**

**Bisher: Strukturierte Programmierung**

* Beschreibt die Programmsteuerung
* Besteht aus 3 Grundbausteinen
  + Anweisungen
  + Verzweigungen
  + Schleifen

**Jetzt: Objektorientierte Programmierung**

Entstand aus dem Verlangen, die „echte“ Welt besser in Softwareentwicklung abbilden zu können.

Eigenschaften (Daten) und Operationen (Methoden) werden zusammengefasst.

**Vorteile:**

* Bessere Strukturierung des Codes
  + z.B. durch Einteilung von Teilaufgaben in Klassen
* Kapselung von Daten
  + z.B. was darf von außen geändert werden
* Bessere Wiederverwertbarkeit
  + z.B. können Klassen, die Teilaufgaben behandeln, wiederverwendet werden

**Konzepte der OOP**

* Datenkapselung
* Vererbung
* Polymorphie

**Datenkapselung**

* Encapsulation, information hiding
* Auf die Eigenschaften eines Objekts kann nur über Methoden zugegriffen werden, der direkte Zugriff ist nicht möglich.

**Vererbung**

* Inheritance
* Klassen können von anderen Klassen abgeleitet werden und erben damit alle Eigenschaften und Methoden der Oberklasse (Superklasse).

**Polymorphie**

* „Mehrgeestalt“, „Vielgestalt“
* Objekte verschiedener Klassen reagieren auf den gleichen Methodenaufruf unterschiedlich.

**Klasse**

* Bauplan für gleichartige Objekte
* Beschreibt Strukturen und Verhalten
* Besteht aus:
  + Eigenschaften (Daten, Attribute) – in Variablen abgelegt.
  + Methoden (Operationen)

**Grafische Darstellung**

**Bestandteile einer Klasse**

|  |
| --- |
| **Konto** |
| nummer: int  stand: double |
| einzahlen (betrag: double)  abheben (betrag: double)  ueberweisen (zielkonto: Konto, betrag: double) |

Klasenname Eigenschaften (Variablen)

Operationen (Methoden)

class Bankaccount {

int number; //Kontonummer

double balance; / /Kontostand

void deposit (double amount) { //einzahlen

balance = balance + amount;

}

// …

}

|  |
| --- |
| **Fahrzeug** |
| hersteller: String  typ: String  anzahlSitze: int  geschwindigkeit: double |
| beschleunigen ()  bremsen () |

Klasenname Eigenschaften (Variablen)

Operationen (Methoden)

|  |
| --- |
| **Kraftfahrzeug** |
| hersteller: String  typ: String  anzahlSitze: int  geschwindigkeit: double  treibstoffMenge: double |
| beschleunigen ()  bremsen ()  betanken (menge: double) |

Klasenname

Eigenschaften (Variablen)

Operationen (Methoden)

**Klasse <> Objekt**

* Klasse ~ „Bauplan“, „Schablone“ zur „Konstruktion von Objekten
* Objekt = Instanz einer Klasse, welches zur Laufzeit erzeugt wird
* Klasse ist etwas Abstraktes
* Objekt ist etwas Konkretes

**Variablen**

* Die Daten jedes Objekts sind in den Variablen oder Klassen abgelegt
* Variablen können in jedem Objekt (=jeder Instanz) anders belegt sein und werden daher als *Instanzvariablen* bezeichnet.

**Methoden**

* Beschreiben das Verhalten der Klasse
* Können den Zustand des Objektes verändern oder Informationen über den Zustand des Objektes zurückliefern.
* Konvention: Methoden sind Verben und beginnen mit einem Kleinbuchstaben (z.B. getBalance(), deposit(), setColor(),…)

**Objekt erzeugen  
Instantisieren**

class BandAccount { … }  
//…  
BankAccount myAccount = new BankAccount();  
//…

**Konstruktor**

* Spezielle „Methode“ zur Initialisierung eines neu erzeugten Objekts (Anfangszustand herstellen).
* Name des Konstruktors = Name der Klasse
* beginnt mit einem Großbuchstaben
* Liefert nie ein Ergebnis zurück, es wird auch kein Ergebnistyp angegeben (auch nicht void)
* Parameterliste möglich
* Solange kein Konstruktor implementier (programmiert) ist, verwendet Java einen Default Konstruktor

**Konstruktor**

class BankAccount {  
 private int number; // Instanzvariable  
  
public BankAccount(int accountNumber) { // Konstruktor  
 number = accountNuber;  
 }  
}  
//…  
BankAccount myAccount = new BankAccount(1000);  
//…

**Zugriffrechte**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Zugriffmodifier | In gleicher Klasse | In gleichem Package | In Unterklassse (Vererbung) | Global |
| privat | J | N | N | N |
| Default | J | J | N | N |
| protected | J | J | J | N |
| public | J | J | J | J |

**Kommunikation zwischen Objekten**

* Attribute und Methoden müssen adressierbar sein
* Punktnotation zum Trennen von Objekten und Attribut/Methode

**this**

**Problemstellung**

|  |
| --- |
| class BankAccount {  private int number;    public BankAccount(int number) {  number = number;  }  } |

|  |
| --- |
| class BankAccount {  private int number;    public BankAccount(int number) {  this.number = number;  }  } |

**this**

* this bedeutet in etwa „*diese Instanz*“
* Die this-referenz löst das Problem, wenn Parameter oder lokale variablen objektvariablen (Instanzvariablen) verdecken.
* Häufig in Konstruktoren und in settern („setter“-Methoden) zu finden

|  |
| --- |
| public setOwner (String owner) {  this.owner = owner;  } |

**static**

* der Modifier *static* dient dazu, Methoden und variablen der Klasse als Ganzes zuzuordnen, anstatt den einzelnen Instanzen.
* *static*-Komponenten sind nur einmal pro Klasse vorhanden.
* static-Methoden können aufgerufen werden, ohne dass (mit **new**) eine Instanz einer Klasse erzeugt wird.
* *static*-Methoden einer Klasse haben nur Zugriff auf *static*-Komponenten einer Klasse.
* Zugriff über **Klassenname.Komponentenname**, z.B. **Bankaccount.nextAccountNumber**, oder **BankAccount.getCountryCode()**, weil statische Komponenten der Klasse (und nicht der Instanz) gehören.
* Man spricht auch von *Klassenvariablen,* bzw. *Klassenmethoden*.

**Unified Modeling Language**

**UML Klassendiagramm**

|  |
| --- |
| Klasse |
| - attribut0  - attribut1  + attribut2 |
| + operation0()  # operation1()  - operation2()  ~ operation3() |

Klassenname

Attribute (Felder) statisches Attribut

Methoden

^

Zugriffsmodifier

**Sichtbarkeit**

**Zugriffsmodifier**

+ public

- private

# protected

~ package

**Klassendiagramm**

**BankAccount**

|  |
| --- |
| BankAccount |
| - number: int  - balance: double |
| +BankAccount(int)  + getBalance(): double  + deposit(double): void  + withdraw(double): void  +transfer(BankAccount, double): void |

**Überladen von Methoden/Konstruktoren**

**engl. “overloading”**

* Mindesten 2 Methoden (od. Konstruktoren) einer Klasse haben den selben Methodennamen
* -> müssen unterschiedliche Parameterliste haben, damit sie von Java untersieden werden können
* Parameterliste ist unterschiedlich, wenn
  + Anzahl der Parametern verschieben
  + Datentypen der Parameter verschieben
  + Datentypen der Parameter verschieden

Nur Datentypen der Parameter sind relevant, nicht ihre Namen

Rückgabetyp der Methode ist hier irrelevant

**Erzeugen eines Objekts aus einem String**

* objekt.toString() -> String
* Klasse.valueOf(String) -> Objekt
* Erzeugung eines Objekts: mit new
  + -> new-Aufruf ist in der Methode valueOf() gekapselt
* Aufruf von valueOf() auf der Klasse
  + -> ist eine statische Methode (static Keyword)

**String-Methoden**

**Bereits bekannte Methoden**

Methoden, die auf Instanzen der Klasse String aufgerufen werden können.

* charAt()
* length()
* equals()
* toLowerCase()
* toUpperCase()

**In valueOf() Implementierung verwendet**

* indexOf()
* lastIndexOf()
* substring()

**indexOf()**

Liefert den Index des ersten Auftretens des angegeben Zeichens oder Strings ab einer bestimmten Position.

* Int indexOf(int ch) ch: Character, z.B. ‘a’
* Int indexOf(int ch, int fromIndex) fromIndex: Position, ab der gesucht werden soll
* Int indexOf(String substring)
* Int indexOf(String substring, int fromIndex)

Referenztyp <> elementarer Typ

Elementarer Datentyp

Ist direkt in Java definiert

Datentypen beginnen mit Kleinbuchstaben

Die Größe von elementaren Datentypen ist Typabhängig

Auf elementaren Datentypen kann keine Methode aufgerufen werden

Haben immer einen Wert (ist nie ‘null‘)

Bsp. Int, boolean, char, …

Referenztyp

Ist nicht direkt in Java definiert

Datentypen beginnen mit Großbuchstaben

Alle referenztypen haben die gleiche (Speicher) Größe

Kann zum Aufruf von Methoden verwendet werden

Kann ‘null‘ sein

Bsp. String, Arrays, Klassen, …