Programmierparadigmen: Objektorientierte Programmierung

Julian Lang

4. Juni 2012

1 Einleitung

Diese Arbeit gibt einen kurzen Überblick über die Objektorientierung und ihre wichtigsten Eigenschaften. Dabei werden die 3 Basiswerkzeuge der Objektorientierung kurz vorgestellt: die Datenkapselung, die Vererbung und die Polymorphie.

2 Objektorientierung

Die objektorientierte Programmierung ist ein Programmierparadigma, das erfunden wurde um die Probleme immer größer werdender Software zu bewältigen. Da heutige Softwareprojekte aus unzähligen Funktionen bestehen versucht die Objektorientierung diese Funktionalitäten durch verschiedene Mechanismen logisch aufzuteilen und dadurch besser plan- und organisierbar zu machen.

3 Das Objekt

3.1 Objekte als Datenkapsel

Ein Computerprogramm besteht aus zwei Bauteilen: Daten und Instruktionen. In klassischen Computerprogrammen können Daten von jeder Instruktion des Programmes bearbeitet werden. In der objektorientierten Programmierung wird der Zugriff auf die Daten des Programms auf einen bestimmten Teil des Programmes beschränkt: dem sogenannten Objekt. Einem Objekt werden bestimmte Daten zugewiesen, die so vom Rest des Programms abgekapselt werden. Dabei werden von einem Objekt Funktionen bereitgestellt, die das Arbeiten mit den Daten auch dem restlichen Code erlauben. Diese Funktionen werden in der Objektorientierung Methoden genannt. Technisch gesehen ist ein Objekt also die Zusammenfassung von Daten und Funktionalität.

3.2 Methoden

Nach außen hin stellt ein Objekt seine Eigenschaften durch Methoden zur Verfügung. Neben den eigentlichen Datenwerten können aber auch berechnete Werte durch Methoden bereitgestellt werden um so Berechnungen mit den Daten, dem Objekt zu überlassen. Es ist aber auch möglich, dass ein Methodenaufruf weitere Methoden von anderen Objekten ausführt. Das Objekt an sich bestimmt jedoch, wie bei einem Methodenaufruf Daten verarbeitet werden, und stellt so den korrekten Zugriff auf die Daten sicher. Das restliche Programm kann auf die Daten nur noch durch Methodenaufrufe zugreifen. Da so die Aufgabe des Zugriffs auf die Daten dem Objekt zugewiesen wird, muss sich der restlichte Code nicht mehr darum kümmern.

3.3 Verantwortung des Objekts

Ein Objekt übernimmt die Verantwortung die von ihm bereitgestellten Methoden korrekt auszuführen. Dadurch kann den restlichen Teilen des Codes nur durch Kenntnis der Methodenaufrufe die gewünschte Funktionalität bereitgestellt werden, ohne von den Einzelheiten der Daten und Implementierung abhängig zu sein. Zusätzlich zur korrekten Ausführung können bei der Definition von Methoden auch bestimmte Vorbedingungen, Nachbedingungen oder Invarianten von der Methode gefordert werden. Diese Verantwortung werden in der Objektorientierung auch als Kontrakt zwischen dem Aufrufer und dem Objekt bezeichnet. Dies ist besonders wichtig, da so Objekte durch andere Objekte mit denselben Methoden ausgetauscht werden können (siehe Vererbung und Polymorphie).

3.4 Objektidentität

In einem objektorientierten System kann ein Objekt nicht mit einem bestimmten Wert gleichgesetzt werden. Ein Objekt besitzt im gegensatz zu einem Wert immer eine eigene Identität. So kann es sein, dass auf einem System zwei verschiedene Objekte vorhanden sind, die beide dieselben Daten, aber dennoch unterschiedliche Identitäten besitzen. Bei solchen Objekten ist es also möglich, dass eines dieser beiden Objekte nach einem Aufruf seinen Zustand verändert, dals andere jedoch seinen Zustand behält. Deshalb muss man in objektorientierten Systemen darauf achten, ob man bei einer Überprüfung auf Gleichheit zweier Objekte die Daten innerhalb der Objekte vergleichen will oder die beiden Objekt-Identitäten.

3.5 Beispiel für Datenkapselung

Als Beispiel für die Datenkapselung kann man sich ein Objekt vorstellen, welches die Daten eines Rechtecks speichert und dieses bei jeder Aktualisierung auf dem Bildschirm zeichnet. Das Objekt stellt die Methoden *getWidth*, *getHeight*, *getArea*, *setWidth* und *setHeight* bereit. Intern hat das Objekt die Werte für die Länge und Breite des Rechtecks gespeichert und gibt diese bei einem Aufruf von den jeweiligen Methoden zurück. Beim Aufruf von getArea wird anhand der Längen- und Breitenwerte die Fläche des Rechtecks

ausgerechnet. Beim Aufruf von einer der *set* Methoden wird automatisch das Rechteck neu angezeigt. Diese Implementierung hat den Vorteil, dass nicht jeder Codeteil der auf die Fläche zugreifen will diese selbst berechnen muss. Zusätzlich wird verhindert, dass andere Teile des Codes die Daten des Rechteckes verändern, ohne dass die Ausgabe aktualisiert wird.

4 Klassen

In einem Computerprogramm hat man meist mehrere Objekte, die dieselben „Dinge“ beschreiben. So würde es im vorherigen Beispiel mehrere Objekte geben, die jeweils ein Rechteck beschreiben. Da für alle solchen Objekte die gleichen Bedingungen gelten und die gleichen Berechnungen gemacht werden müssen, kann man diese Gemeinsamkeiten in der Objektorientierung zu einer sogenannten Klasse zusammenfassen. Dabei wird die Klasse als Bauplan für die einzelnen Objekte verwendet. Eine Klasse beschreibt alle Daten die ein Objekt speichert und implementiert die Methoden die die daraus erzeugten Objekte haben. Die Klasse kann man also auch als Datentyp der einzelnen Objekte bezeichnen. Dabei übernimmt die Klasse die gesamten Aufgaben, die bei der Initialisierung und Zerstörung des Objekts vorgenommen werden müssen.

5 Vererbung und Polymorphie

In der Objektorientierung ist es auch möglich, dass Klassen Beziehungen untereinander haben. So kann eine Klasse ihre Spezifikation und sogar ihren Code an eine andere Klasse weitergeben. Dabei wird die Klasse die ihre Eigenschaften weitergibt Oberklasse, und jene die sie erhält Unterklasse genannt. Dabei wird zwischen Vererbung der Spezifikation und Vererbung der Implementierung unterschieden.

5.1 Vererbung der Spezifikation

Bei der Vererbung der Spezifikation erbt die Unterklasse alle Methoden, die in der Oberklasse spezifiziert wurden. Damit hat die Unterklasse die Aufgabe jede Methode die in der Oberklasse definiert ist, selbst korrekt zu implementieren. Die Unterklasse selbst kann jedoch bestimmen wie sie die Methoden implementiert solange folgende Regeln zu den Vorbedingungen, Nachbedingungen und Invarianten nicht verletzt werden:

1. Die Vorbedingungen können geschwächt werden.
2. Die Nachbedingungen können verschärft werden.
3. Auch in der Unterklasse müssen alle Invarianten gelten.

Wenn eine Klasse von mehreren Klassen erbt, wird dies Mehrfachvererbung genannt. Dadurch dass nun mehrere Objekte dieselbe Schnittstelle haben, ist es auch möglich,

dass man an jeder Stelle an der man die Oberklasse verwendet hat, nun auch die Unterklasse verwenden kann. Da beide Objekte dieselbe Schnittstelle haben, ist es für alle aufrufenden Objekte egal ob die Oberklasse oder die Unterklasse verwendet wird.

5.2 Vererbung der Implementierung

Bei der Vererbung der Implementierung erbt die Unterklasse neben allen Spezifikationen auch die Implementierung für die in der Oberklasse definierten Methoden. Dadurch kann redundanter Code in der Unterklasse verhindert werden. Bei der Verwendung von Mehrfachvererbung kann es jedoch zu Problemen kommen.

5.3 Dynamische Polymorphie

Dynamische Polymorphie bedeutet, dass eine Unterklasse ihre geerbten Methoden auch anders als die Oberklasse Implementieren kann. Dadurch ist es möglich durch den Austausch von Objekten die gesamte Funktionalität eines Programmes zu verändern. Durch diesen Mechanismus ist es möglich, dass erst zur Laufzeit bestimmt wird welches Objekt und damit auch welche Implementierung einer Methode verwendet wird.

5.4 Statische Polymorphie

Statische Polymorphie bezeichnet einen Mechanismus der es erlaubt Methoden zu haben, die unterschiedlich auf andere Eingabewerte reagieren. Dieser Mechanismus wird meist auch Überladung genannt.

6 Ausblick

In dieser kurzen Einführung zur Objektorientierung wurde nur auf einige der vielen Mechanismen der Objektorientierung eingegangen. Die Objektorientierung bietet noch viele weitere Mechanismen an die aber den Rahmen dieser Arbeit sprengen würden.

Literatur

[1] Peter Forbrig. Objektorientierung - stand und aktuelle entwicklungen. *LOG IN*, 24(128/129):12–19, 2004.

[2] Bernhard Lahres and Gregor Rayman. *Objektorientierte Programmierung - Das umfassende Handbuch.* Galileo Press GmbH, Bonn, 2. aufl. edition, 2009.

[3] Max Vetter. *Objektmodellierung - eine Einführung in die objektorientierte Analyse und das objektorientierte Design (2. Aufl.).* Leitfäden der Informatik. Teubner, 1998.