gds2 Gottlieb-Daimler-Schule 2
Technisches Schulzentrum Sindelfingen
mit Abteilung Akademie für Datenverarbeitung

Inhalt

1 Begriffsdefinitionen

Zur Erinnerung:
*"Eine Klasse ist eine Gruppe von Dingen, Lebewesen oder Begriffen mit gemeinsamen Merkmalen. D.h. eine Menge von Objekten mit gleichen **Attributen** und gleichen **Operationen** (Methoden) und gleichen **Beziehungen** (Assoziationen, Vererbungsbeziehungen)*
*Eine Klassenbeschreibung dient als **Schablone**, die angibt, wie ein Objekt einer Klasse aussehen soll.*
Eine Klasse besitzt einen Mechanismus um Objekte zu erzeugen (new).
Der Klassenname ist ein Substantiv im Singular."

Je nach Verwendungszweck der Klasse unterscheidet man unterschiedliche **Klassentypen**:

Typ	**Beschreibung**	**Beispiel**
Start-Klasse	• startet ein Programm **enthält main-Methode** • bei strukturierter/prozeduraler Programmierung auch die Programmlogik • Klassenname beginnt bei komplexeren Programmsystemen mit dem Präfix "Start" • Ein Programm hat immer nur eine Startklasse.	Uebung1, StartKunde
Standardklassen	sie stellen häufig benötigte Programmlogik zur Verfügung	StringBuilder, StringTokenizer, usw.
Utility-Klassen	enthalten nur Klassenmethoden	Eingabe, Kalender, Math
Fachkonzeptklassen	• sie modellieren die fachliche Logik einer Anwendung. • sie beschreiben eine Menge von Objekten mit gleichen Attributen, Operationen und Beziehungen	Kunde, Auto, Bruch, …
UI-Klasse (UI = User Interface)	• nimmt Bildschirmausgaben oder/und Tastatureingaben vor. Kann auch gleichzeitig StartKlasse sein, also main enthalten • Klassenname endet mit dem Suffix "UI". (Bei grafischen Benutzerober-flächen "GUI"; genaueres später)	KundeUI, AutoUI, BruchrechnenUI
Container-Klasse	• übernimmt die Datenhaltung, also die Speicherung der erfassten Daten auf geeignetem Medium • Klassenname endet mit dem Suffix "Container". (Genaueres später)	KundeContainer, AutoContainer

2 Eigene Fachkonzeptklassen erstellen

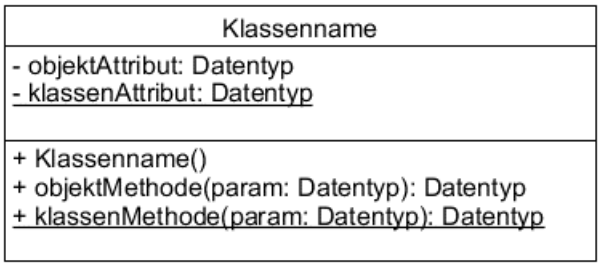
Eigene Fachkonzeptklassen (im folgenden der Einfachheit halber "Klassen") zu erstellen ist dann notwendig, wenn man nicht auf bereits vorhandene Klassen wie z.B. die Standardklassen des Jdk zurückgreifen kann oder will.

Sind eigene Klassen erst einmal erstellt, werden sie genauso benutzt wie andere Klassen auch. Im Prinzip wird also mit einer eigenen Klasse ein *neuer, individueller Datentyp* erstellt.

2.1 Beschreibung von Klassen mit Hilfe von UML

Als Beschreibungssprache für die Entwicklung von objektorientierten Anwendungen wird **UML** (**U**nified **M**odeling **L**anguage) verwendet.
UML stellt eine Reihe von verschiedenen Diagrammen zur Verfügung, um ein System umfassend zu beschreiben, u.a. ein ***Klassendiagramm***: darin werden die Klassen, die für die Entwicklung einer Anwendung notwendig sind, genau beschrieben und ihr Zusammenhang untereinander dargestellt.

Die Beschreibung einer Klasse besteht aus 3 Teilen:



Der Klassenname

- ist ein **Substantiv** im Singular
- wird großgeschrieben
- soll innerhalb eines Java Package eindeutig sein.

Die ***Attribute***

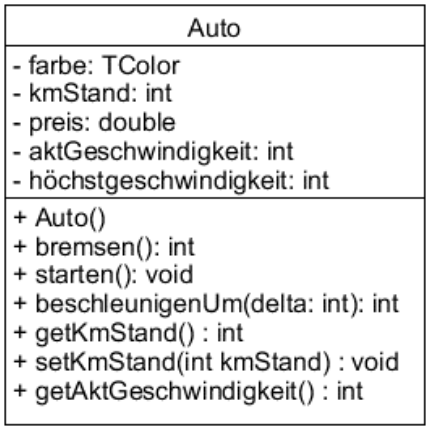
- sind **Substantive** im Singular
- dienen zur Beschreibung von Eigenschaften und Zuständen der Objekte einer Klasse
- werden klein geschrieben
- müssen innerhalb der Klasse eindeutig sein
- dürfen **nur über die Operationen** der zugehörigen Klasse gelesen oder geschrieben werden (***Geheimnisprinzip***). Alle Attribute sollen daher mit dem **Zugriffsmodifizierer**[1] **private** versehen werden. Dies wird in der UML durch ein vorangestelltes Minus-Zeichen (**-**) gekennzeichnet.
  Daher gibt es **für jedes Attribut 2 Zugriffsmethoden**: eine zum Lesen und eine zum Schreiben. Man bezeichnet diese Methoden als **Accessor**- und **Mutator-Methoden**.
  Namenskonvention: **getAttributsname()** bzw. **setAttributsname()**
  (siehe Informationsblatt "Zugriffsmodifiziererer")
- können mit einem Datentyp versehen werden, der dann durch Doppelpunkt vom Attributsnamen getrennt wird: z.B. Name: String
- Klassenattribute werden im Gegensatz zu Objektattributen unterstrichen.

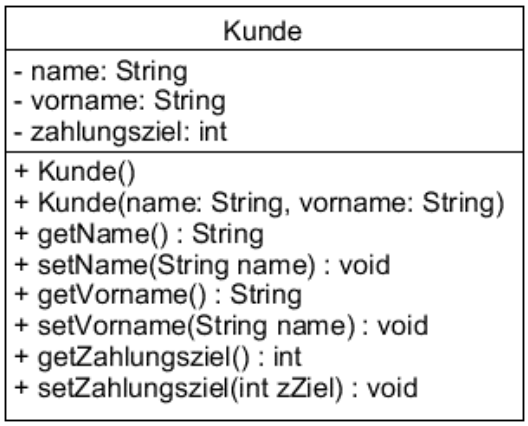
[1] Engl. „access modifier“; deutsch auch Sichtbarkeitsmodifizierer

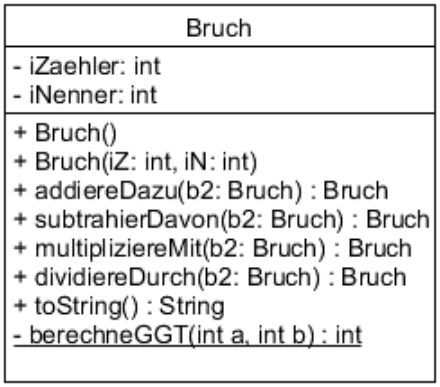
Die ***Methoden (Operationen)***

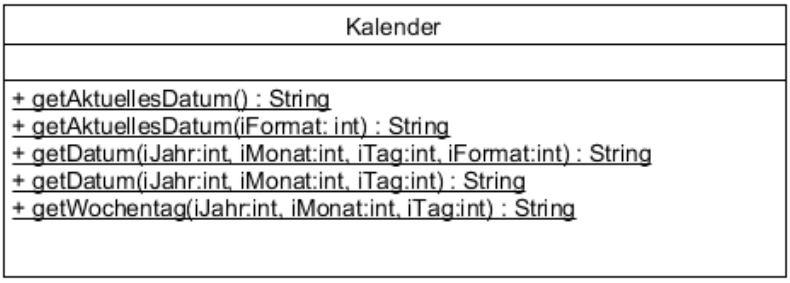
- Methodennamen beginnen im Allgemeinen mit einem **Verb**, dem ein Substantiv folgen kann. Der Methodenname soll möglichst die Art der Operation beschreiben.
- Methodennamen werden klein geschrieben
- dem Methodennamen kann eine **Parameterliste** folgen, die in runde Klammern eingeschlossen ist
- der Parameterliste kann ein **Ergebnistyp** der Operation folgen. Er entspricht dem Datentyp des Rückgabewerts der Operation. Liefert eine Methode keinen Rückgabewert, so erhält sie den Datentyp void.
- Damit Methoden von anderen Klassen aus aufgerufen werden können, müssen sie mit dem **Zugriffsmodifizierer public** versehen werden. Dies wird in der UML durch ein vorangestelltes **+** gekennzeichnet.
- Klassenmethoden werden im Gegensatz zu Objektmethoden unterstrichen.

Beispiele:









2.2 Beispiel: Programmierung der Fachkonzeptklasse Kunde in Java

```
0   public class Kunde
    {
      // Attribute
1     private String sName;
      private String sVorname;
      private int iZahlungsziel;

      // Konstruktor(en)
2a    public Kunde()
      {

      }

      // Accessor-Methoden
2b    public String getName()
      {
3       return this.sName;
      }

      public String getVorname()
      {
        return this.sVorname;
      }

      public int getZahlungsziel()
      {
        return this.iZahlungsziel;
      }

      // Mutator-Methoden
4     public void setName(String sName)
      {
5       this.sName = sName;
      }

      public void setVorname(String sVorname)
      {
        this.sVorname = sVorname;
      }

      public void setZahlungsziel(int iZahlungsziel)
      {
        this.iZahlungsziel = iZahlungsziel;
      }

      // Weitere Methoden
      public String toString()
      {
        String sReturn;

        sReturn = "Name:          " + this.sName + "\n"
                + "Vorname:       " + this.sVorname + "\n"
                + "Zahlungsziel:  " + this.iZahlungsziel + "\n";

        return sReturn;
      } // Ende Methoden
    } // Ende Klasse Kunde
```

0	(Der Quellcode muss in einer gleichnamigen Datei gespeichert werden: `Kunde.java`)
1	Die Attribute der Klasse werden als `private` deklariert. Damit sind sie nur innerhalb der Klasse Kunde bekannt und können von anderen Klassen nicht direkt gelesen oder verändert werden.
2	**a.** Jede Klasse hat (mindestens) einen Konstruktor. Der hier gezeigte ist der Standardkonstruktor. Er müsste eigentlich nicht explizit programmiert werden, da er vom Compiler automatisch erzeugt wird. **b.** Die Methoden einer Klasse, die von anderen Klassen aus aufgerufen werden können, müssen `public` deklariert sein.
3	Die Attribute der Klasse (hier: `sName`) sind innerhalb der Methoden dieser Klasse bekannt (siehe 1). Die implizit vorhandene Verweisvariable `this` verweist immer auf das aktuelle Objekt, auf das die Methode beim Aufrufer angewendet wird (siehe Beispiel in 2.3). Der Attributswert des Attributs `sName` wird mit `return` an den Aufrufer der Methode zurückgegeben.
4	Einer Methode kann ein Aufrufparameter übergeben werden, der innerhalb der runden Klammern deklariert wird. In diesem Fall wird der Methode `setName()` ein String übergeben, der dem Attribut `sName` zugewiesen wird. Die Methode `setName()` hat somit Zugriff auf das Attribut `sName`. Wichtig: Nur Objektmethoden haben über die Verweisvariable `this` Zugriff auf Objektattribute. Klassenmethoden haben nur Zugriff auf Klassenattribute (siehe später) Klassenmethoden kennen die Objektattribut nicht.
5	Der übergebene Parameter `sName` wird dem Attribut `this.sName` zugewiesen. Hat der Parameter den gleichen Namen wie das Attribut, dann muss zur Unterscheidung beim Attribut zwingend die implizit existierende Verweisvariable `this` verwendet werden.

2.3 Anwendung der neuen Klasse

Die neu erstellte Klasse kann jetzt genau so benutzt werden wie die Standardklassen auch. Es muss also ein entsprechendes Programm geschrieben werden, in dem z.B. ein Objekt dieser Klasse erzeugt wird und in dem die Methoden aufgerufen werden, die die neue Klasse anbietet. Häufig ist das eine entsprechende UI-Klasse (oder GUI-Klasse) bzw. eine Startklasse.

Beispiel:

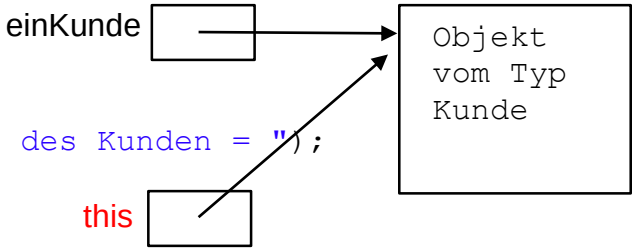
```
public class KundeUI  // Start- und UI-Klasse in einem
{
  public static void main(String[] args)
  {
    Kunde einKunde = new Kunde();
    String sName, sVorname;

    sName = Eingabe.getString("Name des Kunden = ");

    einKunde.setName(sName);

    sVorname = Eingabe.getString("Vorname des Kunden = ");
    einKunde.setVorname(sVorname);
    einKunde.setZahlungsziel(0);

    System.out.println(einKunde.toString());
  }
```



Benutzereingaben und Programmausgabe:

```
Name des Kunden = Müller
Vorname des Kunden = Peter
Name:         Müller
Vorname:      Peter
Zahlungsziel: 0
```

2.4 Konstruktoren einer Klasse

Durch die Kapselung ist auf die privaten Daten einer Klasse kein direkter Zugriff mehr möglich. Teilweise ist jedoch eine Vorbelegung (Initialisierung) von Werten erforderlich. Deshalb gibt es die Möglichkeit dies über spezielle Methoden, die sogenannten Konstruktoren zu tun. Ein Konstruktor wird automatisch im Hintergrund aufgerufen, wenn ein Objekt einer Klasse angelegt wird. Er muss denselben Namen wie die Klasse besitzen und hat keinen Rückgabewert (auch nicht void).

Eine Klasse kann mehrere („überladene“) Konstruktoren besitzen.

Beispiele für die Klasse Kunde:

0	`public class Kunde`
	`{`
	`  // Attribute`
	`  private String sName;`
	`  private String sVorname;`
	`  private int iZahlungsziel;`
	
	`  // expliziter Standard Konstruktor (hat keine Parameter)`

1	`  public Kunde()`
	`  {  // Initialisierung mit Standardwerten`
	`  this.sName = "- leer -";`
	`  this.sVorname = "- leer -";`
	`  this.iZahlungsziel = 0;`
	`  }`

2	`  // überladener Konstruktor (mit Parametern)`
	`  public Kunde(String sName, String sVorname, int iZahlungsziel)`
	`  {  // Initialisierung mit Parameterwerten`
	`  this.sName = sName;`
	`  this.sVorname = sVorname;`
	`  this.iZahlungsziel = iZahlungsziel;`
	`  }`

	`  // weitere Methoden (s.o.)`
	
	`} // Ende Klasse Kunde`

1	Der Standardkonstruktor wird aufgerufen, wenn ein Objekt der Klasse in folgender Weise erzeugt wird: `Kunde einKunde = new Kunde();`
2	Der überladene Konstruktor wird aufgerufen, wenn ein Objekt der Klasse in folgender Weise erzeugt wird: `Kunde einKunde = new Kunde("Müller", "Peter", 0);`

Verkettung von Konstruktoren / Konstruktoraufruf mit this()

Benötigt man in einer Klasse mehrere unterschiedliche Konstruktoren , so ist es sinnvoll die Konstruktoren untereinander zu verketten. Dies wird dadurch ermöglicht, dass ein Konstruktor einen anderen der selben Klasse aufruft. Durch die Verkettung wird die Wartbarkeit der Software erhöht, d.h. mögliche Fehlerquellen verringert.

Am vorherigen Beispiel kann gezeigt werden, wie sich durch Konstruktorverkettung der Quellcode vereinfacht:

```
0   public class Kunde
    {
      // Attribute
      private String sName;
      private String sVorname;
      private int iZahlungsziel;

      // expliziter Standard Konstruktor (hat keine Parameter)
1     public Kunde()
      {  // Initialisierung mit Standardwerten durch Verkettung
         // Der überladene Konstruktor wird mit this() aufgerufen
         this("- leer -", "- leer -", 0);
      }

      // überladener Konstruktor (mit Parametern)
      public Kunde(String sName, String sVorname, int iZahlungsziel)
2     {  // Initialisierung mit Parameterwerten
         this.sName = sName;
         this.sVorname = sVorname;
         this.iZahlungsziel = iZahlungsziel;
      }

       // weitere Methoden (s.o.)

    } // Ende Klasse Kunde
```

2.5 Veröffentlichung der neuen Klasse

Ohne weitere Aktionen ist die neue Klasse von allen Klassen nutzbar, die sich im selben Verzeichnis auf der Platte befinden.
Soll die Klasse jedoch auch anderen Programmierern im Sinne einer Wiederverwendung zugänglich gemacht werden, oder Bestandteil eines größeren Programmsystems werden, müssen noch folgende Aktionen durchgeführt werden:

1. Einbinden der Klasse in ein *package*
2. Erstellen eines *jar-Archives* mit diesem package
3. Speicherort des jar-Archives in die *CLASSPATH*-Umgebungsvariable hinzufügen
4. Erstellen einer API-Dokumentation mit dem Programm *javadoc.exe*

Diese Aktionen werden später genauer besprochen (siehe Merkblatt „Pakete“).