Die Klasse

Java ist eine objektorientierte Programmiersprache, die einen vollständigen privaten Datentyp zur Verfügung stellt. In einer Klassendefinition werden die folgenden Bereiche zusammengefasst:

- die Deklaration der **Datenelemente** (Umsetzung der **Attribute)**
- die Definition der zugelassenen Operationen / **Methoden**

Weiter gibt die Klasse somit an, wie die Datenelemente aussehen sollen und welche Operationen damit erlaubt sind. Darüber hinaus wird festgelegt, ob und wie auf bestimmte Bereiche der Klasse zugegriffen werden kann.

Zugriffsschutz

Die in einer Klasse definierten Methoden erhalten eine Zugriffslizenz auf den privaten Teil der Klasse. Sie - und nur sie - dürfen auf die Elemente des privaten Bereiches zugreifen. Ein Element kann ein Datenelement (Klassenvariable / Attribut) oder auch eine Methode sein.
Den Schutzmechanismus, den ein echter Datentyp bieten muss, führt die Klasse durch Schutzbereiche ein.

- Der erste Zugriffsspezifizierer heißt "private". Hiermit werden Datenelemente / Attribute oder auch Operationen / Methoden deklariert, die niemand von außen benutzen darf.
- Der zweite Zugriffsspezifizierer heißt "public". Die hiermit deklarierten Methoden sind von außen zugänglich. Sie bilden die Schnittstelle zu den privaten Elementen der Klasse.
- Der dritte Zugriffsspezifizierer heißt "protected“. Die Datenelemente / Attribute und Methoden sind in der eigenen und in allen public abgeleiteten Klassen zugreifbar, nicht aber in anderen Klassen oder außerhalb der Klasse.

Die Elemente einer Klasse können Daten- oder Unterprogramme sein. Ein Datenelement bezeichnen wir als Eigenschaft (Attribut) einer Klasse. Die zugehörigen Unterprogramme nennt man nun Methoden oder Operatroren. Die Klassendefinition bildet einen Bauplan, der zur Erstellung (Instanziierung) von Programmelementen sogenannten Objekten genutzt werden kann.
In der englischsprachigen Literatur spricht man auch von Mitgliedern einer Klasse. Dort gibt es Mitgliedsdaten oder Mitgliedsfunktionen (data members / function members). Wir wollen bei der Implementierung (Umsetzung) von Attributen bzw Eigenschaften, lieber die Begriffe Datenelemente bzw. Klassenvariablen. Bei der Implementierung der Methoden wird der gleiche Begriff verwendet.

Klassendefinition

Aufbau einer Klasse Zaehler

Zaehler
- zaehlerstand : int
+ Zaehler( ) + Zaehler(int) + setZaehlerstand(int) : void + getZaehlerstand( ) : int + eingabe( ) : void + ausgabe( ) : void

```
public class Zaehler                    // Beginn der Klassendefinition
{                                       // Privater Bereich: an diese Daten und Methoden kommen
                                        // nur Klassenmitglieder heran.
                                        // Datenelemente / Klassenvariablen ( Implementierung der Attribute )
  private int zaehlerstand;                     // Deklaration von zaehlerstand: zugriffsgeschützt mit Datentyp int

                                        // Public-Bereich: offen für alle...
                                        // Konstruktoren
  public Zaehler()                              // Defaultkonstruktor: Methode zur Festlegung aller
                                                // Anfangswerte der Datenelemente / Klassenvariablen
  {
    zaehlerstand = 0;                               // Startwertfestlegung des Datenelements zaehlerstand
  }

                                        // Verwaltungsmethoden zum Verändern und Ermitteln von
                                        //Datenellementwerten
  public void setZaehlerstand(int z)            // Methode zur Veränderung des Datenelementswertes
                                                //(Attributwert)
  {
    if ( z > 0)
       zaehlerstand = z;                        // Wert von zaehlerstand wird auf übergebenen Wert gesetzt
  }

  public int getZaehlerstand()                  // Methode zur Ermittlung des Datenelementswertes
  {
    return zaehlerstand;                            // zaehlerstand wird an das aufrufende Modul zurückgegeben
  }
                                        // Methodenliste
  public void eingabe()                         // Methode zur Eingabe eines Wertes für zahlerstand
  {
    System.out.print("\nZählerstand: ");
    setZaehlerstand( Tastatur.liesInt() );
  }
  public void ausgabe()                         // Methode zur Ausgabe des Wertes für zahlerstand
  {
    System.out.print("\nZählerstand: " + zaehlerstand);
  }
}                                               // Ende der Klassendefinition
```

Betrachten wir das konkrete Beispiel Klasse **Zaehler**. Als Klassenvariable / Attribut wurde **zaehlerstand** mit dem Datentyp int vereinbart. Durch den Zugriffsspezifizierer „**priviate**“ wird die Klassenvariable / Attribut durch den direkten Zugriff von außen geschützt. Somit kann ein Attributwert z.B. im Hauptprogramm nicht direkt verändert werden. Im public-Teil findet sich der Default-Konstruktor **Zaehler()**, die Verwaltungsmethoden **setZaehlerstand()**, **getZaehlerstand()**, sowie die Methoden **eingabe()** und **ausgabe()**.

Der Default-Konstruktor

Fundamentaler Bestandteil von Klassen in Java sind die Konstruktoren, die automatisch bei Anlegen ("Geburt") eines Objektes vom System aufgerufen werden. Ist eine Klasse **K** definiert worden, so kann mit **K o**; ein Objekt **o** dieser Klasse deklariert und definiert werden. Die Konstruktoren sorgen für den geforderten eindeutigen festgelegten Anfangswert der Klassenvariablen / Attribute.
Die Konstruktoren haben stets denselben Namen wie die Klasse selbst und besitzen keinen Rückgabetyp. Konstruktoren stellen eine Erweiterung der klassischen Initialisierung dar. Während bei einer Variablendeklaration direkt nur Werte zugewiesen werden können, ermöglicht Java bei der Geburt eines Objekts einen Funktionsaufruf. Es findet die entsprechende Wert-Initialisierung der Attribute statt und es können darüber hinaus auch weitere Aufgaben wie z.B. die Anforderung von dynamischer Speicher erfolgen.
Bezogen auf unserem Beispiel heißt der Konstruktor „**Zaehler()**“. Mit dem Schlüsselwort public wird wie bei den folgenden Methoden, der Zugriff von außen erlaubt. Im Methodenrumpf wird hier festgelegt, dass das Datenelement zaehlerstand auf den Startwert 0 festgelegt wird.
Bei der Instantiierung eines Objekts wird dieser Konstruktor aufgerufen und sorgt somit für Initialisierung des Datenelements mit dem Wert 0.

Verwaltungsmethoden

Um an die Datenelementwerte eines Objektes zu gelangen, muss die Klasse je zwei Verwaltungsmethoden für jedes Datenelement zur Verfügung stellen. Die set-Mehtode für ein Datenelement ermöglicht die Zuweisung eines neuen Datenelementwertes nach der Instanziierung. Alle set-Methoden geben keinen Wert zurück. Es wird somit als Rückgabetyp void festgelegt. Für den Übergabeparameter ist immer der entsprechende Datentyp des Datenelements zu wählen. Im Methodenrumpf wird dem entsprechenden Datentelement der Parameter zugewiesen. **Eine entsprechende Überprüfung kann die Zuweisung irregulärer Werte verhindern.**
Mit der get-Methode kann der aktuelle Wert eines Datenelements ermittelt und zurückgegeben werden. Für alle get-Methoden ist als Rückgabetyp der Datentyp des Datenelements anzugeben. Es wird kein Übergabeparameter zugelassen. Im Methodenrumpf wird mit return das Datenelement zurückgegeben.
Als Zugriffsspezifizierer wird im Regelfall für beide Methoden mit public festgelegt. In speziellen Fällen sind aber auch die Zugriffsspezifizierer private und protected möglich. Soll z.B. ein Datenelement nur klassenintern genutzt werden und ein Aufruf der Verwaltungsmethoden nur durch klasseninterne Methoden erlaubt werden, so wäre private als Zugriffsspezifizierer zu wählen.
In unseren Beispiel sollen die Verwaltungsmethoden von außen aufzurufen sein und somit wurde für beide Methoden der Zugriffsspezifizierer public gewählt.
Für die Methode **setZaehlerstand()** wird mit void, wie für alle set-Methoden vorgeschrieben, festgelegt, dass kein Wert zurückgegeben werden soll. In den Klammern wird für den Übergabeparameter mit dem Namen **z** der Datentyp int festgelegt. Im Methodenrumpf wird das Datenelement **zaehlerstand** auf den im Parameter **z** gespeicherten Wert gesetzt.
Für die Methode **getZaehlerstand()** wird mit int als Rückgabtyp festgelegt, dass das entsprechende Datenelement mit diesem Typ zurückgegeben werden kann. Übergabeparameter werden in den runden Klammern nicht zugelassen. Im Methodenrumpf wird durch return der Wert des Datenelements **zaehlerstand** zurückgegeben.

Methodenliste

Hier werden Methoden definiert, die die eigentlich gewünschten Aufgaben erfüllen. Es ist bei der Programmierung auf die Einhaltung des EVA-Prinzips zu achten. Verarbeitungsmethoden sollten keine Ein- bzw. Ausgaben von Daten aufweisen. Für entsprechende Aufgaben sind separaten Methoden zu programmieren.
In unserem Beispiel gibt es die Methode **eingabe()**, die eine Dateneingabe durch den Benutzer erlauben soll sowie die Methode **ausgabe()**, die eine Bildschirmausgabe der Daten zur Verfügung stellt.

Objektdeklaration

Instanzierung/Deklaration eines Objekts der Klasse Zaehler

```
//--- Nun das Hauptprogramm
public class HPZaehler
{
  public static void main(String[] args)
  {
     Zaehler obj_Zaehler = new Zaehler();      // Deklaration eines Zaehler-Objektes
     int faktor = 0, erg = 0;

     obj_Zaehler.eingabe();                    // Aufruf der Methode zum Einlesen eines Zählerstandes
     obj_Zaehler.ausgabe();                    // Aufruf der Methode zur Ausgabe des Zählerstandes
     System.out.print("\nMultiplikator: ");    // Zaehler soll mit eingegebenen Wert multipliziert werden
     faktor = Tastatur.liesInt();
     erg = obj_Zaehler.getZaehlerstand() * faktor; // Der Attributwert von zaehlerstand wird mit faktor multipliziert
     obj_Zaehler.setZaehlerstand( erg );       // Datenelement zaehlerstand wird auf berechneten Wert gesetzt
     obj_Zaehler.ausgabe();                    // Ausgeben der Daten
  }
}
```

Nur die Methoden einer Klasse dürfen auf die Eigenschaften der Objekte zugreifen.
Objektorientiert wird auch davon gesprochen, dass das Objekt **obj_Zaehler** die Nachricht **eingabe()** erhält.
Insgesamt ist bereits an diesem kleinen Beispiel zu sehen: die Daten und deren Zugriffsfunktionen sind (syntaktisch) so zusammengefasst, wie sie logisch auch zusammengehören. Dabei sind die Daten privatisiert (gekapselt, geschützt), können also nur mittels der klassenintern festgelegten Zugriffsmethoden im Schutzbereich public verändert oder gelesen werden.
Beachten Sie noch den Methodenaufruf. Ganz allgemein kann man mit dem Punktoperator auf Elemente einer Struktur oder eines Objektes zugreifen. Da nun auch Methoden Elemente der Klasse sind, mit der das Objekt angelegt wurde, kann man auch Methoden mit einem Punkt für ein bestimmtes Objekt aufrufen.
Der Compiler weiß an Hand des Punktes, mit welchem Objekt bei diesem Methodenaufruf gearbeitet werden soll und wird dafür sorgen, dass alle Zugriffe auf Eigenschaften oder Methoden innerhalb der Methode korrekt erfolgen. Wegen dieser Dienstleistung des Compilers können Sie innerhalb der Methoden direkt auf die Namen der Elemente zugreifen.
Würden weitere Objekte instantziert, so hätte jedes Objekt seine eigenen Methoden und in diesem Fall sein eigenes Datenelement **zaehlerstand**. Veränderung eines Datenelementwertes eines bestimmten Objektes hätten keine Auswirkungen auf den Datenelementwert der anderen Objekte.
In unserem Beispiel wird im Hauptprogramm HPZaehler ein Objekt **obj_Zaehler** deklariert. Durch den Aufruf des Default-Konstruktors (namens **Zaehler()**) wird dafür gesorgt, dass das Datenelement **zaehlerstand** auf 0 gesetzt wird.
Damit ist bereits nach der Zeile **Zaehler obj_Zaehler = new Zaehler()**; (s. oben) die Instanz **obj_Zaehler** der Klasse **Zaehler** initialisiert, **obj_Zaehler**. **zaehlerstand** ist 0. Dieses Objekt könnte also nun bereits direkt mit dem Aufruf **obj_Zaehler.ausgabe()**; auf den Bildschirm den Datenelementwert ausgeben.
(Beim JavaEditor: Die Datei mit der Klassendefinition und die Datei mit dem Hauptprogramm müssen sich im gleichen Verzeichnis befinden)

Der Parametrisierte-Konstruktor

Java lässt aber mehr als einen Konstruktor zu. Die Klasse **Zaehler** könnte u.a. einen weiteren Konstruktor enthalten, der bereits bei dem Anlegen eines **Zaehler**-Objektes einen bestimmten Wert erhält. Dies kann dann aussehen wie folgt:

```
class Zaehler                          // Beginn der Klassendefinition
{
    :
  public Zaehler(int z)                // Parameterübergabe für die Initialisierung
  {
      set_Zaehlerstand( z );
  }
    :
}
```

Zu dem Defaultkonstruktor **Zaehler( )** ist nun ein weiterer Konstruktor mit dem Konstruktorkopf **Zaehler (int z)** gekommen. Dieser Konstruktor käme in der folgenden Objektdeklaration

```
Zaehler ein_Zaehler = new Zaehler( 1 );
```

zum Tragen. Das Objekt **ein_Zaehler** erhält (über den Parameter **z**) den Wert **zaehlerstand =1** (**z=1** wurde beim Anlegen des Objektes mitgegeben)

```
public class HPZaehler
{
  public static void main(String[] args)
  {
    Zaehler obj_Zaehler = new Zaehler();      // Deklaration von Objekt obj_Zaehler mit dem Defaultkonstruktor
    Zaehler ein_Zaehler = new Zaehler(1);     // Deklaration von Objekt ein_Zaehler mit parametrisiertem
Konstruktor

    int faktor = 0, erg = 0;

    ein_Zaehler.ausgabe();                    // Ausgabe von zaehlerstand des Objekts einZaehler => 1
    obj_Zaehler.ausgabe();                    // Ausgabe von zaehlerstand des Objekts einZaehler => 0
    :
  }
}
```