Objektorientierte Programmierung in Java – Teil 2

Zugriffsrechte und Sichtbarkeit

Bisher griffen wir auf die Eigenschaften eines Objektes - auch von außerhalb - immer direkt zu. Dies widerspricht dem Geiste der OOP. Einerseits wird das Geheimnisprinzip und die Kapselung verletzt, andererseits kann durch den direkten Zugriff auf die Eigenschaften das Objekt in ein unbrauchbaren Zustand gebracht werden.

Beispiel: Ein Objekt der Klasse Datum hat die Eigenschaften Tag, Monat, Jahr jeweils vom Typ int.
Wenn man nun die Eigenschaften ungeschickt ändert, kann man das Objekt ein Datum repräsentieren, dass gar nicht existiert, wie z. B. 29.2.2001 (2001 ist kein Schaltjahr!).

Lösung: Man darf von außerhalb eines Objektes bzw. einer Klasse nicht direkt auf die Eigenschaften zugreifen. In Java erreicht man das durch die *Modifier* `public`, `protected`, `private`. Diese Modifier kann man auch auf Methoden und sogar Klassen anwenden. So kann z. B. verhindern, dass man auf eine Methode zugreift, die nur für die Klasse interessante Berechnungen durchführt.

Die Modifier haben folgende Bedeutungen:

- **public** erlaubt „weltweiten Zugriff" auf die Eigenschaft oder Methode. Jede andere Klasse kann auf sie zugreifen.
  Symbol im UML-Klassendiagramm: +
- **private** erlaubt den Zugriff nur innerhalb der eigenen Klasse. Von außerhalb ist eine solche Eigenschaft oder Methode nicht sichtbar.
  Symbol im UML-Klassendiagramm: -
- **protected** ist ähnelt private, schränkt den Zugriff aber nicht ganz so stark ein. Man kann zwar weiterhin nicht von außerhalb auf die Eigenschaft oder Methode zugreifen, dennoch gibt es gewisse Klasse, die auf protected-Elemente zugreifen können. Welche Klassen das genau sind, werden wir später diskutieren.

Um also korrekt objektorientiert zu arbeiten, müssen also alle Eigenschaften `protected` oder `private` sein. Sie kann man mit Hilfe von sogenannten *Zugriffsmethoden* ansprechen. In Java setzt man Eigenschaften üblicherweise durch Methoden, deren Namen mit `set` beginnen. Den Wert einer Eigenschaft liest man mittels `get`-Methoden. Bei Methoden, die einen Rückgabewert vom Typ boolean besitzen, verwendet man in der Regel statt `get` den Präfix `is`.

Beispiel: Zugriff auf eine Eigenschaft Anschrift vom Typ String

```
...
  private String Anschrift;

  public void setAnschrift(String NeueAnschrift){
    // Gültigkeit von neuer Anschrift sollte hier geprüft werden

    // Wert der Eigenschaft setzen
    Anschrift=NeueAnschrift;
  }

  public String getAnschrift(){
    return Anschrift;
  }
...
```


Datum
- Tag: int; - Monat: int; - Jahr: int;
+ setDatum(int Tag, int Monat, int Jahr); + int getTag(); + int getMonat(); + int getJahr(); + void print();

Aufgaben:

1. Ändere die Klassen vom letzten Aufgabenblatt so ab, dass man nur noch mit Hilfe von Zugriffsmethoden auf die Eigenschaften zugreifen kann.
2. a) Übersetze die rechts in UML dargestellte Klasse nach Java.
   b) Schreibe ein Testprogramm für die Klasse.
   c) Prüfe in der Methode setDatum, ob das neue Datum korrekt ist.

Konstruktoren

Ein Konstruktor ist eine spezielle Methode zum Erzeugen von Objekten. Der Name des Konstruktors entspricht dem der Klasse gefolgt von der Parameterliste in Klammern. Die Parameter dienen zum Initialisieren von Eigenschaften der Klasse.
Wenn kein Konstruktor vom Benutzer definiert wurde, erzeugt Java selbstständig einen Konstruktor ohne Parameter. Man nennt diesen *Standardkonstruktor*. Dieser wurde bei den bisherigen Beispielen immer zum Erzeugen der Klassen verwendet.

Beispiel: Konstruktor für die Klasse Haus (vom letzten Blatt)

```
public Haus(String Anschrift, String Farbe){
  this.Anschrift=Anschrift;
  this.Farbe=Farbe;
}
```

Neu ist in diesem Beispiel das Wort **`this`**. this steht für die konkrete Instanz einer Klasse, d.h. this entspricht also dem aktuellen Objekt. In diesem Fall haben die Parameter des Konstruktors den gleichen Namen wie die Eigenschaften des Objektes. Um sie voneinander zu unterscheiden, verwendet man this. Beispielsweise ist **`this.`**`Anschrift` ist die Eigenschaft des Objektes und `Anschrift` der Parameter. Oben wird der Wert des Parameters der Eigenschaft zugewiesen.

Achtung! Häuser (Objekte der Klasse Haus) können jetzt *nicht mehr* mit **`new`** `Haus();` erzeugt werden, sondern nur noch mit dem selbst definierten Konstruktor.

z.B. **`new`** `Haus("Hauptstr. 5, Bühl","gelb");`

Wenn man aber weiterhin Häuser mit **`new`** `Haus();` erzeugen will, muss man den folgenden Konstruktor ergänzen: **`public`** `Haus(){}`

Aufgabe: Ergänze die Klasse Datum um passende Konstruktoren.

Klassenvariablen und Klassenmethoden

Eigenschaften und Methoden, die zu keinem konkreten Objekt, sondern zu einer Klasse gehören, nennt man Klassenvariablen bzw. Klassenmethoden. In Java werden sie durch den Modifier **`static`** gekennzeichnet. static kommt nach public, protected und private, aber vor dem Rückgabewert.
Die main-Methode oder die Methoden, die wir vor der OOP behandelt haben, sind Klassenmethoden. Es braucht kein konkretes Objekt der Klasse zu existieren, damit man sie verwenden kann. Weitere Beispiele sind z. B. `Math.sqrt` oder `Math.sin`. Klassenmethoden können nur auf Klassenvariablen zugreifen.
Klassenvariablen speichern Werte, die für alle Objekte einer Klasse gelten. Eine typische Anwendung für sie sind Zähler. Mit ihnen kann z.B. geprüft werden, wie viele Objekte einer Klasse erzeugt wurden. Ein anderes Beispiel ist eine Klasse Konto und alle Konten (Objekte der Klasse Konto) einer Bank haben den gleichen Zinssatz.