2

Objektorientierte Programmierung – ein erster Eindruck

Woche 1

Objektorientierte Programmierung ist fast wie Bier.

Die meisten, die das erste Mal ein Glas des malzigen Getränks zu sich nehmen, mögen es nicht und stellen unter Umständen die Zurechnungsfähigkeit derjenigen in Frage, die ein Loblied auf dieses Getränk singen. »Was hab' ich Dir angetan«, fragen sie, »daß Du mich dieses Höllenzeug trinken läßt?«

Nach einiger Zeit kann es durchaus sein, daß die, die trotzdem weiter Bier trinken, es schätzen lernen. (Bei manchen wird dieser Zeitraum Studium genannt.)

Objektorientierte Programmierung ist, wie Bier, ein Geschmack, an den man sich gewöhnen muß. Es ist zum einen eine der bemerkenswertesten Ideen der Programmierung, die in den letzten Jahren eingeführt wurden, und zum anderen der Quell großer Bestürzung bei den Programmierern, die damit nicht vertraut sind.

In gewisser Weise ist dieser Ruf gerechtfertigt. Objektorientierte Programmierung, auch OOP genannt, ist ein Fach, das man jahrelang studieren und üben kann. Die Grundidee ist allerdings einfach: Organisieren Sie Ihre Programme auf eine Art, die die Art widerspiegelt, mit der Objekte in der realen Welt organisiert sind.

Heute werden Sie einen ersten Eindruck davon erhalten, wie Java die Prinzipien der objektorientierten Programmierung verinnerlicht. Die folgenden Themen werden wir behandeln:

- Programme in Form von sogenannten Klassen organisieren und wie diese Klassen verwendet werden, um Objekte zu erzeugen.
- Eine Klasse über zwei Aspekte ihrer Struktur entwerfen: wie sie sich verhalten und über welche Attribute sie verfügen soll.
- Klassen so miteinander verbinden, daß eine Klasse die Funktionalität von einer anderen erbt.
- Klassen über Pakete und Schnittstellen miteinander verbinden.

Wenn Sie mit der objektorientierten Programmierung bereits vertraut sind, wird vieles der heutigen Lektion eine Wiederholung für Sie sein. Selbst, wenn Sie die einführenden Abschnitte überspringen wollen, sollten Sie das Beispielprogramm erstellen, um Erfahrung bei der Erzeugung Ihres ersten Applets zu sammeln.

In Objekten denken

Objektorientierte Programmierung ist in ihrem Kern eine Methode, Computerprogramme zu strukturieren. Sie stellen sich ein Computerprogramm eventuell als Liste von Anweisungen vor, die dem Computer mitteilen, was er zu tun hat, oder als eine Menge kleiner Programme, die auf bestimmte Ereignisse, die der Benutzer auslöst, reagieren.

Die OOP sieht ein Programm auf eine völlig andere Art. Hier ist ein Programm eine Reihe von Objekten, die in vordefinierter Art und Weise zusammenarbeiten, um bestimmte Aufgaben zu erledigen. Nehmen wir LEGO-Steine als Beispiel zur Verdeutlichung.

LEGO-Steine sind – für die unter Ihnen, die keine Kinder haben oder kein inneres Kind, das beschäftigt werden muß – kleine Plastikblöcke, die in unterschiedlichsten Farben und Größen verkauft werden. Diese Steine haben kleine, runde Noppen auf der einen Seite, die fest in die entsprechenden Löcher anderer Steine passen. Über Kombinationen dieser Steine lassen sich größere Formen erzeugen. Es gibt viele verschiedene LEGO-Teile, wie z.B. Räder, Motoren, Gelenke und Flaschenzüge, die man dazu verwenden kann.

Mit LEGO-Bausteinen können Sie alle möglichen Dinge bauen: Burgen, Autos, lange Anhänger, Hosenträger, Sportkleidung... einfach alles, was Sie sich vorstellen können. Jedes LEGO-Steinchen ist ein Objekt, das mit anderen Objekten auf eine ganz bestimmte Art zusammenpaßt, um ein größeres Objekt zu erzeugen.

Nehmen wir ein anderes Beispiel. Mit ein bißchen Erfahrung und Hilfe können Sie in den nächsten Computerladen gehen und sich einen kompletten PC aus verschiedenen Einzelteilen zusammenbauen: Motherboard, CPU, Grafikkarte, Festplatte, Tastatur, usw. Idealerweise erhalten Sie, nachdem Sie die einzelnen Komponenten zusammengesetzt haben, ein System, in dem alle Einheiten zusammenarbeiten, um ein größeres System zu bilden. Sie können dieses größere System dann vorrangig zur Lösung der Probleme verwenden, für die Sie den Computer gekauft haben.

Intern kann jede dieser Komponenten ziemlich komplex sein. Auch können sie von verschiedenen Firmen mit unterschiedlichen Methoden entwickelt worden sein. Allerdings müssen Sie nicht wissen, wie die einzelnen Komponenten funktionieren, was jeder einzelne Chip auf der Platine tut oder wie ein »A« an Ihren Computer geschickt wird, wenn Sie auf die [A]-Taste drücken. Jede Komponente, die Sie verwenden, ist eine abgeschlossene Einheit, und als derjenige, der das Gesamtsystem zusammenbaut, sind Sie nur daran interessiert, wie die einzelnen Einheiten miteinander interagieren:

- Wird diese Grafikkarte in einen Slot auf dem Motherboard passen?
- Wird dieser Monitor mit dieser Grafikkarte zusammenarbeiten?
- Werden die einzelnen Komponenten die richtigen Kommandos an die Komponenten senden, mit denen sie zusammenarbeiten, so daß sich die einzelnen Teile des Computers miteinander verstehen?

Sobald Sie die Interaktionen zwischen den einzelnen Komponenten kennen und die entsprechenden Voraussetzungen dafür schaffen, ist es einfach, das Gesamtsystem zusammenzusetzen.

Objektorientierte Programmierung hat sehr viel Ähnlichkeit mit dem Aufbau von Strukturen mit LEGO-Steinen oder dem Zusammenbau eines PC. Bei der OOP bauen Sie Ihre Gesamtprogramme aus unterschiedlichen Komponenten auf, die Objekte genannt werden.

Ein *Objekt* ist ein abgeschlossenes Element eines Computerprogramms, das eine Gruppe miteinander verwandter Features darstellt und dafür ausgelegt ist, bestimmte Aufgaben zu erfüllen. Objekte werden auch als *Instanzen* bezeichnet.

Jedes Objekt hat eine spezielle Rolle in einem Programm, und alle Objekte können auf definierte Arten in einem Programm zusammenarbeiten.

Objekte und Klassen

Die objektorientierte Programmierung wird nach der Beobachtung modelliert, daß in der realen Welt Objekte aus vielen Arten kleinerer Objekte aufgebaut sind. Die Fähigkeit, Objekte zu kombinieren, ist allerdings nur ein allgemeiner Aspekt der objektorientierten Programmierung. Sie umfaßt außerdem Konzepte und Features, die die Erzeugung und den Umgang mit Objekten einfacher und flexibler machen. Das wichtigste dieser Features ist die Klasse.

Eine *Klasse* ist eine Vorlage, die zur Erzeugung vieler Objekte mit ähnlichen Eigenschaften verwendet wird.

Klassen umfassen alle Features eines bestimmten Satzes von Objekten. Wenn Sie ein Programm in einer objektorientierten Sprache schreiben, dann definieren Sie nicht einzelne Objekte, sonder Sie definieren Klassen von Objekten.

Nehmen Sie z.B. die Klasse `Tree` (engl. Baum), die alle Features aller Bäume beschreibt:

- Hat Blätter und Wurzeln
- Wächst
- Erzeugt Chlorophyll

Die Klasse `Tree` dient als abstraktes Modell für das Konzept »Baum«. Um ein tatsächliches Objekt in einem Programm zur Verfügung zu haben, das man manipulieren kann, benötigt man konkrete Instanzen der `Tree`-Klasse.

Klassen werden dazu verwendet, Objekte zu erstellen. Mit diesen Objekten arbeiten Sie dann direkt in einem Programm. Die Klasse `Tree` kann dazu verwendet werden, eine Vielzahl unterschiedlicher `Tree`-Objekte zu schaffen, die alle unterschiedliche Features haben:

- Klein oder groß
- Sehr dichtes Astwerk oder nur sehr spärliches
- Mit Früchten oder ohne

Obwohl diese Objekte sich alle voneinander unterscheiden, haben sie dennoch genug gemein, daß man die Verwandschaft zwischen ihnen sofort erkennt. Abbildung 2.1 zeigt die `Tree`-Klasse und einige Objekte, die von dieser Vorlage erzeugt wurden.

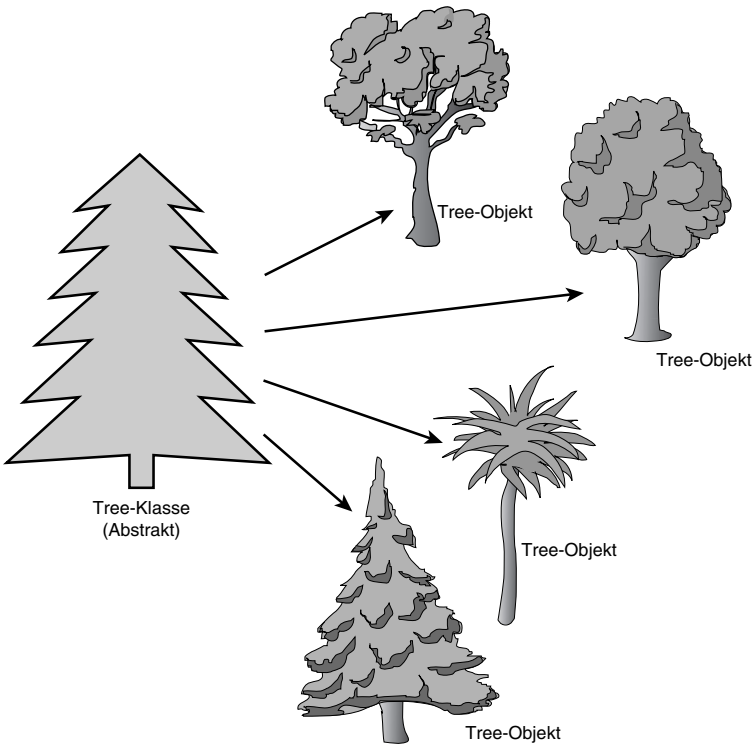

Abbildung 2.1: Die Tree-Klasse und einige Tree-Objekte

Ein Beispiel für den Entwurf einer Klasse

In einem Beispiel, das eher dem entspricht, was Sie wahrscheinlich mit Java machen werden, könnten Sie eine Klasse für eine Schaltfläche erzeugen. Dies ist ein Element,

das in Fenstern, Dialogen und anderen interaktiven Programmen verwendet wird. Die folgenden Eigenschaften könnte die Klasse `CommandButton` definieren:

- Den Text, der den Zweck der Schaltfläche kennzeichnet.
- Die Größe der Schaltfläche.
- Eigenschaften der Erscheinung, wie z.B. ob die Schaltfläche über einen 3D-Schatten verfügt oder nicht.

Die Klasse `CommandButton` könnte zusätzlich noch definieren, wie sich eine Schaltfläche verhalten soll:

- Ob die Schaltfläche einfach oder doppelt angeklickt werden muß, um eine Aktion auszulösen.
- Ob sie Mausklicks eventuell komplett ignorieren soll.
- Was sie tun soll, wenn sie erfolgreich angeklickt wurde.

Sobald Sie die Klasse `CommandButton` definiert haben, können Sie Instanzen der Schaltfläche erzeugen – mit anderen Worten `CommandButton`-Objekte. Die Objekte besitzen alle die Features einer Schaltfläche, wie es in der Klasse definiert ist. Allerdings kann jede Schaltfläche eine andere Erscheinung haben abhängig davon, was für eine Schaltfläche konkret realisiert werden soll.

Eine der Standardklassen von Java – `java.awt.Button` – beinhaltet die gesamte Funktionalität des hypothetischen `CommandButton`-Beispiels und noch mehr. Sie werden am Tag 11 Gelegenheit bekommen, mit dieser Klasse zu arbeiten.

Wenn Sie ein Java-Programm schreiben, dann entwerfen und erstellen Sie eine Reihe von Klassen. Wenn Ihr Programm ausgeführt wird, werden Objekte dieser Klassen erzeugt und nach Bedarf verwendet. Ihre Aufgabe als Java-Programmierer ist es, die richtigen Klassen zu entwerfen, um das umzusetzen, was Ihr Programm tun soll.

Glücklicherweise müssen Sie nicht bei Null beginnen. Jede Version von Java umfaßt eine Gruppe von Klassen, die einen Großteil der Basisfunktionalität, die Sie benötigen, implementieren. Solche Gruppierungen werden als Bibliotheken bezeichnet.

Eine *Klassenbibliothek* ist eine Gruppe von Klassen, die zur Verwendung mit anderen Programmen entworfen wurden. Die Standard-Java-Klassenbibliothek beinhaltet Dutzende von Klassen.

Wenn Sie über die Anwendung der Sprache Java sprechen, dann sprechen Sie eigentlich über die Verwendung der Java-Klassenbibliothek und einiger Schlüsselwörter und Operatoren, die vom Java-Compiler erkannt werden.

Javas Standardbibliothek hat eine große Zahl von Aufgaben, wie z.B. mathematische Funktionen, Umgang mit Text, Grafik, Sound, Interaktion mit dem Benutzer und den Zugriff auf Netzwerke. In vielen Fällen wird die Java-Klassenbibliothek für Ihre Anforderungen ausreichend sein. Hier ist es dann Ihre Aufgabe, eine einzige Klasse zu erstellen, die die Objekte der Standardklassen erzeugt und deren Interaktionen koordiniert.

Für komplexe Java-Programme müssen Sie eventuell eine ganze Reihe neuer Klassen mit definierten Interaktionsmöglichkeiten erzeugen. Diese können Sie dazu verwenden, Ihre eigene Klassenbibliothek zu erstellen, die Sie später auch in anderen Programmen verwenden können.

Die Wiederverwendung ist einer der fundamentalen Vorzüge der objektorientierten Programmierung.

Attribute und Verhaltensweisen

Im allgemeinen besteht jede Klasse, die Sie schreiben, aus zwei Komponenten: Attributen und Verhaltensweisen. In diesem Abschnitt lernen Sie die beiden Komponenten kennen, wie sie in einer theoretischen Klasse mit dem Namen `Jabberwock` angewendet werden. Um diesen Abschnitt zu vervollständigen, werden Sie eine Java-Klasse erstellen, die einen Jabberwock – ein drachenartiges Monster aus dem Gedicht *Jabberwocky* von *Lewis Carroll* – implementiert.

Attribute einer Klasse

Attribute sind die einzelnen Dinge, die die einzelnen Klassen voneinander unterscheiden. Sie legen auch die Erscheinung, den Zustand und andere Qualitäten der Klasse fest. Überlegen wir uns einmal, wie die theoretische Klasse `Jabberwock` erstellt werden könnte. Ein Jabberwock könnte unter anderem die folgenden Attribute aufweisen:

- `color` (Farbe): Orange, Dunkelbraun, Zitronengelb, Dunkelgelb
- `sex` (Geschlecht): männlich, weiblich
- `hungry` (hungrig): ja, nein

Die Attribute einer Klasse können auch Informationen über den Zustand eines Objektes umfassen. Sie könnten z.B. ein Attribut für den Gemütszustand (wütend oder ruhig) des Jabberwock, den Gesundheitszustand (lebendig oder tot) und das Wahlverhalten (konservativ, liberal oder alternativ) festlegen.

In einer Klasse werden Attribute über Variablen definiert. Sie können sich diese analog zu globalen Variablen für jedes Objekt einer Klasse vorstellen. Jedes Objekt kann andere Werte in seinen Variablen speichern, weshalb diese Variablen auch *Instanzvariablen* genannt werden.

Eine *Instanzvariable* ist ein Stück Information, das ein Attribut eines Objekts definiert. Die Klasse des Objekts definiert die Art des Attributs, und jede Instanz speichert ihren eigenen Wert für dieses Attribut. Instanzvariablen werden auch als *Objektvariablen* bezeichnet.

Jedes Attribut einer Klasse besitzt eine dazugehörige Variable. Sie ändern also dieses Attribut in einem Objekt, indem Sie den Wert dieser Variablen ändern. In dem Programm, das Sie etwas weiter unten erstellen werden, wird die folgende Anweisung verwendet, um anzuzeigen, daß ein `Jabberwock`-Objekt nicht mehr hungrig ist:

```
j.hungry = false;
```

Instanzvariablen kann bei der Erzeugung eines Objekts ein Wert zugewiesen werden, der während der Lebenszeit des Objekts konstant bleibt. Auf der anderen Seite können Instanzvariablen auch verschiedene Werte zugewiesen werden, während das Objekt in einem laufenden Programm verwendet wird.

Ein anderer Typ von Attributen wird verwendet, um eine ganze Klasse von Objekten anstelle einzelner Objekte dieser Klasse zu beschreiben. Diese werden *Klassenvariablen* genannt.

Eine *Klassenvariable* ist ein Stück Information, das ein Attribut einer Klasse definiert. Die Variable bezieht sich auf die Klasse selbst und all ihre Instanzen, so daß nur ein Wert gespeichert wird unabhängig davon, wie viele Objekte dieser Klasse erzeugt wurden.

Ein gutes Beispiel für eine Klassenvariable ist eine Variable, die zum Zählen der einzelnen `Jabberwock`-Objekte, die in einem Programm erzeugt wurden, verwendet wird. Wenn für diese Aufgabe eine Instanzvariable in der `Jabberwock`-Klasse verwendet werden würde, könnte jedes Objekt einen anderen Zählerstand aufweisen. Damit aber nur ein Wert gespeichert werden muß, wird eine Klassenvariable verwendet, auf die jedes `Jabberwock`-Objekt Zugriff haben kann.

Verhaltensweisen einer Klasse

Das Verhalten ist die Art, mit der eine Klasse bestimmte Dinge gegenüber sich selbst oder anderen Objekten ausführt. Das Verhalten einer Klasse legt fest, was die Objekte dieser Klasse tun, um deren Attribute zu verändern oder wenn andere Objekte sie bitten, etwas zu tun. Ein `Jabberwock`-Objekt könnte die folgenden Verhaltensweisen beinhalten:

- Ärgerlich werden
- Sich beruhigen
- Einen Bauern fressen

- Das Abendessen ausfallen lassen
- Gesund werden

Das Verhalten einer Klasse wird durch Methoden definiert.

Methoden sind Gruppen von miteinander in Beziehung stehenden Anweisungen in einer Klasse. Diese Anweisungen beziehen sich auf die eigene Klasse und auf andere Klassen und Objekte. Sie werden verwendet, um bestimmte Aufgaben zu erledigen, wie das in anderen Programmiersprachen Funktionen tun.

Objekte kommunizieren miteinander über Methoden. Eine Klasse oder ein Objekt kann Methoden in anderen Klassen oder Objekten aus vielen unterschiedlichen Gründen aufrufen, darunter die folgenden:

- Um ein anderes Objekt von einer Änderung zu berichten.
- Um ein anderes Objekt anzuweisen, etwas an sich selbst zu ändern.
- Um ein anderes Objekt zu bitten, etwas zu tun.

Denken Sie z.B. an den Schwertkämpfer in dem Gedicht »Jabberwocky«. Der folgende Auszug aus dem Gedicht von Lewis Carroll beschreibt genau, was passiert, als der Schwertkämpfer den Jabberwock mit seinem Schwert angreift:

»One, two! One, two! And through and through

The vorpal blade went snicker-snack!

He left it dead, and with its head

He went galumphing back.«

Im Deutschen heißt das soviel wie, der Ritter schlägt mit seinem Schwert dem Jabberwock den Kopf ab und kehrt dann mit dem Kopf im Gepäck dorthin zurück, wo er hergekommen ist.

In Java könnte der Schwertkämpfer als `Knight`-Objekt (Knight engl. für Ritter) erzeugt werden, mit der `Knight`-Klasse als Vorlage dafür, wie ein `Knight`-Objekt sein sollte. Wenn der Ritter dem Jabberwock den Kopf abschlägt, hat dies eine Änderung des inneren Zustandes des Jabberwock zur Folge. Um dieser Änderung Rechnung zu tragen, würde das `Knight`-Objekt eine Methode verwenden, um dem `Jabberwock`-Objekt mitzuteilen »Hey! Ich habe Dir Deinen Kopf abgeschlagen. Du bist tot.«

So wie zwischen Instanz- und Klassenvariablen unterschieden wird, gibt es auch Instanz- und Klassenmethoden. Instanzmethoden, die so häufig verwendet werden, daß sie einfach nur Methoden genannt werden, gehören zu einem Objekt einer Klasse. Wenn eine Methode ein einzelnes Objekt verändert, dann muß diese eine Instanzmethode sein. Klassenmethoden gehören zu einer Klasse selbst.

Eine Klasse erstellen

Nachdem die grundlegende Terminologie der objektorientierten Programmierung nun eingeführt wurde, wird alles vielleicht anhand eines konkreteren Beispiels klarer. Sie werden ein funktionierendes Beispiel der `Jabberwock`-Klasse erstellen, damit Sie sehen, wie Instanzvariablen und Methoden in einer Klasse definiert werden. Sie werden auch ein Java-Applet erstellen, das ein neues Objekt der `Jabberwock`-Klasse erzeugt, die Werte der Instanzvariablen dieses Objektes verändert und bestimmte Aktionen aufgrund der Werte der Instanzvariablen ausführt.

Auf die eigentliche Syntax dieses Beispiels wird hier nicht sehr ausführlich eingegangen. Sehen Sie dies hier mehr als Einführung in die objektorientierte Programmierung und weniger als eine Lektion über die Syntax von Java, in die Sie am dritten Tag eintauchen werden.

Öffnen Sie den Texteditor, den Sie zur Erstellung von Java-Programmen verwenden, so daß Sie mit der Erzeugung der Quelldatei beginnen können. Anstatt ein ganzes Programm einzugeben, werden Sie einige Anweisungen eingeben, während Sie etwas über deren Verwendung lernen. Sie erhalten am Ende die Gelegenheit, Ihre Arbeit zu überprüfen, um sicherzugehen, daß alles korrekt ist.

Den Beginn stellt eine elementare Klassendefinition dar. Geben Sie folgendes ein:

```
class Jabberwock {
}
```

Nun haben Sie eine Klasse erzeugt. Momentan tut sie noch nicht viel, aber die beiden Zeilen sind ein Beispiel für die einfachste Klassendefinition in Java.

Um `Jabberwock` etwas spezieller zu gestalten, erstellen Sie drei Instanzvariablen für diese Klasse. Direkt unterhalb der Zeile `class Jabberwock {` fügen Sie die folgenden drei Zeilen ein:

```
String color;
String sex;
boolean hungry;
```

Diese drei Zeilen erzeugen drei Instanzvariablen. Zwei davon, `color` (Farbe) und `sex` (Geschlecht), können `String`-Objekte beinhalten. Ein String ist ein allgemeiner Begriff, der für eine Gruppe von Zeichen steht. In Java wird ein `String`-Objekt mit einer der Standardklassen aus der Java-Klassenbibliothek erzeugt. Die `String`-Klasse wird zur Speicherung von Text verwendet und bietet diverse Funktionen zur Bearbeitung von Text.

Die dritte Variable, `hungry` (hungrig), ist eine boolesche Variable, die nur zwei verschiedene Werte annehmen kann: `true` (wahr) oder `false` (falsch). Dieses Objekt wird dafür verwendet, anzuzeigen, ob der Jabberwock hungrig ist (`true`) oder voll (`false`).

Boolesche Werte sind spezielle Variablentypen, die nur die Werte `true` oder `false` aufnehmen können. Im Gegensatz zu anderen Sprachen haben boolesche Werte keine numerischen Werte, wobei `1 true` und `0 false` entspricht. Der Ausdruck boolesch geht auf George Boole, einen irischen Mathematiker zurück, der von 1815 bis 1864 lebte. Ein anderer Begriff, der auf ihn zurückgeht, ist die Boolesche Algebra, die fundamental für die Programmierung von Computern, die digitale Elektronik und die Logik ist.

Sie können der `Jabberwock`-Klasse Verhaltensweisen hinzufügen, in dem Sie Methoden implementieren. Ein Jabberwock kann alle möglichen Dinge tun (mit den Krallen zupacken, zubeißen und so weiter); um das Ganze kurz zu halten, wollen wir nur zwei Methoden hinzufügen – eine, die das Monster füttert, und eine andere, um die Attribute des Monsters zu überprüfen.

Zum Start fügen Sie die folgenden Zeilen unter den drei Instanzvariablen in Ihrer Klassendefinition ein:

```
void feedJabberwock() {
    if (hungry == true) {
        System.out.println("Yum -- a peasant.");
        hungry = false;
    } else
        System.out.println("No, thanks -- already ate.");
}
// In Kürze mehr
```

Die letzte Zeile `// In Kürze mehr` ist eine Kommentarzeile. Kommentare werden als Information für diejenigen eingefügt, die den Quellcode lesen, um herausfinden, was dieser tut. Computer sind daran überhaupt nicht interessiert – alles beginnend mit `//` bis zum Ende der Zeile wird von einem Java-Compiler ignoriert. In der `Jabberwock`-Klasse wird der Kommentar als Platzhalter verwendet. Sie werden diesen bald ersetzen.

Die Methode `feedJabberwock()` prüft, ob ein `Jabberwock`-Objekt hungrig ist (in der Zeile `if (hungry == true)`). Wenn es hungrig ist, wird das Objekt gefüttert (zu seiner großen Freude), und der Status von `hungry` wird auf `false` gesetzt. Wenn das Objekt nicht hungrig ist, wird eine Meldung angezeigt, daß das Monster bereits gegessen hat. Das gesamte Programm sollte bis hierher wie folgt aussehen:

Listing 2.1: Der aktuelle Text von Jabberwock.java

```
1: class Jabberwock {
2:     String color;
3:     String sex;
4:     boolean hungry;
5:
```

```
 6:     void feedJabberwock() {
 7:         if (hungry == true) {
 8:             System.out.println("Yum -- a peasant!");
 9:             hungry = false;
10:         } else
11:             System.out.println("No, thanks -- already ate.");
12:     }
13:
14: // In Kürze mehr
15:}
```

Die Einrückungen und Leerzeilen, die im Quelltext für Platz sorgen, werden von einem Java-Compiler nicht beachtet. Wie Kommentare werden auch diese für den Programmierer eingefügt, damit sich die Logik eines Programms leichter nachvollziehen läßt. So wie die Einrückungen und die Abstände hier verwendet wurden (Leerzeilen zwischen Methoden und Einrükkungen bei Methoden und Variablen), werden sie im gesamten Buch verwendet. Die Java-Klassenbibliothek verwendet eine ähnliche Einrükkung. Sie können einen beliebigen Stil dabei verwenden.

Bevor Sie diese Klasse kompilieren, müssen Sie eine weitere Methode hinzufügen. Die Methode `showAttributes()` zeigt die aktuellen Werte der Instanzvariablen einer Instanz der Klasse `Jabberwock`.

Löschen Sie in dem Programm die Kommentarzeile `// In Kürze mehr`, und ersetzen Sie sie durch den folgenden Code:

```
void showAttributes() {
    System.out.printl ("This is a " + sex + " " + color + " jabberwock.");
if (hungry == true)
    System.out.println("The jabberwock is hungry.");
else
    System.out.println("The jabberwock is full.");
}
```

Die Methode `showAttributes()` gibt zwei Zeilen auf dem Bildschirm aus: in der ersten die Werte der Variablen `sex` und `color` und in der zweiten, ob das Jabberwock hungrig ist. Speichern Sie die Quelldatei in Ihrem Texteditor, und achten Sie darauf, daß die Datei auch `Jabberwock.java` heißt, damit der Dateiname mit dem Namen der Klasse übereinstimmt.

An diesem Punkt verfügen Sie über eine `Jabberwock`-Klasse mit Instanzvariablen und Instanzmethoden, die zur Anzeige und Veränderung der Werte dieser Variablen verwendet werden können.

Kompilieren Sie das Programm mit einer der beiden folgenden Methoden – abhängig davon, welches System Sie verwenden.

Windows:

Wechseln Sie von der MS-DOS-Eingabeaufforderung mit dem `CD`-Kommando in den Ordner, in dem sich Ihre Java-Quelldatei befindet. Und verwenden Sie anschließend das Kommando `javac`, um die Datei zu kompilieren:

```
javac Jabberwock.java
```

Solaris:

Wechseln Sie von der Kommandozeile aus mit dem `CD`-Kommando in das Verzeichnis, in dem sich Ihre Java-Quelldatei befindet. Und verwenden anschließend das Kommando `javac`, um die Datei zu kompilieren:

```
javac Jabberwock.java
```

Wenn bei der Kompilierung Probleme auftreten, dann prüfen Sie anhand von Listing 2.2, ob Sie eventuell Tippfehler gemacht haben.

Listing 2.2: Der aktuelle Text von Jabberwock.java

```
 1: class Jabberwock {
 2:     String color;
 3:     String sex;
 4:     boolean hungry;
 5:
 6:     void feedJabberwock() {
 7:         if (hungry == true) {
 8:             System.out.println("Yum -- a peasant!");
 9:             hungry = false;
10:         } else
11:             System.out.println("No, thanks -- already ate.");
12:     }
13:
14:     void showAttributes() {
15:         System.out.println("This is a " + sex + " " + color + " jabberwock.");
16:         if (hungry == true)
17:             System.out.println("The jabberwock is hungry.");
18:         else
19:             System.out.println("The jabberwock is full.");
20:     }
21: }
```

Das Programm ausführen

Wenn Sie die Datei `Jabberwock.java` mit einem Kommandozeilen-Tool wie dem Java-Interpreter ausführen, erhalten Sie einen Fehler:

```
In class Jabberwock: void main(String argv[]) is not definedp
```

Dieser Fehler tritt auf, da der Java-Interpreter davon ausgeht, daß das Programm eine Applikation ist, wenn Sie es von der Kommandozeile aus aufrufen. Wenn eine Applikation ausgeführt wird, ist deren `main()`-Metode der Startpunkt des Programms. Da die Klasse `Jabberwock` keine `main()`-Methode besitzt, weiß der Interpreter nicht, was er mit ihr machen soll.

Es gibt zwei Möglichkeiten, die `Jabberwock`-Klasse zu verwenden:

- Erzeugen Sie eine separates Java-Applet oder eine Java-Applikation, die diese Klasse verwendet.
- Fügen Sie eine `main()`-Methode in die `Jabberwock`-Klasse ein, so daß diese direkt ausgeführt werden kann.

In dieser Übung wollen wir letzteres tun. Laden Sie `Jabberwock.java` in den Texteditor, und fügen Sie eine Leerzeile direkt über der letzten Zeile des Programms ein (Zeile 21 in Listing 2.2).

Geben Sie hier nun folgendes ein:

```
public static void main (String arguments[]) {
    Jabberwock j = new Jabberwock();
    j.color = "orange";
    j.sex = "male";
    j.hungry = true;
    System.out.println("Calling showAttributes ...");
    j.showAttributes();
    System.out.println("-----");
    System.out.println("Feeding the jabberwock ...");
    j.feedJabberwock();
    System.out.println("-----");
    System.out.println("Calling showAttributes ...");
    j.showAttributes();
    System.out.println("-----");
    System.out.println("Feeding the jabberwock ...");
    j.feedJabberwock();
}
```

Mit der `main()`-Methode kann die `Jabberwock`-Klasse nun als Applikation verwendet werden. Speichern Sie die Datei, und kompilieren Sie sie anschließend.

Das Listing 2.3 zeigt die endgültige Version der Datei `Jabberwock.java`, für den Fall, daß Sie bei der Kompilierung Probleme haben.

Sie finden auf der CD eine Kopie der Quelldateien und anderer benötigter Dateien. Wenn Sie Probleme mit Programmen in diesem Buch haben, können Sie anhand dieser Dateien prüfen, wo das Problem liegt.

Listing 2.3: Die endgültige Version von Jabberwock.java

```
 1: class Jabberwock {
 2:     String color;
 3:     String sex;
 4:     boolean hungry;
 5:
 6:     void feedJabberwock() {
 7:         if (hungry == true) {
 8:             System.out.println("Yum -- a peasant!");
 9:             hungry = false;
10:         } else
11:             System.out.println("No, thanks -- already ate.");
12:     }
13:
14:     void showAttributes() {
15:         System.out.println("This is a " + sex + " " + color + " jabberwock.");
16:         if (hungry == true)
17:             System.out.println("The jabberwock is hungry.");
18:         else
19:             System.out.println("The jabberwock is full.");
20:     }
21:
22:     public static void main (String arguments[]) {
23:         Jabberwock j = new Jabberwock();
24:         j.color = "orange";
25:         j.sex = "male";
26:         j.hungry = true;
27:         System.out.println("Calling showAttributes ...");
28:         j.showAttributes();
29:         System.out.println("-----");
30:         System.out.println("Feeding the jabberwock ...");
31:         j.feedJabberwock();
32:         System.out.println("-----");
33:         System.out.println("Calling showAttributes ...");
34:         j.showAttributes();
35:         System.out.println("-----");
36:         System.out.println("Feeding the jabberwock ...");
37:         j.feedJabberwock();
38:     }
39: }
```

Anhand von Listing 2.3 wollen wir im folgenden ansehen, was in der `main()`-Methode passiert:

- Zeile 22: Die `main()`-Methode wird deklariert. Die erste Zeile der `main()`-Methode sieht immer so aus. Die einzelnen Elemente dieses Ausdrucks lernen Sie später in dieser Woche kennen.
- Zeile 23: `Jabberwock j = new Jabberwock();`, erzeugt eine neue Instanz der `Jabberwock`-Klasse und speichert eine Referenz zu dieser in einer neuen Variablen mit dem Namen `j`. Wie Sie bereits gelernt haben, arbeiten Sie in Java-Programmen normalerweise nicht direkt mit Klassen. Statt dessen erzeugen Sie Objekte dieser Klassen und rufen Methoden dieser Objekte auf, um mit den Objekten zu arbeiten.
- Die Zeilen 24–26: Den Instanzvariablen `color`, `sex` und `hungry` des `Jabberwock`-Objektes, das in Zeile 23 erzeugt wurde, werden Werte zugewiesen. `color` erhält den Wert »`orange`«, `sex` den Wert »`male`« und die Variable `hungry` den booleschen Wert `true`. Dies zeigt an, daß dieses `Jabberwock`-Objekt hungrig ist.
- Zeile 27: In dieser Zeile und einigen weiteren, die folgen, wird die Anweisung `System.out.println()` verwendet, um Informationen auf dem Bildschirm auszugeben. Alles, was sich dabei zwischen den Klammern befindet, wird angezeigt.
- Zeile 28: Die Methode `showAttributes()`, die in dem `Jabberwock`-Objekt definiert ist, wird hier aufgerufen. Die fordert das `Jabberwock`-Objekt dazu auf, die Werte seiner Variablen `color`, `sex` und `hungry` auszugeben.
- Zeile 31: Hier wird die Methode `feedJabberwock()` des `Jabberwock`-Objekts aufgerufen, die den Wert der Variable `hungry` von `true` auf `false` setzt und eine Bemerkung voll des Dankes von dem `Jabberwock`-Objekt anzeigt: »`Yum -- a peasant!`« (zu deutsch »Lecker -- ein Bauer!«)
- Zeile 33: Die Methode `showAttributes()` wird hier erneut aufgerufen, um die Werte der Instanzvariablen des `Jabberwock`-Objekts auszugeben. Diesmal sollte die Meldung aussagen, daß das Jabberwock voll ist (`hungry` hat den Wert `false`).
- Zeile 36: Die Methode `feedJabberwock()` wird erneut aufgerufen, um einen Versuch zu starten, das Jabberwock zu füttern. Da das Jabberwock aber bereits voll ist, lehnt es mit einem höflichen »`No thanks -- already ate.`« (zu deutsch »Nein danke -- habe schon gegessen.«) die Nahrung ab.

Die `Jabberwock`-Applikation können Sie mit einer der folgenden plattformspezifischen Prozeduren ausführen:

Windows:

Wechseln Sie von der MS-DOS-Eingabeaufforderung aus mit dem `CD`-Kommando in das Verzeichnis, das die Datei `Jabberwock.class` enthält, und starten Sie den Interpreter mit dem Befehl `java`:

```
java Jabberwock
```

Solaris:

Wechseln Sie von der Kommandozeile aus mit dem `CD`-Kommando in das Verzeichnis, das die Datei `Jabberwock.class` enthält, und starten Sie den Interpreter mit dem Befehl `java`:

```
java Jabberwock
```

Wenn Sie die `Jabberwock`-Klasse ausführen, sollten Sie die folgende Ausgabe auf dem Bildschirm erhalten:

```
Calling showAttributes ...
This is a male orange jabberwock.
The jabberwock is hungry.
-----
Feeding the jabberwock ...
Yum -- a peasant!
-----
Calling showAttributes ...
This is a male orange jabberwock.
The jabberwock is full.
-----
Feeding the jabberwock ...
No, thanks -- already ate.
```

Ich gehe hier davon aus, daß Sie wissen, wie eine Java-Applikation kompiliert und ausgeführt wird. Eine umfangreichere Anleitung finden Sie am Tag 1 und in der Dokumentation Ihres Entwicklungstools.

Klassen und deren Verhalten organisieren

Eine Einführung in die objektorientierte Programmierung mit Java ist nicht komplett ohne eine erste Betrachtung der folgenden drei Konzepte: Vererbung, Schnittstellen und Pakete.

Diese drei Konzepte stellen allesamt Mechanismen zur Organisation von Klassen und dem Verhalten von Klassen dar. Die Klassenbibliothek von Java verwendet diese Konzepte, und auch die Klassen, die sie für Ihre eigenen Programme erstellen, werden diese benötigen.

Vererbung

Die Vererbung ist eines der entscheidenden Konzepte der objektorientierten Programmierung und beeinflußt direkt die Art und Weise, wie Sie Ihre eigenen Java-Klassen schreiben.

Vererbung ist ein Mechanismus, der es einer Klasse ermöglicht, all Ihre Verhaltensweisen und Attribute von einer anderen Klasse zu erben.

Über die Vererbung verfügt eine Klasse sofort über die gesamte Funktionalität einer vorhandenen Klasse. Aus diesem Grund kann man eine neue Klasse erstellen, indem man angibt, wie sie sich von einer bestehenden unterscheidet.

Durch die Vererbung werden alle Klassen Teil einer strengen Hierarchie – Klassen, die Sie erzeugen, Klassen aus der Klassenbibliothek von Java und anderen Bibliotheken.

Eine Klasse, von der andere Klassen abgeleitet werden (sprich, die ihre Funktionalität an andere Klassen vererbt), wird *Superklasse* genannt. Die Klasse, die die Funktionalität erbt, wird als *Subklasse* bezeichnet.

Eine Klasse kann lediglich eine Superklasse besitzen. Jede Klasse kann allerdings eine uneingeschränkte Anzahl von Subklassen haben. Subklassen erben alle Attribute und sämtliche Verhaltensweisen ihrer Superklasse.

In der Praxis bedeutet dies, daß, wenn eine Superklasse Verhaltensweisen und Attribute besitzt, die Ihre Klasse benötigt, Sie diese nicht neu definieren oder den Code kopieren müssen, um über dieselbe Funktionalität in Ihrer Klasse zu verfügen. Ihre Klasse erhält all dies automatisch von ihrer Superklasse, die Superklasse erhält das Ganze wiederum von ihrer Superklasse und so weiter, der Hierarchie folgend. Ihre Klasse wird eine Kombination aller Features der Klassen über ihr in der Hierarchie und ihrer eigenen.

Diese Situation läßt sich sehr gut damit vergleichen, wie Sie bestimmte Eigenschaften, wie z.B. Größe, Haarfarbe, die Leidenschaft für Ska-Musik und den Widerwillen, nach dem Weg zu fragen, von Ihren Eltern geerbt haben. Diese haben einige dieser Dinge wiederum von ihren Eltern geerbt, die von den ihren und so weiter bis zurück in den Garten Eden, zum Urknall oder was auch immer am Anfang war.

Die Abbildung 2.2 zeigt, wie eine Hierarchie von Klassen organisiert ist.

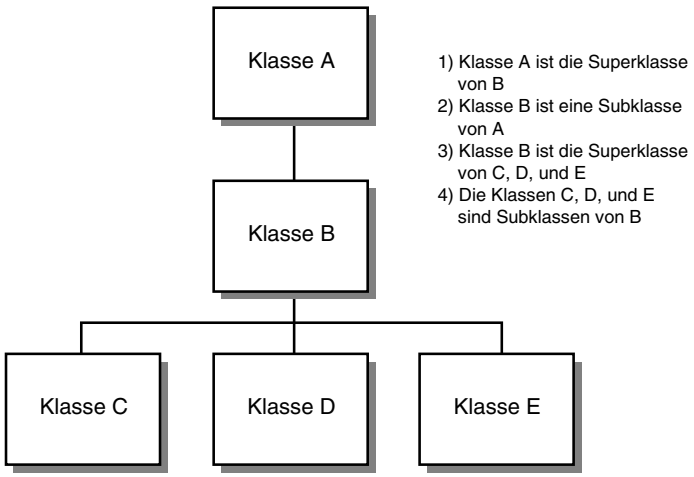

Abbildung 2.2:
Eine Klassenhierarchie

An der Spitze der Hierarchie der Java-Klassen steht die Klasse `Object` – alle Klassen werden von dieser Superklasse abgeleitet. `Object` stellt die allgemeinste Klasse in der Hierarchie dar und legt die Verhaltensweisen und Attribute, die an alle Klassen in der Java-Klassenbiliothek vererbt werden, fest. Jede Klasse, die sich weiter unten in der Hierarchie befindet, ist stärker auf einen bestimmten Zweck zugeschnitten. Eine Klassenhierarchie definiert abstrakte Konzepte an der Spitze der Hierarchie. Diese Konzepte werden, je weiter Sie in der Hierarchie hinabsteigen, immer konkreter.

Oft werden Sie, wenn Sie in Java eine neue Klasse erstellen, die gesamte Funktionalität einer existierenden Klasse mit einigen eigenen Modifikationen haben wollen. Es könnte z.B. sein, daß Sie eine Version von `CommandButton`-Schaltflächen wollen, die beim Anklicken ein ohrenbetäubendes Explosionsgräusch erzeugen. (Weder die Autoren noch der Verlag halten dies für eine gute Idee, noch können sie für etwaige Schäden beim Anwender haftbar gemacht werden.)

Um die gesamte Funktionalität von `CommandButton` ohne den Aufwand, sie neu erstellen zu müssen, zu erhalten, können Sie eine Klasse als Subklasse von `CommandButton` definieren. Ihre Klasse verfügt dann automatisch über die Attribute und Verhaltensweisen, die in `CommandButton`, und über die, die in den Superklassen von `CommandButton` definiert wurden. Sie müssen sich jetzt nur noch um das kümmern, was Ihre neue Klasse von `CommandButton` unterscheidet. Der Mechanismus zur Definition neuer Klassen über die Unterschiede zwischen diesen und deren Superklasse wird im Englischen als *Subclassing* bezeichnet.

Sollte Ihre Klasse ein komplett neues Verhalten definieren und keine Subklasse einer bestehenden Klasse sein, können Sie diese direkt von der Klasse `Object` ableiten.

Dies erlaubt es ihr, sich nahtlos in die Klassenhierarchie von Java zu integrieren. Wenn Sie eine Klassendefinition erstellen, die keine Superklasse angibt, nimmt Java an, daß die neue Klasse direkt von `Object` abgeleitet wird. Die Klasse `Jabberwock`, die Sie weiter oben erstellt haben, ist direkt von `Object` abgeleitet.

Eine Klassenhierarchie erzeugen

Wenn Sie eine große Menge von Klassen erzeugen, ist es sinnvoll, diese zum einen von den bestehenden Klassen der Klassenhierarchie abzuleiten, und zum anderen, daß diese eine eigene Hierarchie bilden. Ihre Klassen auf diese Art und Weise zu organisieren bedarf einer umfangreichen Planung. Als Entschädigung erhalten Sie allerdings unter anderem die folgenden Vorteile:

- Funktionalitäten, die mehrere Klassen gemein haben, können in einer Superklasse zusammengefaßt werden. Dadurch wird es möglich, diese Funktionalitäten in allen Klassen, die sich in der Hierarchie unterhalb dieser Superklasse befinden, wiederholt zu verwenden.
- Änderungen in einer Superklasse werden automatisch in all ihren Subklassen, deren Subklassen usw. widergespiegelt. Es ist nicht nötig, die Klassen, die sich in der Hierarchie darunter befinden, zu ändern oder neu zu kompilieren, da diese die neuen Informationen über die Vererbung erhalten.

Stellen Sie sich einmal vor, Sie hätten eine Klasse erstellt, die sämtliche Features eines Jabberwock implementiert. (Dies sollte nicht allzuschwer sein, wenn Sie den entsprechenden Teil des heutigen Tages nicht übersprungen haben.)

Die Klasse `Jabberwock` ist vollständig, arbeitet erfolgreich, und alles ist fein. Nun wollen Sie eine Klasse erstellen, die `Dragon` (engl. für Drachen) heißt. Drachen und Jabberwocks besitzen einige ähnliche Features – beides sind sehr große Monster und fressen Bauern. Beide haben Krallen, kräftige Zähne und eine ausgeprägte Persönlichkeit. Ihre erste Idee ist nun vielleicht, die Quelldatei `Jabberwock.java` zu öffnen und einen großen Teil des Quellcodes in eine neue Quelldatei mit dem Namen `Dragon.java` zu kopieren.

Eine wesentlich bessere Idee wäre es allerdings, die gemeinsame Funktionalität von `Dragon` und `Jabberwock` zu ermitteln und sie in einer etwas allgemeineren Klassenhierarchie zu organisieren. Dies wäre für die Klassen `Dragon` und `Jabberwock` eventuell etwas viel Aufwand. Was aber, wenn Sie noch die Klassen `Medusa`, `Yeti`, `Sasquatch`, `Grue` und `DustBunny` hinzufügen wollen? Allgemeine Verhaltensweisen und Attribute in eine oder mehrere wiederverwendbare Superklassen einzufügen, reduziert den Gesamtaufwand ganz beträchtlich.

Um eine Klassenhierarchie zu entwerfen, die diese Aufgabe erfüllt, beginnen wir an der Spitze mit der Klasse `Object`, dem Ausgangspunkt aller Klassen unter Java. Die

allgemeinste Klasse, der sowohl ein Jabberwock als auch ein Drachen angehören, könnte `Monster` genannt werden. Ein Monster könnte allgemein als grausame Kreatur gelten, die Menschen terrorisiert und die Werte von Eigenschaften verringert. In der `Monster`-Klasse definieren Sie nur das Verhalten, das etwas als grausam, terrorisierend und schlecht für die Nachbarschaft beschreibt.

Unterhalb von `Monster` könnte es zwei Klassen geben: `FlyingMonster` (fliegendes Monster) und `WalkingMonster` (gehendes Monster). Der offensichtlichste Unterschied zwischen diesen beiden Klassen ist, daß Monster der einen Klasse fliegen können und die Monster der anderen Klasse nicht. Die Verhaltensweisen der fliegenden Monster könnten die folgenden einschließen: auf die Beute stürzen, Bauern ergreifen und mit in die Lüfte tragen, diese aus großen Höhen abstürzen lassen usw. Gehende Monster würden sich hingegen ganz anders verhalten. Abbildung 2.3 zeigt, was wir bis jetzt haben.

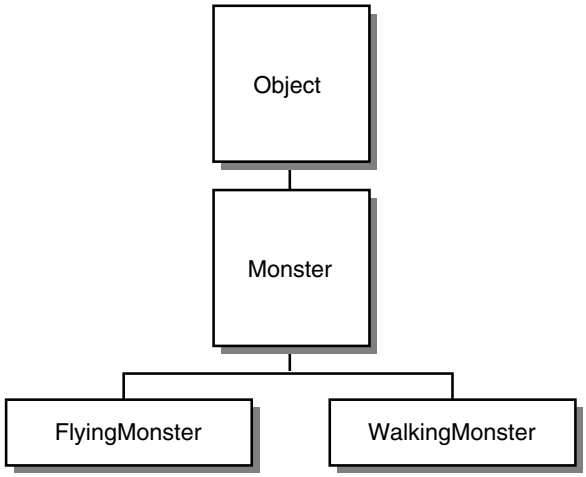

Abbildung 2.3:
Die grundlegende Monster-Hierarchie

Davon ausgehend, kann die Hierarchie noch spezieller werden. Von `FlyingMonster` könnten Sie einige Klasse ableiten: `Mamal`, `Reptile`, `Amphibian` usw. Alternativ könnten Sie auch weitere Funktionalitäten ausmachen und in den Zwischenklassen `TwoLegged` und `FourLegged` für zwei- und vierbeinige Monster mit jeweils unterschiedlichen Verhaltensweisen zusammenfassen (siehe auch Abbildung 2.4).

Zu guter Letzt steht die Hierarchie, und Sie haben einen Platz für `Jabberwock`. Dies wäre eine Subklasse von `Reptile`, `Fourlegged`, `FlyingMonster`, `Monster` und `Object`, da `FlyingMonster` eine Subklasse von `Monster` und `Monster` eine Subklasse von `Object` ist.

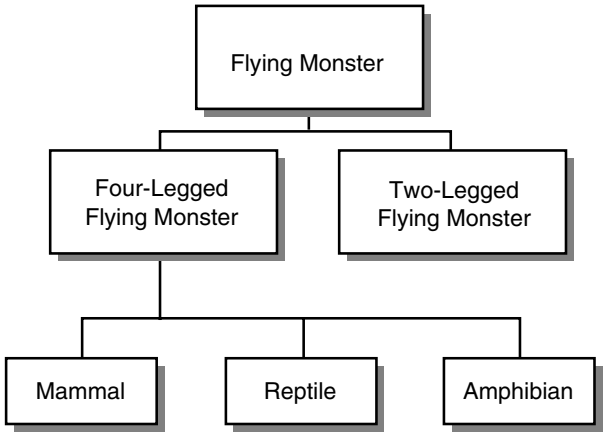

Abbildung 2.4: Zwei- und vierbeinige fliegende Monster

Wo kommen nun Eigenschaften, wie Geschlecht, Farbe oder Appetit ins Spiel? Dort, wo Sie sich in die Klassenhierarchie am harmonischsten einfügen. Sie können `sex` und `color` als Instanzvariablen in Monster definieren, und alle Subklassen werden ebenfalls über diese Variablen verfügen. Denken Sie daran, daß Sie eine Verhaltensweise oder ein Attribut nur einmal in der Hierarchie definieren müssen und alle Subklassen es dann automatisch erben.

Der Entwurf einer effektiven Klassenhierarchie schließt eine umfangreiche Planung und Überarbeitung ein. Während Sie versuchen, Attribute und Verhaltensweisen in einer Hierarchie anzuordnen, werden Sie wahrscheinlich oftmals erkennen, daß es sinnvoll ist, Klassen innerhalb der Hierarchie an andere Stellen zu verschieben. Das Ziel ist es, die Anzahl sich wiederholender Features auf das Nötigste zu reduzieren. Wenn Sie eine Hierarchie von Monstern entwickeln, werden Sie eventuell `Mammal`, `Reptile` und `Amphibian` direkt unterhalb von `Monster` anordnen wollen, falls dies die Funktionalität, die Ihre Klassen beinhalten, besser beschreiben sollte.

Vererbung in Aktion

Die Vererbung funktioniert in Java wesentlich einfacher als in der realen Welt. Hier sind weder Testamentsvollstrecker noch Richter oder Gerichte irgendeiner Art nötig.

Wenn Sie ein neues Objekt erzeugen, ermittelt Java jede Variable, die für dieses Objekt definiert wurde, und jede Variable, die in jeder der Superklassen des Objekts definiert wurde. Auf diesem Weg werden alle dieser Klassen kombiniert, um eine Vorlage für das aktuelle Objekt zu formen. Jedes Objekt füllt die einzelnen Variablen dann mit den Informationen, die seiner Situation entsprechen.

Methoden arbeiten auf ganz ähnliche Art: Neue Objekte haben Zugriff auf alle Methodennamen der eigenen Klasse und deren Superklassen in der Hierarchie. Dies wird allerdings dynamisch festgelegt, wenn eine Methode in einem laufenden Programm ausgeführt wird. Wenn Sie die Methode eines bestimmten Objekts aufrufen, dann prüft der Java-Interpreter zunächst die Klasse des Objekts, ob sich die Methode hier befindet. Wenn er die Methode nicht findet, dann sucht er nach dieser in der Superklasse. Das geht so lange weiter, bis er die Definition der Methode gefunden hat. Die Abbildung 2.5 illustriert dies.

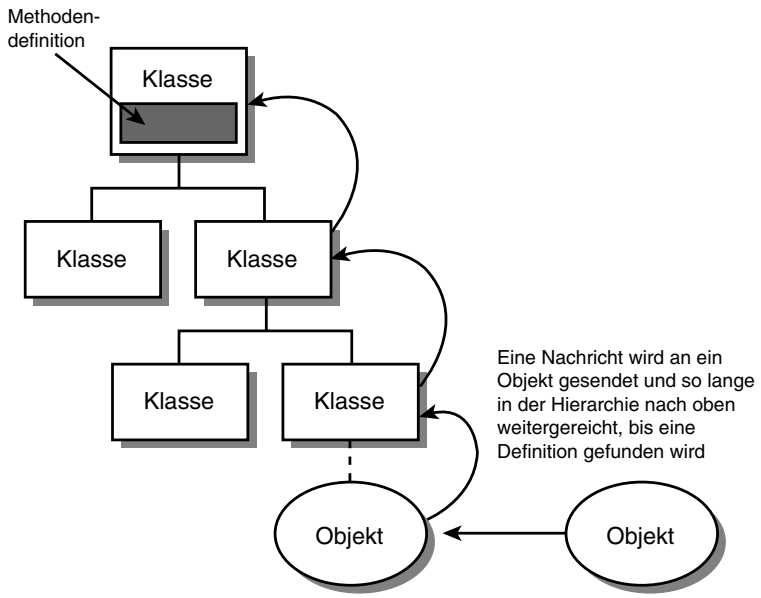

Abbildung 2.5: So werden Methoden in der Klassenhierarchie geortet

Die Dinge werden komplizierter, wenn eine Subklasse eine Methode definiert, die denselben Namen, denselben Rückgabetyp und dieselben Parameter hat wie eine Methode in einer Superklasse. In diesem Fall wird die Methodendefinition, die als erstes gefunden wird (ausgehend vom unteren Ende der Hierarchie nach oben), verwendet. Aus diesem Grund können Sie in einer Subklasse eine Methode erstellen, die verhindert, daß eine Methode in einer Superklasse ausgeführt wird. Dazu geben Sie einer Methode denselben Namen, denselben Rückgabetyp und dieselben Argumente, wie sie die Methode in der Superklasse hat. Dieses Vorgehen wird als Überschreiben bezeichnet (siehe Abbildung 2.6).

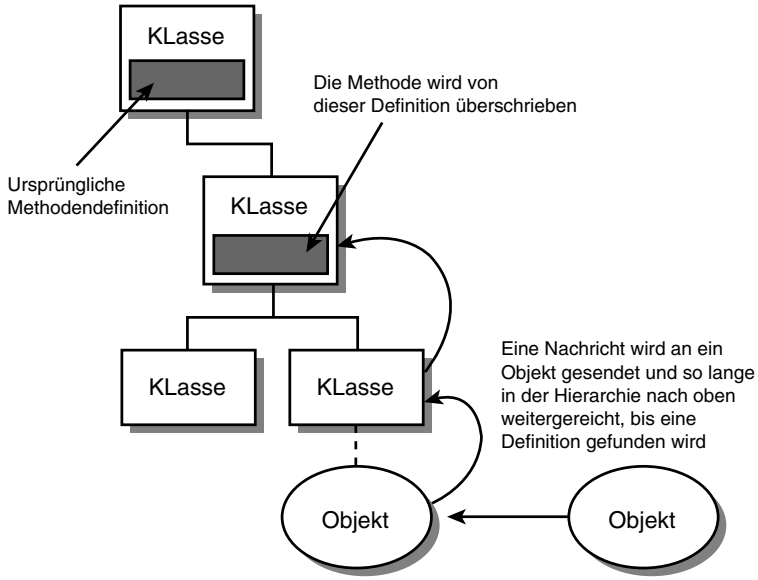

Abbildung 2.6: Methoden überschreiben

Einfach- und Mehrfachvererbung

Javas Form der Vererbung wird Einfachvererbung genannt, da jede Java-Klasse nur eine Superklasse haben, kann (allerdings kann jede Superklasse beliebig viele Subklassen haben).

In anderen objektorientierten Programmiersprachen, wie z.B. C++, können Klassen mehr als eine Superklasse haben, und sie erben natürlich die Variablen und Methoden aus allen Superklassen. Dies wird als Mehrfachvererbung bezeichnet und bietet die Möglichkeit, Klassen zu erzeugen, die nahezu alle denkbaren Verhaltensweisen und Attribute beinhalten. Allerdings werden dadurch die Definition von Klassen und der Code, der für deren Erstellung benötigt wird, bedeutend komplizierter. Java vereinfacht die Vererbung, indem es nur die Einfachvererbung zuläßt.

Schnittstellen

Durch die Einfachvererbung ist die Beziehung zwischen Klassen und die Funktionalität, die diese Klassen implementieren, einfacher zu verstehen und zu entwerfen. Allerdings kann dies auch einschränkend sein – besonders dann, wenn Sie ähnliche Verhaltensweisen in verschiedenen Zweigen der Klassenhierarchie duplizieren müssen. Java löst das Problem von gemeinsam genutzten Verhaltensweisen durch Schnittstellen.

Eine *Schnittstelle* (engl. Interface) ist eine Sammlung von Methoden, die benannt aber nicht implementiert sind. Dadurch wird angezeigt, daß eine Klasse neben dem aus der Superklasse geerbten Verhalten noch zusätzliche Verhaltensweisen hat.

Eine Klasse kann beliebig viele Schnittstellen implementieren. Durch die Implementierung einer Schnittstelle wird die Klasse gezwungen die Methoden zu implementieren, deren Namen von der Schnittstelle definiert wurden. Wenn zwei sehr unterschiedliche Klassen dieselbe Schnittstelle implementieren, können beide auf Aufrufe der Methoden, die in der Schnittstelle definiert sind, reagieren. Allerdings kann die Reaktion auf diese Methodenaufrufe bei den einzelnen Klassen total unterschiedlich sein.

Pakete

Pakete (engl. Packages) sind eine Möglichkeit, um verwandte Klassen und Schnittstellen zu gruppieren. Pakete ermöglichen es, daß Gruppen von Klassen nur bei Bedarf verfügbar sind. Zusätzlich beseitigen sie mögliche Namenskonflikte zwischen Klassen in unterschiedlichen Gruppen von Klassen.

Zum jetzigen Zeitpunkt gibt es nur ein paar wenige Dinge, die Sie über Pakete wissen müssen:

- Die Klassenbibliotheken von Java befinden sich in einem Paket, das `java` heißt. Für die Klassen in dem Paket `java` wird garantiert, daß sie in jeder Implementierung von Java zur Verfügung stehen. Dies sind die einzigen Klassen, für die die Garantie besteht, daß sie in unterschiedlichen Implementierungen zur Verfügung stehen. Das Paket `java` beinhaltet kleinere Pakete, die spezielle Teile der Funktionalität der Sprache Java, wie z.B. Standard-Features, Dateiein- und ausgaben, Multimedia und viele andere Dinge, definieren. Klassen in anderen Paketen, wie z.B. `sun` oder `netscape`, stehen oft nur in bestimmten Implementierungen zur Verfügung.
- Standardmäßig haben Ihre Klassen nur Zugriff auf die Klassen in dem Paket `java.lang` (Standard-Features der Sprache). Um Klassen aus irgendeinem anderen Paket zu verwenden, müssen Sie sich direkt auf diese beziehen oder sie in Ihren Quelltext importieren.
- Um sich auf eine Klasse in einem Paket zu beziehen, müssen Sie alle Pakete, in denen sich die Klasse befindet, angeben. Die einzelnen Elemente müssen dabei durch Punkte (.) voneinander getrennt werden. Nehmen Sie als Beispiel die Klasse `Color`, die sich in dem Paket `awt` befindet. Dieses Paket befindet sich seinerseits im Paket `java`. Um sich nun auf die Klasse `Color` in Ihren Programmen zu beziehen, können Sie die folgende Notation verwenden: `java.awt.Color`.

Eine Subklasse erstellen

Als abschließendes Projekt für heute werden Sie eine Subklasse einer anderen Klasse erstellen und einige Methoden überschreiben. Sie werden auch eine besseres Verständnis dafür bekommen, wie Pakete funktionieren.

Wenn Sie mit der Programmierung in Java beginnen, werden Sie Klassen zur Erstellung von Applets ableiten. Die Erstellung von Applets unterscheidet sich von der von Applikationen. Java-Applets werden als Teil einer Webseite ausgeführt. Aus diesem Grund gibt es besondere Regeln für deren Verhalten. Wegen dieser speziellen Regeln für Applets ist die Erstellung eines einfachen Applets komplizierter als die einer einfachen Applikation.

Alle Applets sind Subklassen der Klasse `Applet` (diese ist Bestandteil des Paketes `java.applet`). Indem Sie die Klasse `Applet` ableiten, erhalten Sie automatisch alle Verhaltensweisen und Attribute, die es einem Programm ermöglichen, als Teil einer Webseite zu laufen.

In diesem Beispiel erstellen Sie ein Applet, das der Applikation `HelloDan` von gestern ähnelt. Zu Beginn dieses Beispiels erzeugen Sie die Klassendefinition. Starten Sie Ihren Texteditor, und geben Sie die folgenden Anweisungen ein:

```
public class Palindrome extends java.applet.Applet {
    // mehr in Kürze
}
```

Dies definiert eine Klasse namens `Palindrome`. Die Anweisungen ähneln der Art und Weise, wie Sie die `HelloDan`-Applikation am ersten Tag erstellt haben. Eine Neuerung stellt der Text `extends java.applet.Applet` dar.

Über die Anweisung `extends` wird festgelegt, daß eine Klasse eine Subklasse einer anderen ist. Die Klasse `Palindrome` ist demnach eine Subklasse der Klasse `Applet`, die Teil des Paketes `java.applet` ist. Um in einem Programm die Beziehung zwischen den zwei Klassen deutlich zu machen, wird die Anweisung `extends java.applet.Applet` verwendet.

Da sich die Klasse `Applet` in dem Paket `java.applet` befindet, haben Sie nicht automatisch Zugriff auf diese Klasse. Aus diesem Grund müssen Sie sich explizit auf diese über den Paket- und Klassennamen beziehen. Die einzigen Klassen, die Sie ohne Angabe der Paketzugehörigkeit verwenden können, sind die in dem Paket `java.lang`.

Ein weiteres neues Element im `class`-Statement ist das Schlüsselwort `public`. Dieses Schlüsselwort zeigt an, daß andere Klassen auf Ihre Klasse zugreifen können, wenn sie diese benötigen. Im Normalfall müssen Sie eine Klasse nur dann als `public` deklarieren, wenn Sie wollen, daß sie von anderen Klassen in Ihrem Java-Programm verwendet werden kann. Alle Applets müssen allerdings `public` sein.

Eine Klassendefinition mit nichts außer dem Kommentar `// mehr in Kürze` darin ergibt nicht viel Sinn – sie fügt ihrer Superklasse weder etwas hinzu, noch überschreibt sie Methoden oder Variablen der Superklasse. Um die `Palindrome`-Klasse gegenüber ihrer Superklasse zu verändern, löschen Sie die Kommentarzeile `// mehr in Kürze` und fügen neue Anweisungen, beginnend mit der folgenden, hinzu:

```
Font f = new Font("TimesRoman", Font.BOLD, 36);
```

Diese Anweisung erfüllt zwei Aufgaben:

- Es wird ein `Font`-Objekt mit dem Namen `f` erzeugt. Font als Teil des `java.awt`-Paketes wird verwendet, um Bildschirmschriften zu repräsentieren. Mit einem solchen Objekt kann man eine Schrift anzeigen, die sich von der standardmäßig in Applets benutzten Schrift unterscheidet.
- Dem `Font`-Objekt wird die Schrift Times Roman, fett in 36-Punkt zugewiesen. Die `new`-Anweisung erzeugt das neue `Font`-Objekt mit den in den Klammern angegebenen Werten. Das neu erzeugte Objekt wird anschließend der Variablen `f` zugewiesen.

Indem Sie eine Instanzvariable erstellen, die dieses `Font`-Objekt aufnimmt, machen Sie es für alle Methoden in Ihrer Klasse verfügbar. Der nächste Schritt in dem `Palindrome`-Projekt ist die Erstellung von Methoden, die dieses Objekt nutzen.

Wenn Sie Applets schreiben, werden Sie gewöhnlich einige Methoden, die in der Superklasse `Applet` definiert sind, in Ihrem Applet überschreiben. Diese beinhalten Methoden, um das Applet vor der eigentlichen Ausführung zu initialisieren, das Applet zu starten, auf Mauseingaben zu reagieren und aufzuräumen, wenn das Applet beendet wird.

Eine dieser Methoden ist `paint()`, die sich um die Anzeige des Applets auf einer Webseite kümmert. Die `paint()`-Methode, die `Palindrome` erbt, tut überhaupt nichts – sie ist einfach eine leere Methode. Indem Sie `paint()` überschreiben, legen Sie fest, was im Applet-Fenster, während das Programm läuft, bei Bedarf ausgegeben werden soll. Fügen Sie eine leere Zeile unter der `Font`-Anweisung ein, und geben Sie den folgenden Code ein:

```
public void paint(Graphics screen) {
    screen.setFont(f);
    screen.setColor(Color.red);
    screen.drawString("Go hang a salami, I'm a lasagna hog.", 5, 40);
}
```

Die `paint()`-Methode ist wie das Applet selbst als `public` deklariert. Hier allerdings aus einem anderen Grund: `paint()` muß in diesem Fall `public` sein, da die Methode, die es hier überschreibt, auch `public` ist. Eine `public`-Methode einer Superklasse kann nur von einer `public`-Methode überschrieben werden. Andernfalls wird das Java-Programm nicht kompiliert.

Die `paint()`-Methode besitzt ein einziges Argument: eine Instanz der Klasse `Graphics` mit dem Namen `screen`. Die Klasse `Graphics` stellt die Verhaltensweisen zur Darstellung von Schriften, Farben, zum Zeichnen von Linien und anderen Formen zur Verfügung. Sie werden während der zweiten Woche, wenn Sie diverse weitere Applets erstellen, mehr über die Klasse `Graphics` lernen.

In der `paint()`-Methode haben Sie drei Dinge getan:

- Sie haben dem `Graphics`-Objekt mitgeteilt, daß sich die Schrift zur Anzeige von Text in der Instanzvariablen `f` befindet.
- Sie haben dem `Graphics`-Objekt mitgeteilt, daß die Farbe für Text-Ausgaben und andere Zeichenoperationen eine Instanz der Klasse `Color` für die Farbe rot ist.
- Schließlich haben Sie den Text »`Go hang a salami, I'm a lasagna hog`« auf dem Bildschirm bei den (x, y)-Koordinaten `5, 25` ausgegeben. Der String wird in der festgelegten Schrift und Farbe angezeigt.

Das Applet sieht bis jetzt folgendermaßen aus:

```
public class Palindrome extends java.applet.Applet {
    Font f = new Font("TimesRoman", Font.BOLD, 36);

    public void paint(Graphics screen) {
        screen.setFont(f);
        screen.setColor(Color.red);
        screen.drawString("Go hang a salami, I'm a lasagna hog.", 5, 40);
    }
}
```

Sie haben vielleicht bemerkt, daß an diesem Punkt des Beispiels noch etwas fehlt. Wenn Sie nämlich die Datei gespeichert und versucht haben, diese zu kompilieren, dann werden Sie eine Reihe von Fehlermeldungen wie die folgende erhalten haben:

```
Palindrome.java:2: Class Font not found in type declaration.p
```

Diese treten auf, da die Klassen `Graphics`, `Font` und `Color` Teil des `java.awt`-Paketes sind. Und dieses Paket ist nicht standardmäßig verfügbar. Auf die `Applet`-Klasse haben Sie sich direkt in der ersten Zeile der Klassendefinition bezogen, indem Sie deren vollen Paketnamen (`java.applet.Applet`) angegeben haben. Im übrigen Programm haben Sie Klassen ohne deren Paketnamen verwendet.

Es gibt zwei Möglichkeiten, dieses Problem zu lösen:

- Beziehen Sie sich auf alle externen Klassen über deren vollen Paketnamen, wie z.B. `java.awt.Graphics`, `java.awt.Font` und `java.awt.Color`.
- Verwenden Sie eine `import`-Anweisung am Anfang des Programms, um ein oder mehrere Pakete und Klassen in dem Programm verfügbar zu machen.

Welche der beiden Lösungen Sie verwenden, bleibt Ihrer Wahl überlassen. Wenn Sie sich allerdings häufig auf eine Klasse in einem anderen Paket beziehen, dann werden Sie wahrscheinlich die `import`-Anweisung verwenden wollen, um den Tippaufwand zu reduzieren.

In diesem Beispiel werden wir die letztere Variante verwenden.

Um die Klassen zu importieren, fügen Sie die folgenden drei Anweisungen über der Anweisung `public class Palindrome` ein:

```
import java.awt.Graphics;
import java.awt.Font;
import java.awt.Color;
```

Sie können auch ein komplettes Paket importieren, indem Sie ein Sternchen (*) anstelle eines konkreten Klassennamens verwenden. Um z.B. alle Klassen des `java.awt`-Pakets zu importieren, verwenden Sie das folgende Statement:

```
import java.awt.*;
```

Nun, da die richtigen Klassen in Ihr Programm importiert wurden, sollte sich `Palindrome.java` problemlos in eine `.class`-Datei kompilieren lassen. In Listing 2.4 finden Sie zu Vergleichszwecken die endgültige Version.

Listing 2.4: Die endgültige Version von Palindrome.java

```
 1: import java.awt.Graphics;
 2: import java.awt.Font;
 3: import java.awt.Color;
 4:
 5: public class Palindrome extends java.applet.Applet {
 6:     Font f = new Font("TimesRoman", Font.BOLD, 36);
 7:
 8:     public void paint(Graphics screen) {
 9:         screen.setFont(f);
10:         screen.setColor(Color.red);
11:         screen.drawString("Go hang a salami, I'm a lasagna hog.", 5, 40);
12:     }
13: }
```

Speichern Sie diese Datei als `Palindrome.java`. Diese Quelldatei kann auf dieselbe Art und Weise kompiliert werden wie die Java-Applikationen, die Sie bisher erstellt haben. Um das Programm auszuführen, müssen Sie allerdings eine Webseite erzeugen und es darauf einfügen.

Viele Programme zur Entwicklung von Webseiten, wie z.B. Claris Home Page, Macromedia Dreamweaver oder Microsoft FrontPage, ermöglichen es, ein Java-Applet auf einer Webseite einzufügen.

Wenn Sie keines dieser Tools zur Verfügung haben, können Sie eine einfache Webseite mit den entsprechenden Features zur Integration von Java-Applets direkt über die Seitenbeschreibungssprache HTML erstellen.

Obwohl einige der HTML-Features, die in Bezug zu Java stehen, in diesem Buch beschrieben werden, ist die Entwicklung von Webseiten mit HTML außerhalb des Rahmens dieses Buches. Wenn Sie an HTML interessiert sind, dann finden Sie in dem Buch *HTML 4 in 14 Tagen* von Laura Lemay (ISBN 3-8272-2019-X), erschienen bei Sams, eine umfangreiche und leichtverständliche Anleitung.

Um eine HTML-Seite zu erstellen, die das `Palindrome`-Applet aufnehmen kann, öffnen Sie denselben Texteditor, den Sie auch zur Erstellung Ihrer Java-Programme verwenden, und erzeugen ein neues Dokument.

Geben Sie den Text aus Listing 2.5 ein, und speichern Sie die Datei als `Palindrome.html` in demselben Ordner, in dem sich auch die Dateien `Palindrome.java` und `Palindrome.class` befinden. Wenn Sie Windows 95 verwenden, schließen Sie den Namen beim Abspeichern in Anführungszeichen ein, um sicherzugehen, daß die Erweiterung `.txt` nicht hinzugefügt wird.

Listing 2.5: Die Webseite Palindrome.html

```
1: <APPLET CODE="Palindrome.class" WIDTH=600 HEIGHT=100>
2: </APPLET>
```

Sie lernen über das HTML-Tag `<APPLET>` an anderer Stelle in diesem Buch mehr. Zwei Dinge sollen hier allerdings angemerkt sein:

- Das Attribut `CODE` gibt den Namen der Klasse an, die das Applet beinhaltet – in diesem Beispiel `Palindrome.class`.
- Die Attribute `WIDTH` und `HEIGHT` legen fest, wie groß das Applet-Fenster auf der Webseite in Pixeln sein wird. In diesem Beispiel wird das Fenster 600 Pixel breit und 100 Pixel hoch sein.

Um dieses Applet sehen zu können, benötigen Sie einen Web-Browser, der Java-Applets ausführen, kann oder das Tool `appletviewer` aus dem JDK.

Alle Applets in diesem Buch verwenden, sofern nicht anders angegeben, nur Features von Java 1.0.2, so daß die Applets mit allen Browsern, die Java unterstützen, angezeigt werden können. Applikationen werden die Features von Java 1.2 verwenden, da diese direkt mit dem Java 1.2 Interpreter ausgeführt werden können.

Um die Webseite `Palindrome.html` in einem Browser zu öffnen, verwenden Sie ein Menükommando zum Öffnen lokaler Dateien anstelle von Seiten aus dem Web. Im Netscape Navigator 4.04 ist dies das Kommando DATEI | SEITE ÖFFNEN | DATEI WÄHLEN.

Um eine Seite mit dem `appletviewer`-Tool des JDK zu öffnen, wechseln Sie an der MS-DOS-Eingabeaufforderung in das Verzeichnis, das die Datei `Palindrome.html` beinhaltet, und geben das folgende Kommando ein:

```
appletviewer Palindrome.html
```

Anders als ein Web Browser zeigt der Appletviewer nur das Applet (oder die Applets) an, die sich auf der Webseite befinden. Alles andere, was sich auf der Seite befindet, verarbeitet er nicht.

Die Abbildung 2.7 zeigt das Applet in dem Appletviewer.

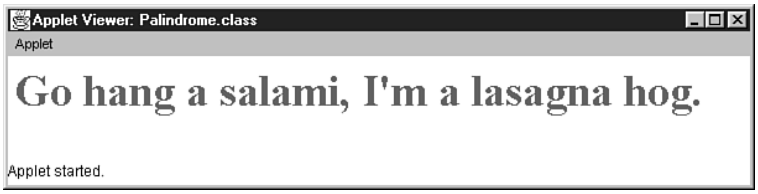

Abbildung 2.7: Das Palindrome-Applet im Appletviewer

Wenn Sie noch nicht damit vertraut sind, was ein Palindrom ist, dann sehen Sie sich einmal die Abbildung 2.7 an, und lesen Sie den Satz »Go hang a salami, I'm a lasagna hog« einmal rückwärts. Palindrome sind Wörter oder Ausdrücke, die sich von vorne wie von hinten gleich lesen – wenn Sie alle Leer- und Satzzeichen vernachlässigen. Die Wörter »Otto« und »Rentner« wären weitere Beispiele für Palindrome. Im Web finden Sie unter der URL `http://www.tsoft.net/~derf/palindrome.html`, Neil/Fred's Gigantic List of Palindromes, eine umfangreiche Liste von Palindromen.

Zusammenfassung

Wenn dies Ihre erste Begegnung mit der objektorientierten Programmierung war, dann haben Sie vielleicht noch eine weitere Gemeinsamkeit mit Bier festgestellt.

Die objektorientierte Programmierung ist auch in der Lage, Sie schwindelig und verwirrt zu machen und vielleicht sogar einen Brechreiz auszulösen.

Wenn Ihnen der Stoff des heutigen Tages theoretisch und erschlagend erschien, dann sollten Sie nicht beunruhigt sein. Sie werden die objektorientierten Techniken im Rest des Buches verwenden, und diese werden Ihnen um so vertrauter werden, je mehr Erfahrung Sie damit haben.

Eine der größten Hürden der objektorientierten Programmierung ist nicht notwendigerweise das Konzept selbst, sondern sind die verwendeten Bezeichnungen. In der OOP findet sich mehr Jargon und eine leicht bedrohlich klingende technische Sprache als in einer Folge von *AkteX*.

Um den Stoff des heutigen Tages zusammenzufassen, finden Sie im folgenden ein Glossar der Begriffe und Konzepte, die heute behandelt wurden:

***Klasse*:**	Eine Vorlage für ein Objekt. Diese beinhaltet Variablen, um das Objekt zu beschreiben, und Methoden, um zu beschreiben, wie sich das Objekt verhält. Klassen können Variablen und Methoden von anderen Klassen erben.
***Objekt*:**	Eine Instanz einer Klasse. Mehrere Objekte, die Instanzen derselben Klasse sind, haben Zugriff auf dieselben Methoden, aber oft unterschiedliche Werte für deren Instanzvariablen.
***Instanz*:**	Dasselbe wie ein Objekt. Jedes Objekt ist eine Instanz einer Klasse.
***Methode*:**	Eine Gruppe von Anweisungen in einer Klasse, die definieren, wie sich Objekte dieser Klasse verhalten werden. Methoden sind analog zu Funktionen in anderen Programmiersprachen. Im Unterschied zu Funktionen müssen sich Methoden immer innerhalb einer Klasse befinden.
***Klassenmethode*:**	Eine Methode, die auf eine Klasse selbst angewendet wird und nicht auf eine bestimmte Instanz einer Klasse.
***Instanzmethode*:**	Eine Methode, die auf Instanzen einer Klasse angewendet wird. Da Instanzmethoden wesentlich häufiger verwendet werden als Klassenmethoden, werden Sie auch einfach nur als Methoden bezeichnet.
***Klassenvariable*:**	Eine Variable, die ein Attribut einer ganzen Klasse anstatt einer bestimmten Instanz einer Klasse beschreibt.
***Instanzvariable*:**	Eine Variable, die ein Attribut einer Instanz einer Klasse beschreibt.
***Schnittstelle*:**	Eine Beschreibung abstrakter Verhaltensweisen, die einzelne Klassen implementieren können.
***Paket*:**	Eine Sammlung von Klassen und Schnittstellen. Klassen, die sich nicht in dem Paket `java.lang` befinden, müssen explizit importiert oder direkt über den vollen Paket- und Klassennamen angesprochen werden.

***Subklasse*:**	Eine Klasse, die sich in der Klassenhierarchie weiter unten befindet als eine andere Klasse, ihre Superklasse. Das Erzeugen einer Klasse, die Features einer bestehenden Klasse erbt, wird oft auch als Ableiten bezeichnet.
***Superklasse*:**	Eine Klasse, die sich in der Klassenhierarchie weiter oben befindet als eine oder mehrere andere Klasse(n), ihre Subklasse(n). Eine Klasse kann nur eine Superklasse direkt über sich haben. Diese Klasse kann aber ihrerseits wieder eine Superklasse haben usw. Zu einer Superklasse kann es beliebig viele Subklassen geben.

Fragen und Antworten

F Methoden sind doch eigentlich nichts anderes als Funktionen, die innerhalb von Klassen definiert wurden. Wenn sie wie Funktionen aussehen und sich auch wie Funktionen verhalten, warum nennt man Sie dann nicht Funktionen?

A *In einigen objektorientierten Programmiersprachen werden sie Funktionen genannt (in C++ werden sie als Elementfunktionen bezeichnet). Andere objektorientierte Programmiersprachen unterscheiden zwischen Funktionen innerhalb und außerhalb des Rumpfes einer Klasse oder eines Objekts, da bei diesen Sprachen die Unterscheidung zwischen den Begriffen wichtig ist zum Verständnis, wie die einzelnen Funktionen arbeiten. Da dieser Unterschied in anderen Sprachen relevant und der Begriff »Methode« in der objektorientierten Programmierung üblich ist, verwendet Java ihn auch.*

F Was ist der Unterschied zwischen Instanzvariablen und -methoden und deren Gegenstücken Klassenvariablen und -methoden?

A *Nahezu alles, was Sie in Java programmieren, bezieht sich auf Instanzen (auch Objekte genannt) und nicht auf die Klassen selbst. Für manche Verhaltensweisen und Attribute ist es allerdings sinnvoller, sie in der Klasse selbst zu speichern als in einem Objekt. Um z.B. eine neue Instanz einer Klasse zu erzeugen, benötigen Sie eine Methode, die für die Klasse selbst definiert und verfügbar ist. Andernfalls würden Sie in ein Henne-Ei-Dilemma hineingeraten – Sie können kein Baby-Objekt erzeugen ohne ein Mama-Objekt, das über eine Methode zum Babymachen verfügt. Und kein Mama-Objekt kann existieren, ohne zuvor ein Baby gewesen zu sein.*