Übersicht: Zugriff auf Methoden und Eigenschaften (Teil 1)

Wir nehmen an, dass unser Projekt folgende Klasse enthält:

```
public class Spielfigur {
  public String name;
  public int kampfkraft;
  public int schaden;
}
```

Außerdem legen wir ein Objekt der Klasse an, z.B. so:

```
Spielfigur sf = new Spielfigur();
```

Die Klasse Spielfigur enthält noch keine Methoden. Dennoch können wir die drei Eigenschaften direkt ändern, z.B. folgendermaßen:

```
sf.name = "Donald";    sf.kampfkraft = 10;    sf.schaden = 0;
```

Oft möchte man sicherstellen, dass manche Eigenschaften nur bestimmte Werte annehmen können (z.B. soll **`kampfkraft`** und **`schaden`** nur Werte zwischen 0 und 10 annehmen können). Um dies sicherzustellen, könnte man eine „Setter"-Methode programmieren:

```
public void setSchaden(int s) {
    if((s>=0) && (s<= 10)) {
        schaden = s;
    }
}
```

Ein Aufruf von **`sf.setSchaden(-500);`** verhindert dann also, dass die Eigenschaft **`schaden`** einen ungültigen Wert annimmt. Aber: Noch immer wäre es möglich mit **`sf.schaden = -500;`** einen ungültigen Wert zu setzen. Wenn man erzwingen will, dass man nur über die Setter-Methode Werte verändern darf, so macht man die entsprechende Eigenschaft privat (also „von außen nicht sichtbar"), indem man statt **`public`** nun **`private`** schreibt:

```
public class Spielfigur {
  private String name;
  private int kampfkraft;
  private int schaden;

   // ... hier fehlen noch die Methoden

}
```

Die einzelnen Eigenschaften sind nun nicht mehr z.B. mit **`sf.schaden`** ansprechbar – dies würde einen Übersetzungsfehler erzeugen. Stattdessen muss man Setter- und Getter-Methoden benutzen (die man natürlich vorher schreiben muss). Die entsprechende Getter-Methode für **`schaden`** könnte so aussehen:

```
public int getSchaden() {
    return schaden;
}
```

Ebenso würde man mit den anderen Eigenschaften vorgehen.

Übersicht: Zugriff auf Methoden und Eigenschaften (Teil 2)

Wir nehmen an, dass unsere Klasse **`Spielfigur`** nun die drei (privaten) Eigenschaften **`name`**, **`kampfkraft`** und **`schaden`** besitzt und außerdem die dazu passenden Getter- und Setter-Methoden.

Damit man beim Anlegen eines neuen Objektes nicht für jede Eigenschaft extra die passende Setter-Methode aufrufen muss, schreiben wir uns einen sinnvollen Konstruktor, den wir z.B. wie folgt verwenden:

```
Spielfigur sf = new Spielfigur("Donald");
```

Wir erwarten nun, dass eine Spielfigur mit Namen „Donald“ und sinnvollen Anfangswerten für **`kampfkraft`** und **`schaden`** angelegt wurde. Die Anfangswerte sollten aber nicht als „magische Zahl“ irgendwo im Code stehen. Stattdessen könnten wir diese als „spezielle Eigenschaften“ in die Klasse schreiben:

```
public class Spielfigur {
  private String name;
  private int kampfkraft, schaden;
  public int kampfkraftAnfang = 10, schadenAnfang = 0;
}
```

Allerdings haben wir nun zwei „Probleme“:
● Die Anfangswerte sind für alle Spielfigur-Objekte gleich und müssten auch bei tausend verschiedenen Objekten eigentlich nur einmal gespeichert werden. Außerdem sollten die Werte selbst dann abfragbar sein, wenn gar kein Spielfigur-Objekt existiert: Wir deklarieren die Variablen deshalb als **`static`**.
● Wenn die Anfangswerte abfragbar, aber nicht veränderbar sein sollen, so können wir wie bei den anderen Eigenschaften statt **`public`** nun **`private`** verwenden. Dann brauchen wir aber wieder eine Getter-Methode, um den Wert auslesen zu können. Damit die Getter-Methode aber auch aufrufbar ist, wenn noch gar kein Spielfigur-Objekt existiert, muss auch diese **`static`** sein!

Merke: Statische Eigenschaften / Methoden sind von Objekten unabhängig.

```
public class Spielfigur {
  private String name;
  private int kampfkraft, schaden;
  private static int kampfkraftAnfang = 10, schadenAnfang =0;

  public Spielfigur(String n) {
      name = n;
      kampfkraft = kampfkraftAnfang;
      schaden = schadenAnfang;
  }

  public static int getSchadenAnfang() {
      return schadenAnfang;
  }
  // ... hier fehlen noch weitere Methoden
}
```

Auch ohne Spielfigur-Objekt können wir nun woanders Anfangswerte abfragen:

```
int sa = Spielfigur.getSchadenAnfang();
```