Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

Objektorientierte Softwareentwicklung

Part 2 - Klassen, Objekte und Methoden

Die Inhalte der Vorlesung wurden primär auf Basis der angegebenen Literatur erstellt. Darüber hinaus sind ausgewählte Aspekte direkt aus der Vorlesung von Prof. Dr.-Ing. Faustmann (ebenfalls HWR Berlin) übernommen worden. Für die Bereitstellung dieses Vorlesungsmaterials möchte ich mich an dieser Stelle noch einmal recht herzlich bedanken.

Abstraktion I

- Wichtiges OO-Prinzip - Trennung zwischen Konzept und Umsetzung
- Vergleich mit anderen Bereichen:
  - Bauplan ←→ Bauteil
  - Rezept ←→ Speise
  - Technisches Handbuch ←→ technisches Geräte
- Der Umgang mit Objekten der realen Welt, von denen das Konzept bekannt ist, kann ohne Kenntnis der internen Details erfolgen
  - Führerschein ermöglicht das Autofahren
  - Rezept für Sachertorte ermöglicht das Backen

→ OOP Unterscheidung zwischen Klasse und Objekt

Abstraktion II

- Klasse als Konzept eines oder mehrerer ähnlicher Objekte
- Objekt als tatsächlich existierendes „Ding“ innerhalb eines Programms
- Eine Klasse beschreibt folgende Aspekte:
  - Welche Eigenschaften hat das Objekt?
  - Wie verhält sich das Objekt?
  - Wie ist das Objekt zu bedienen?
  - Wie wird das Objekt hergestellt?
  - Wie kann das Objekt wieder beseitigt werden?
- Unterscheidung zwischen Klassen und Objekten (Abstraktion)
  - Ignorieren von Details
  - Reduktion von Komplexität

Konzept von Klassen I

- Aufgaben der Analyse als erste Phase der SW-Entwicklung:
  - Objekte identifizieren und beschreiben
  - Beziehungen zwischen Objekten erkennen
    - Vererbungsbeziehungen (Generalisierung & Spezialisierung)
    - Aggregations- und Kompositionsbeziehungen
    - Verwendungs- und Aufrufbeziehungen
  - Etablierung entsprechender Klassifikationen (Abstraktion)

  Ergebnis: → Klassen in einem Klassenmodell
- Eine Klasse legt die Eigenschaften & Fähigkeiten von Objekten fest.
- Ein Objekt ist dann eine Variable vom Typ einer bestimmten Klasse

Merke: Die Begriffe Objekt und Instanz werden in der OOSE zumeist synonym verwendet

Konzept von Klassen II

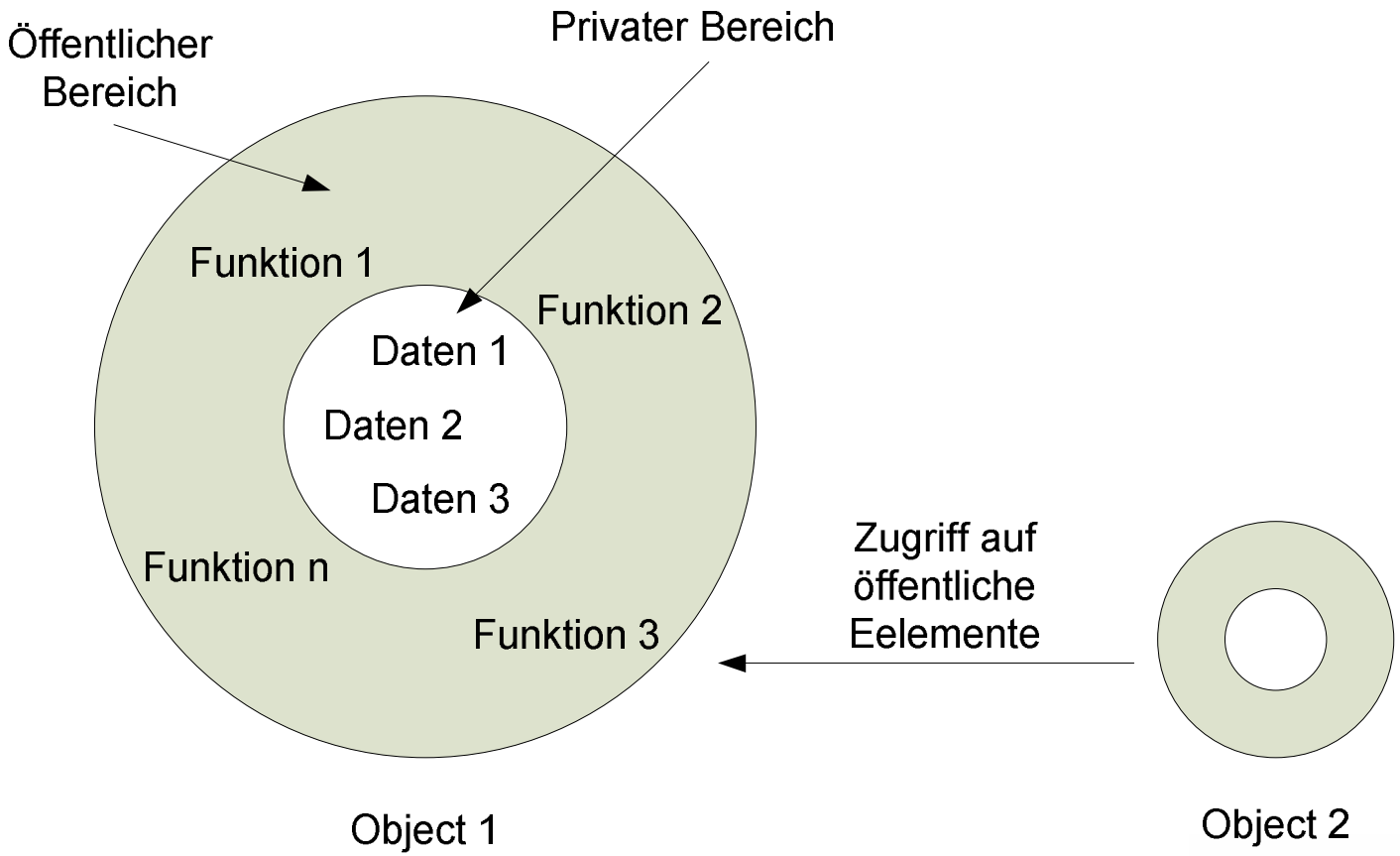
Eine Klasse enthält:

- Private Elemente, sind vor dem Zugriff von außen geschützt.
- Öffentliche Elemente, sind von außen zugreifbar.
- Solche Elemente können sowohl Daten als auch Funktionen sein.

Kapselung

- Zusammenfassung von Daten und Funktionen (Methoden)
- Daten als Satz von Variablen (bei jeder Instanz neu):
  - Attribute
  - Membervariablen (analog zu C++)
  - Instanzvariabelen
- Methoden sind im ausführbaren Programmcode nur 1x vorhanden

Konzept von Klassen III



Daten als privat deklarieren und öffentliche Funktionen zur Verfügung stellen, mit denen auf die Daten zugegriffen werden kann.

Definition von Klassen - Java

```
public class Konto{
    // Geschützte Elemente:
    private String name;              // Kontoinhaber
    private long nr;                  // Kontonummer
    private double stand;             // Kontostand

    // Öffentliche Schnittstelle:
    public boolean init(String name, long nr, double stand){…}
    public void display(){…}
}
```

- Die Bezeichner private und public dürfen beliebig oft in einer Klassendefinition erscheinen.
- Private Elemente sind nur in der Klasse selbst sichtbar.
- Elemente verschiedener Klassen können den gleichen Namen haben.

Definition von Methoden - Java

- Die Definition einer Methode erfolgt innerhalb der Klassendefinition
- Methoden als Pendant zu den Funktionen anderer Prg.-Sprachen
  - Innerhalb der Methode können alle Elemente (Attribute & Methoden) der Klasse *direkt mit ihrem Namen* angesprochen werden.
  - Bei der Definition einer Klasse wird noch *kein Speicherplatz* für die Elemente angelegt. Dies erfolgt erst bei Bildung einer Instanz.

Syntax (ähnlich wie bei C/C++):

```
{Modifier}
Typ Name ([parameterliste])
{
   {Anweisung;}
}
```

Definition von Methoden - Java

```
public class Konto{
  private String name;                  // Kontoinhaber
  private long nr;                      // Kontonummer
  private double stand;                 // Kontostand

// Die Methode init() kopiert die übergebenen Argumente
// in die privaten Elemente der Klasse.
public bool init(String name, long nr, double stand){
   if(name.length() < 1){      // Kein leerer Name
      return false;
   }
   this.name  = name;
   this.nr    = nr;
   this.stand = stand;

   return true;
}
```

Variable Parameterliste (ab J2SE 5.0)

```
// Methode mit variabler Parameterliste
public setDate(int ... date) {
        this.tag = date[0];
        this.monat = date[1];
        this.jahr = date[2];
}
```

Verwendung der Methode im Kontext des Objekts myDatum2

```
System.out.println("Variable Parameterliste");
Datum myDatum2 = new Datum();
myDatum2.setDate(new int[] {15,04,2015});
myDatum2.print();
```

Definition von Methoden - Java

- `this` – Zeiger der beim Anlegen eines Objektes automatisch generiert wird
  - Referenzvariable die auf das aktuelle Objekt zeigt
  - Ansprechen der eigenen Methoden und Instanzvariablen
- `this` – Zeiger kann explizit verwendet werden
  - Ggf. nicht unbedingt erforderlich, aber:
  - Hervorheben eines Zugriffs auf eine Instanzvariable
- Der `this` – Zeiger wird als versteckter Parameter an jede nicht statische Methode übergeben.

Verwendung von Objekten I

Bilden eines Objektes (Instanz) von einer Klasse:

- Deklarieren einer Variablen (Referenzvariable) vom Typ der Klasse
- Zuweisung eines neu erzeugten Objektes an die Referenzvariable
  - Objekterzeugung mit Hilfe des new-Operators
  - Objektinitialisierung durch Aufruf des Konstruktors

Syntax:

```
klassenname objektname = new klassenname();
```

Beispiel:

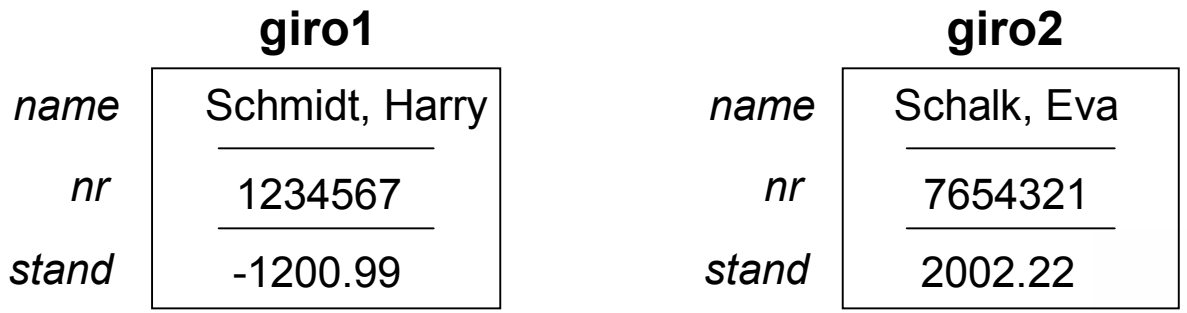
```
Konto giro1 = new Konto();
Konto giro2 = new Konto();
```

Verwendung von Objekten II

- Der Zugriff auf die Elemente (Variablen und Methoden) eines Objekts erfolgt mit Hilfe des Punktnotation (auch Punktoperator):

Beispiel:

```
giro1.init("Schmidt, Harry", 1234567, -1200.99);
giro2.init("Schalk, Eva", 7654321, 2002.22);
```



Verwendung von Objekten III

Die Verwendung des init-Operators im Beispiel kann nicht durch direkte Zuweisung ersetzt werden:

```
giro1.name = "Lustig, Peter";          // Fehler, da
giro1.nr = 1234567;                    // Datenelemente
giro1.stand = -1200.99                 // privat deklariert!!!
```

Genauso wenig können die einzelnen Datenelemente z.B. angezeigt werden:

```
System.out.print(giro1.stand);         // Fehler!
giro1.display();                       // Ok
```

`display()` muss an ein Objekt gebunden werden. Ohne Objektbindung würde ein Fehler ausgegeben.

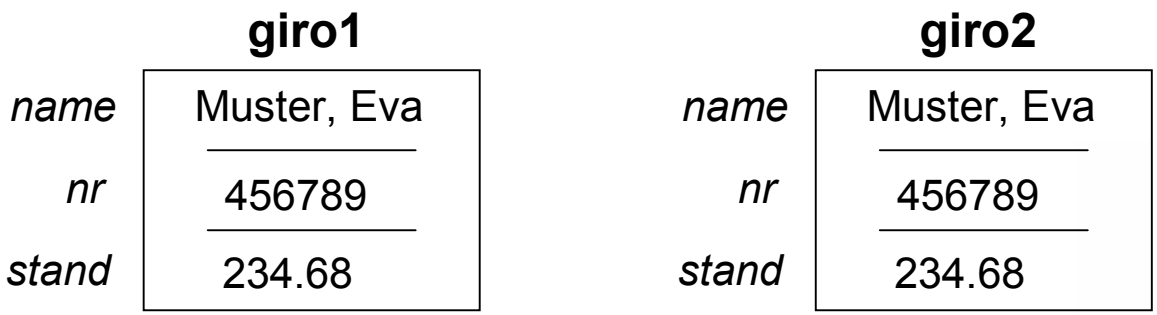
Verwendung von Objekten IV

- Die Zuweisung einer Referenz kopiert lediglich den Verweis auf das betreffende Objekt, das Objekt selbst dagegen bleibt unkopiert.
- Nach einer Zuweisung zweier Referenztypen zeigen diese also auf ein und dasselbe Objekt.
- Sollen Objekte kopiert werden, so muß dies explizit durch Aufruf der Methode `clone()` der Klasse `Object` erfolgen.
  - Z.B. beim Schutz vor Seiteneffekten notwendig

Verwendung von Objekten V

Bsp:

```
Konto giro1 = new Konto();
giro1.init("Muster, Eva", 456789, 234.68);
Konto giro2;
try{
                    giro2=(Konto)giro1.clone();
}catch(CloneNotSupportedException e){…}
```



Übung 2-1

- Erstellen Sie eine Klasse `Datum` mit drei ganzzahligen Datenelementen für Tag, Monat und Jahr. Implementieren Sie außerdem folgende Methoden:
  - `public void init(int tag, int monat, int jahr)` kopiert die übergebenen Werte in die entsprechenden Datenelemente.
  - `public void print()` gibt das Datum im Format `Tag.Monat.Jahr` auf der Standardausgabe aus.
  - `public void setDate(int … date)` setzen des Datums mit variabler Parameterliste.

  Testen Sie die Klasse mit einem Anwenderprogramm.

Muss

- Erstellen Sie zusätzlich die Methode `public void init()`, die das **aktuelle** Datum in einem neuen Objekt speichert.

  Weiterführende Hinweise: siehe folgende Seite

Kann

Übung - Hinweise

- Verwenden Sie die Datumsobjekte, die aus der Klasse `GregorianCalendar` (`import java.util.*;`) erzeugt werden:

```
GregorianCalendar cal = new GregorianCalendar();
```

- Aus diesen Objekten können dann die gewünschten Informationen abgefragt werden:

```
cal.get(Calendar.DATE);
cal.get(Calendar.MONTH); // Von 0…11!
cal.get(Calendar.YEAR);
```

- Auch Zeitangaben können abgefragt werden:

```
Calendar.HOUR    Calendar.MINUTE     Calendar.SECOND
```

Konstruktoren I

- Spezielle Methode in jeder OO-Programmiersprache die bei der Initialisierung eines Objektes aufgerufen wird.
- Mit Hilfe eines Konstruktors können während der Bildung einer Instanz Datenelemente initialisiert werden.
- Konstruktoren in Java:
  - Erhalten den Namen der Klasse zu der sie gehören
  - Definiert als Methoden ohne Rückgabewert
  - Der Konstruktor ist als public-Methode zu definieren
  - Dürfen beliebige Anzahl an Parametern haben
  - Es können mehrere Konstruktoren vorhanden sein

Konstruktoren II

- Konstruktoren können überladen (mehrere Konstruktoren) und mit unterschiedlichen Parameterlisten ausgestattet werden.
- Im Rahmen der new-Anweisung wählt der Compiler anhand der aktuellen Parameterliste den passenden Konstruktor aus
- Existiert kein expliziter Konstruktor wird vom Compiler automatisch ein parameterloser „default“-Konstruktor erzeugt.

Beispiel:

```
public Konto( String_name, long nr, double stand){
   tis.name   = name;
   this.nr    = nr;
   this.stand = stand;
}
```

Konstruktoren III

- Wenn ein Objekt definiert wird, können hinter dem Namen des Objekts Anfangswerte in Klammern angegeben werden.

  Syntax:

```
klasse objekt = new klasse(initialisierungsliste);
```

  Beispiel:

```
Konto giro1 = new Konto("Muster, Eva", 456789, 34.0);
```

Konstruktoren IV

- Es muss ein passender Konstruktor vorhanden sein. Werden eigene Konstruktoren definiert, so gibt es keinen parameterlosen Standardkonstruktor mehr!

```
// Erster Konstruktor
public Konto(String name){…}
// Zweiter Konstruktor
public Konto(String name, long nr){…}

Konto giro4 = new Konto("Glueck, Maria"); // möglich
Konto giro5 = new Konto("Meiser, Hans",123456); // möglich
Konto giro6 = new Konto();     // nicht möglich!!!
```

Destruktoren I

- Wenn ein Objekt zerstört wird, müssen ggfs. Aufräumarbeiten durchgeführt werden (z.B. Freigeben von Ressourcen, Schließen von Dateien).
- Diese Aufgabe übernimmt der Destruktor eines Objekts. Der Destruktor einer Klasse heißt `finalize()`:

  Syntax: `protected void finalize(){…}`
- Der „geschützte“ Destruktor besitzt weder eine Parameterliste noch einen Rückgabewert.

ACHTUNG!

Der Aufruf eines Destruktors erfolgt automatisch. Es wird allerdings nicht garantiert, dass er aufgerufen wird, da erst der „Garbage Collector“ den Speicherplatz freigibt. Dieser Zeitpunkt ist unbekannt!

Destruktoren II

Beispiel eine Destruktors:

```
protected void finalize(){
   System.out.println("Das Konto mit der Nummer" +
    nr +
    "wurde gelöscht.");
}
```

Erstellen Sie, falls Sie mit Sicherheit Aufräumarbeiten durchführen müssen, eine eigene `dispose`-Methode, die Sie selbst vor Zerstörung eines Objekts der Klasse aufrufen! Diese gibt zugeordnete Ressourcen bzw. ggf. vorhandene Registrierungen explizit frei.

Destruktoren III

- Destruktoren haben in Java geringere Bedeutung als in anderen objektorientierten Programmiersprachen
- Bei Verwendung von Java muss sich der Entwickler nicht explizit um die Rückgabe von belegten Speicher kümmern → anders als bei C++
- Sprachspezifikation von `finalize` garantiert nicht den Aufruf des Destruktors, daher kann die Objektreferenz ggf. noch immer vorhanden sein
- Die Verwendung von Destruktoren sollte in Java mit der nötigen Vorsicht durchgeführt werden!

Zugriffsmethoden

- Der Zugriff auf private Datenelemente sollte durch eigene Methoden erfolgen (sog. getter- und setter-Methoden):

```
String getName()          // liefert Namen
long getNr()              // liefert Nummer
double getStand()         // liefert Stand
```

bzw.

```
void setName(String s)  // setzt Namen
void setNr(long n)      // setzt Nummer
void setStand(double x) // setzt Kontostand
```

Übung 2-2

Erstellen Sie eine Java-Klasse `Artikel` mit den folgenden Elementen:

- privat:
  - Artikelnummer long
  - Artikelbezeichnung string
  - Verkaufspreis double
- öffentlich:
  - Artikel(long, String, double); // Konstruktor
  - finalize(); // Destruktor
  - void print(); // Formatierte Ausgabe
  - set- und get-Methoden für jedes Datenelement

Der Destruktor soll Nummer und Bezeichnung des zerstörten Artikels ausgeben. Testen Sie die Klasse so, dass auch der Destruktor aufgerufen wird. Wo müssen Artikelobjekte dann angelegt werden?

Statische Datenelemente

- Eigenschaften die nicht an Instanzen einer Klasse gebunden sind
  - Klassen- oder statische Variablen
  - Klassenvariablen existieren unabhängig von einem Objekt
- Die enthaltenen Daten sind für alle Instanzen gleich (vgl. globale Variable)
  - Kennzahlen, wie Umrechnungskurse, Zinssätze, Zeitlimits
  - Statusinformationen, wie die Anzahl existierender Objekte
- Diese Daten werden innerhalb der Klasse verwaltet.
- Eine Klassenvariable wird als statisches Datenelement innerhalb der Klassendefinition definiert.

  Bsp.: `static private int anzahl = 0;`

  Hier wird eine für alle Objekte der Klasse gültige Variable angelegt und initialisiert. Änderungen an dieser Variablen sind für alle Objekte sichtbar.

Der Zugriff erfolgt über `Klassenname.Variablenname`

Statische Datenelemente

- Der Zugriff auf statische Datenelemente sollte über die Klasse erfolgen:

```
count = Konto.anzahl;
```

- Ist das statische Datenelement jedoch `private` deklariert, muss der Zugriff über eine Methode erfolgen. Diese Methode kann als statische Elementfunktion realisiert werden, damit der Zugriff über die Klasse erfolgen kann. Das ist auch dann möglich, falls noch kein Objekt existiert:

```
static public void getAnzahl();     // innerhalb der
                                     // Klassdendefinition

Konto.getAnzahl();                   // innerhalb eines
                                     // beliebigen Bereichs
```

Übung 2-3

- Erweitern Sie die Klasse `Artikel` um ein statisches Datenelement `anzahl`, das die Anzahl der Artikel verwaltet. Dieses Datum soll als `private` verwaltet werden. Dementsprechend werden Methoden benötigt, um auf dieses Datum zuzugreifen.
- Das Auslesen soll als statische Elementfunktion realisiert werden, wogegen die Änderung des Wertes nur durch Methoden der Objekte (Auf- und Abbau von Artikel-Objekten) vollzogen werden darf.
- Verändern Sie dazu den Konstruktor und den Destruktor der Artikelklasse geeignet.