Objektorientierte Systementwicklung

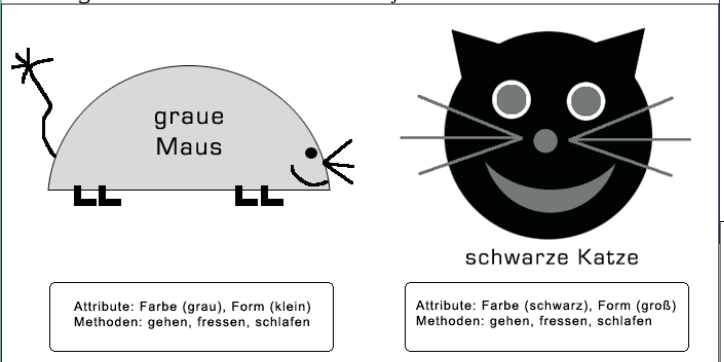
Die Zeit des prozeduralen Denkens in der Programmierwelt ist längst vorbei. Objekte, Klassen, Instanzierung und Vererbung sind angesagt. Man entwickelt Software mit einem neuen Denkansatz – „objektorientiert“ und daher unglaublich effizient.

Thomas Obermayer

Programmiersprachen wie C++, JAVA oder SMALLTALK sind in aller Munde. Sie unterscheiden sich von solchen wie C oder BASIC durch eine wichtige Eigenschaft: Sie sind „objektorientiert“. Man sagt, dies würde „natürliches“ Denken unterstützten und dadurch zu hoher Effizienz bei der Programm-Entwicklung verhelfen. Was „objektorientiert“ heißt und warum von einem „natürlichen“ Konzept gesprochen wird, soll dieser Artikel erklären.

Objekte (Instanzen)

Die Bezeichnung „objektorientiert“ lässt schon vermuten, dass „Objekte“ („Instanzen“) die zentralen Bausteine des Denkansatzes sind. Dabei handelt es sich um „Dinge“ in unserer Welt, die sich eindeutig definieren lassen und eine klare Abgrenzung zu anderen Objekten haben (z.B. „`die graue Maus`“, „`die schwarze Katze`“, etc). Auch Menschen und Tiere werden als Objekte aufgefasst.Wir beurteilen die Objekte unserer Welt nach zwei



Objekte („graue Maus“, „schwarze Katze“) mit Attributen („Farbe“, „Form“) und Methoden („gehen“, „fressen“, „schlafen“)

Kriterien: zum einen nach statischen Merkmalen, wie „`Farbe`“ oder „`Form`“ und zum anderen nach bestimmten Verhaltensweisen, die ein Objekt aufweist. So könnte das Verhalten einer Katze beispielsweise durch Funktionen, wie „`gehen`“, „`fressen`“ und „`schlafen`“ modelliert werden.

Objektorientiert betrachtet, spricht man bei den Eigenschaften von „Attributen“ oder „Instanzvariablen“ und bei den Funktionen von „Methoden“.

Klassen, Objekte und Instanzierung

Irgendwann als kleines Kind erfährt man, was ein Tier ist. Im Laufe der Zeit lernt man unterschiedliche Tiere kennen und begreift, dass alle Tiere gewisse Eigenschaften (z.B. „Farbe“, „Form“, etc) bzw. Methoden (z.B. „`gehen`“, „`fressen`“, etc) gemein haben.

Wir haben die Fähigkeit, alle Tiere zu klassifizieren, sie also einer bestimmten „Klasse“ (z.B. „`Tier`“) zuzuordnen, und trotzdem jedes einzelne Tier als Individuum von anderen zu unterscheiden (z.B. „`Maus`“, „`Katze`“). Wir wissen, dass alle Tiere gehen können, erkennen aber, dass eine Katze andere Eigenschaften (z.B. „`Farbe`“, „`Form`“, etc) als eine Maus besitzt.

Dieser Vorstellungswelt entspringt das Konzept der Klassen und ihrer Objekte in der objektorientierten Systementwicklung.

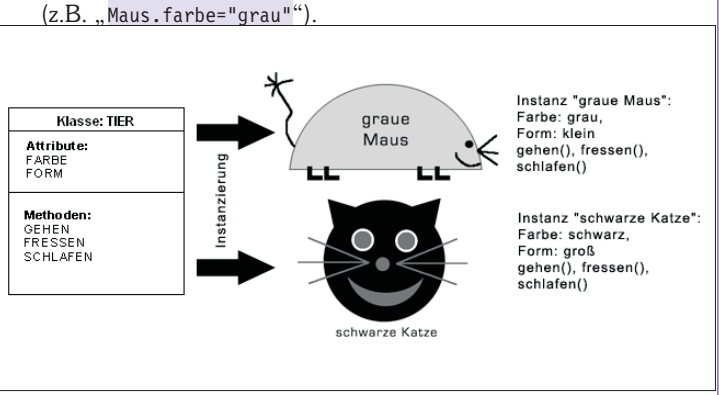
Es gibt Klassen (z.B. „`Tier`“), die man sich als eine Art Vorlage zur Erstellung konkreter Objekte (z.B. „`graue Maus`“, „`schwarze Katze`“) vorstellen kann. Durch sogenannte „Instanzierung“ werden „Objekte“ („Instanzen“) erstellt, deren Attribute und Methoden der Klasse entnommen sind. Während der Objekterstellung werden die jeweiligen Attribute mit Daten „gefüllt“, z.B. bekommt die Eigenschaft „`Farbe`“ der Katze den Wert „`schwarz`“.

Die in der Klasse definierten Attribute existierten pro Instanz einmal und werden häufig auch als „Instanzvariablen“ bezeichnet, d.h. Maus und Katze besitzen jeweils das Attribut „`Farbe`“, wobei die Farbe der Maus völlig unabhängig von der Farbe der Katze ist.

Auch Methoden, wie „`gehen`“, „`fressen`“ und „`schlafen`“ kommen pro Instanz einmal vor. Eine Maus besitzt also eine Methode „`gehen`“, die mit der „`gehen`“-Methode einer Katze nichts zu tun hat.

So wird die Programmierung von Anwendungen, wie z.B. Spielen sehr einfach. Da jede Instanz ihr eigenes Verhalten und somit ihre eigenen Methoden besitzt, ist es möglich, verschiedene Objekte, wie „`Lemminge`“, „`Tiere`“ und „`Feinde`“ völlig unabhängig voneinander auf dem Bildschirm herumlaufen zu lassen. In diesem Konzept liegt die wesentliche Stärke des objektorientierten Ansatzes.

Üblicherweise erfolgen Methodenaufrufe in Programmiersprachen durch Angabe des Instanznamens, einem Trennpunkt und der Methodenbezeichnung (z.B. „`katze.gehen()`“, „`maus.gehen()`“). Der Zugriff auf Attribute funktioniert ähnlich: Dem Instanznamen folgt ein Punkt und der Name des Attributes (z.B. „`Maus.farbe="grau"`“).



Klassen („Tier“), Instanzierung, Objekte („graue Maus“, „schwarze Katze“) mit eigenem Verhalten

Vererbung

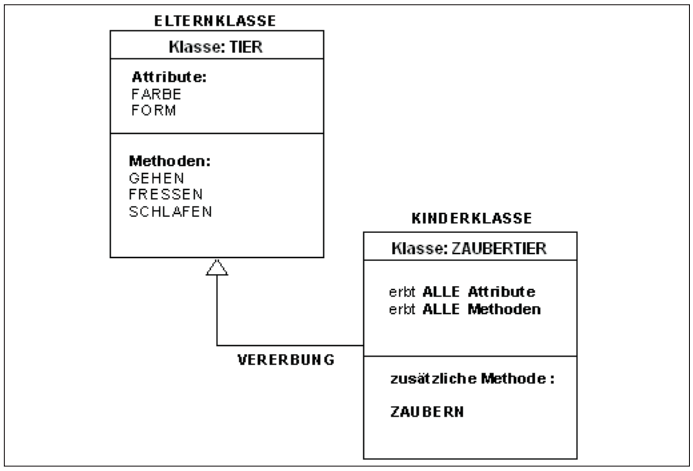
Unsere Natur sieht vor, dass Kinder bestimmte Eigenschaften und Fähigkeiten von Ihren Eltern erben. Auch im objektorientierten Entwicklungskonzept gibt es Vererbung.

Dabei erbt eine „Subklasse“ (bzw. „Unterklasse“ oder „Kinder-Klasse“) von einer „Superklasse“ (bzw. „Oberklasse“ oder „Eltern-Klasse“) alle Attribute und Methoden. Zusätzlich besteht die Möglichkeit, dass die Subklasse weitere Attribute und Eigenschaften einführt. Man spricht davon, dass die Subklasse die Superklasse „erweitert“ bzw. „spezialisiert“.

Beispielsweise könnte es ein spezielles „`Tier`“ geben, das zaubern kann. Die zugehörige Klasse soll „`ZauberTier`“ heißen und eine zusätzliche Methode „zaubern“ besitzen. Gibt man an, dass „`ZauberTier`“ von „`Tier`“ erben soll (oder anders: „`ZauberTier`“ soll „`Tier`“ „erweitern“ bzw. „spezialisieren“), dann ist ZauberTier automatisch ein Tier, besitzt dessen Attribute und Methoden und hat eine weitere Methode, nämlich „`zaubern`“. Wichtig: In einer Vererbungshierachie ist die „ist ein“-Beziehung immer erfüllt (z.B. „das ZauberTier ist ein Tier“).

Mit Vererbung wird die objektorientierte Entwicklung zu einem naturorientierten Prozess. Viele Vorgänge in unserer Welt lassen

sich so sehr gut modellieren, wobei Verständlichkeit und Natürlichkeit erhalten bleiben. Das Programmbeispiel am Ende des Artikels zeigt Ihnen die praktische Anwendung dieses Konzepts.



Polymorphie

Da jede Klasse eigenständig ist und eine klare Abgrenzung zu anderen besitzt, ist es möglich, dass unterschiedliche Klassen Methoden gleichen Namens besitzen (z.B. könnte es sowohl in der Klasse „`Tier`“ als auch in der Klasse „**`Mensch`**“ eine Methode „`gehen`“ geben). Dieser Sachverhalt wird in der objektorientierten Welt als „Polymorphie“ (griech. „Vielgestaltigkeit“) bezeichnet.

Sinnvoll erweist sich dies, wenn man eine ganze Klassen-Bibliothek erstellt und für einfache Handhabbarkeit sorgen möchte. Oft haben verschiedene Klassen gleiche Fähigkeiten und es wäre verwirrend, diese Fähigkeiten über Methoden unterschiedlichen Namens zugänglich zu machen. Daher ist es zweckmäßig, gleiche Fähigkeiten durch gleichnamige Methoden zu realisieren.

Auch in Vererbungshierachien findet Polymorphie Anwendung. Wenn eine Subklasse eine Superklasse spezialisiert, so erbt sie automatisch alle Methoden der Superklasse (z.B. erbt die Klasse „`ZauberTier`“ die Methoden „`gehen`“, „`fressen`“ und „`schlafen`“ von der Klasse „`Tier`“). Trotzdem ist es möglich, dass in der Unterklasse eine Methode definiert wird, die den gleichen Namen trägt, wie eine Methode der Oberklasse (z.B. eine Methode „`fressen`“ in der Klasse „`ZauberTier`“). Die Methode der Subklasse „überschreibt“ die Methode der Superklasse, d.h. der Inhalt wird durch einen neuen ersetzt und die Methode der Oberklasse wird aus Sicht der Unterklasse irrelevant.

Programmiersprachen

Die Programmiersprachen lassen sich unterscheiden in rein objektorientierte Sprachen, die nur Klassen, Objekte und die Kommunikation zwischen Objekten kennen, und in sogenannte „hybride“ Sprachen. Letztere bauen auf konventionellen Programmiersprachen auf. Sie ergänzen rein prozedurale Sprachen um Elemente, die objektorientierte Entwicklung ermöglichen. Beispiele für hybride Sprachen sind C++, OO-Cobol und Object-PASCAL; rein objektorientierte Sprachen sind JAVA und SMALLTALK.

OOS und JAVA

Folgender Code soll auf einfache Art und Weise zeigen, wie objektorientierte Systementwicklung in der Praxis aussieht. Als Programmiersprache wird JAVA verwendet, wobei nur die grundlegenden Konzepte behandelt werden. Das Programm wird durch Kommentare (vorangestellte Schrägstriche) erklärt. Bevor Sie den Code durchdenken, sollten Sie den Artikel gelesen haben.

```
// Definition der Klasse Tier: Attribute und Methoden

class Tier {
     // Attribute
     String farbe;  //die Farbe wird als Zeichenkette gespeichert
     String form;

     // Methoden

     //gehen
     public void gehen() {
          // ...Implementierung der Methode gehen() ....
     }

     //fressen
     public void fressen() {
          //...Implementierung der Methode fressen() ....
     }

     //schlafen
     public void schlafen() {
          //...Implementierung der Methode schlafen() ....
     }
}

// ZauberTier ist ein spezielles Tier, das zaubern kann.
// Vererbung wird durch das Schlüsselwort „extends"
// in der Klassendefinition angezeigt.
// ZauberTier besitzt automatisch
// alle Attribute und Methoden von „Tier".
// Die zusätzlich angegebene Methode „zaubern"
// spezialisiert die Klasse „Tier".

class ZauberTier extends Tier {

     public void zaubern() {
          //...Implementierung der Methode zaubern() ....
     }
}

// Die Klasse Tierpark dient nur als Startpunkt
// (ein JAVA-Programm beginnt mit der main()-Methode,
// die in irgendeiner Klasse stehen kann).
// Im Tierpark sollen 2 Tiere leben,
// die abwechselnd gehen, fressen und schlafen.
// Später entsteht ein ZauberTier,
// das zuerst geht und dann zaubert.

class Tierpark {
     //Hier startet das Programm
     public static void
     main( String args[] ) {
          Tier katze = new Tier(); //Instanzierung: Katze ist ein Tier
          Tier maus = new Tier();  //Instanzierung: Maus ist ein Tier

          katze.gehen();      //die Katze geht, die Maus noch nicht
          maus.fressen();     //die Maus frisst

          katze.fressen();    //jetzt fressen beide Tiere
          maus.schlafen();    //nun schläft die Maus

          katze.schlafen();   //und nun auch die Katze...

          //Hier entsteht ein ZauberTier durch Instanzierung
          ZauberTier magicpet = new ZauberTier();

          magicpet.gehen();   //magicpet kann gehen,
                              //da es ein Tier ist
          magicpet.zaubern(); //..und zaubern,
                              //weil es ein ZauberTier ist
     }
}
```

Obwohl die Syntax von JAVA vielleicht an C oder C++ erinnert, ist JAVA eine andere Sprache mit teils völlig unterschiedlichen Konzepten. Es handelt sich insgesamt um eine sehr effiziente Sprache, die einen hohen Abstraktionslevel bei der Programmierung ermöglicht. Ein großer Vorteil von JAVA ist die Plattform-Unabhängigkeit des fertigen Programms.