In diesem Kapitel wurden an verschiedenen Stellen Beispiele gezeigt, in denen Schlüsselwörter wie public oder private zusammen mit bestimmten Programmelementen verwendet wurden. Mit Hilfe dieser Attribute können die Eigenschaften von Klassen, Methoden und Variablen verändert werden. Sie haben insbesondere Einfluß auf die Lebensdauer, Sichtbarkeit und Veränderbarkeit dieser Programmelemente und werden meist als Modifier bezeichnet. Wir wollen sie nun im Zusammenhang betrachten und ihre Wirkungsweise auf die verschiedenen Elemente eines Java-Programms erläutern.

8.2.1 Sichtbarkeit

Die eingangs erwähnte Tatsache, daß in einer abgeleiteten Klasse alle Eigenschaften der Basisklasse übernommen werden, ist nicht in allen Fällen ganz korrekt. Zwar besitzt sie immer alle Variablen und Methoden der Basisklasse, kann aber unter Umständen nicht darauf zugreifen, wenn ihre Sichtbarkeit eingeschränkt wurde.

Die Sichtbarkeit von Variablen und Methoden wird mit Hilfe folgender Modifier geregelt:

- Elemente des Typs `public` sind in der Klasse selbst (also in ihren Methoden), in Methoden abgeleiteter Klassen und für den Aufrufer von Instanzen der Klasse sichtbar.
- Elemente des Typs `protected` sind in der Klasse selbst und in Methoden abgeleiteter Klassen sichtbar. Zusätzlich können Klassen desselben Pakets sie aufrufen.
- Elemente des Typs `private` sind lediglich in der Klasse selbst sichtbar. Für abgeleitete Klassen und für Aufrufer von Instanzen bleiben `private`-Variablen verdeckt.
- Elemente, die ohne einen der drei genannten Modifier deklariert wurden, werden als *package scoped* oder Elemente mit *Standard-Sichtbarkeit* bezeichnet. Sie sind nur innerhalb des Pakets sichtbar, zu dem diese Klasse gehört. In anderen Paketen sind sie dagegen unsichtbar.

Tip

Mit Hilfe dieser Sichtbarkeitsebenen kann der Zugriff auf Klassenelemente eingeschränkt werden. private-Elemente sollten immer dann verwendet werden, wenn implementierungsabhängige Details zu verstecken sind, die auch in abgeleiteten Klassen nicht sichtbar sein sollen. protected-Elemente sind vor Zugriffen von außen geschützt, können aber von abgeleiteten Klassen verwendet werden. Die public-Elemente schließlich bilden die für alle sichtbaren Teile einer Klassendefinition und können daher als ihre Schnittstelle angesehen werden. Nachfolgend werden die verschiedenen Sichtbarkeitsattribute noch einmal genau beschrieben. Elemente mit Standard-Sichtbarkeit verhalten sich innerhalb des Pakets wie public- und außerhalb wie private-Elemente.

8.2.2 Die Attribute im Überblick

Nachfolgend wollen wir die wichtigsten Attribute noch einmal zusammenfassend darstellen und ihre jeweiligen Auswirkungen auf die Sichtbarkeit, Lebensdauer oder Veränderbarkeit von Variablen, Methoden und Klassen beschreiben.

private

Methoden oder Variablen vom Typ private sind nur in der aktuellen Klasse sichtbar, in allen anderen Klassen bleiben sie dagegen unsichtbar.

Hinweis

Diese Einschränkung bedeutet überraschenderweise nicht, daß die Methoden einer Klasse nur auf die privaten Membervariablen des eigenen Objekts zugreifen dürfen. Vielmehr ist ebenfalls möglich, (quasi von außen) auf die `private`-Variablen eines *anderen* Objekts zuzugreifen. Vorausgesetzt, es handelt sich um eine Instanz derselben Klasse.

Das folgende Beispielprogramm demonstriert dies mit Hilfe der Klasse `ClassWithPrivateA`, die eine private Membervariable `a` besitzt. An der Implementierung von `setOtherA` können wir erkennen, wie der Zugriff auf fremde Objekte desselben Typs möglich ist:

Hinweis

```
001 /* Listing0806.java */
002
003 public class Listing0806
004 {
005   public static void main(String[] args)
006   {
007     ClassWithPrivateA a1 = new ClassWithPrivateA(7);
008     ClassWithPrivateA a2 = new ClassWithPrivateA(11);
009     a2.setOtherA(a1, 999);
010     System.out.println("a1 = " + a1.toString());
011     System.out.println("a2 = " + a2.toString());
012   }
013 }
014
015 class ClassWithPrivateA
016 {
017   private int a;
018
019   public ClassWithPrivateA(int a)
020   {
021     this.a = a;
022   }
023
024   public void setOtherA(ClassWithPrivateA other, int newvalue)
025   {
026     other.a = newvalue;
027   }
028
029   public String toString()
030   {
031     return "" + a;
032   }
033 }
```

Listing 8.6: Zugriff auf fremde private Membervariablen

An der Ausgabe des Programms kann man erkennen, daß über das Objekt a2 auf private Membervariablen des Objekts a1 zugegriffen wurde:

```
a1 = 999
a2 = 11
```

protected

Methoden oder Variablen vom Typ protected sind in der aktuellen Klasse und in abgeleiteten Klassen sichtbar. Darüber hinaus sind sie für Methoden anderer Klassen innerhalb desselben Pakets sichtbar. Sie sind jedoch nicht für Aufrufer der Klasse sichtbar, die in anderen Paketen definiert wurden.

public

Membervariablen und Methoden vom Typ public sind im Rahmen ihrer Lebensdauer überall sichtbar. Sie können daher in der eigenen Klasse und von beliebigen Methoden anderer Klassen verwendet werden. Das Attribut public ist zusätzlich auch bei der Klassendefinition selbst von Bedeutung, denn nur Klassen, die als public deklariert wurden, sind außerhalb des Pakets sichtbar, in dem sie definiert wurden. In jeder Quelldatei darf nur eine Klasse mit dem Attribut public angelegt werden.

Standard (package scoped)

Klassen, Methoden, Variablen mit Standard-Sichtbarkeit sind nur innerhalb des Pakets sichtbar, in dem sie definiert wurden. Sie sind beispielsweise nützlich, um in aufwendigeren Paketen allgemein zugängliche Hilfsklassen zu realisieren, die außerhalb des Pakets unsichtbar bleiben sollen. Sie können mitunter nützlich sein, um zu verhindern, daß Elemente als public deklariert werden.

static

Variablen und Methoden mit dem Attribut static sind nicht an die Existenz eines konkreten Objekts gebunden, sondern existieren vom Laden der Klasse bis zum Beenden des Programms. Das static-Attribut beeinflußt bei Membervariablen ihre Lebensdauer und erlaubt bei Methoden den Aufruf, ohne daß der Aufrufer ein Objekt der Klasse besitzt, in der die Methode definiert wurde.

Wird das Attribut static nicht verwendet, so sind Variablen innerhalb einer Klasse immer an eine konkrete Instanz gebunden. Ihre Lebensdauer beginnt mit dem Anlegen des Objekts und dem Aufruf eines Konstruktors und endet mit der Freigabe des Objekts durch den Garbage Collector.

final

Membervariablen mit dem Attribut final dürfen nicht verändert werden, sind also als Konstanten anzusehen. Methoden des Typs final dürfen nicht überlagert werden; ebensowenig dürfen Klassen des Typs final zur Ableitung neuer Klassen verwendet werden. Wird das Attribut final dagegen nicht verwendet, sind Membervariablen veränderbar, können Methoden überlagert und Klassen abgeleitet werden.

Falls eine Methode oder Klasse das Attribut final besitzt, kann der Compiler auf die dynamische Methodensuche verzichten. final-Methoden können daher performanter aufgerufen werden als normale Methoden. Dies ist einer der Gründe dafür, daß die Java-Designer einige der mitgelieferten Klassen als final deklariert haben. Es führt aber gleichzeitig dazu, daß die entsprechenden Klassen nicht mehr erweitert werden können. Ein prominentes Beispiel aus der Laufzeitbibliothek ist die als final deklarierte Klasse String.

Seit dem JDK 1.1 kann das final-Attribut auch auf Parameter von Methoden und lokale Variablen angewendet werden. Dadurch stellt der Compiler sicher, daß die Variable bzw. der Parameter nach der Initialisierung nicht mehr verändert wird. Die Initialisierung muß dabei nicht unbedingt bei der Deklaration erfolgen, sondern kann auch später vorgenommen werden. Wichtig ist, daß nur genau einmal ein Wert zugewiesen wird.

Warnung

Im Gegensatz zu C oder C++ gibt es allerdings bei als final deklarierten Objektparametern keine Möglichkeit, zwischen dem Objekt insgesamt und seinen einzelnen Elementen zu unterscheiden. Eine als final deklarierte Objektvariable wird zwar insgesamt vor Zuweisungen geschützt, der Wert einzelner Membervariablen kann jedoch verändert werden. Dies gilt analog für Arrays, die ja ebenfalls Objekte sind: final bietet keinen Schutz gegen die unerwünschte Zuweisung eines Werts an ein einzelnes Element des Arrays.