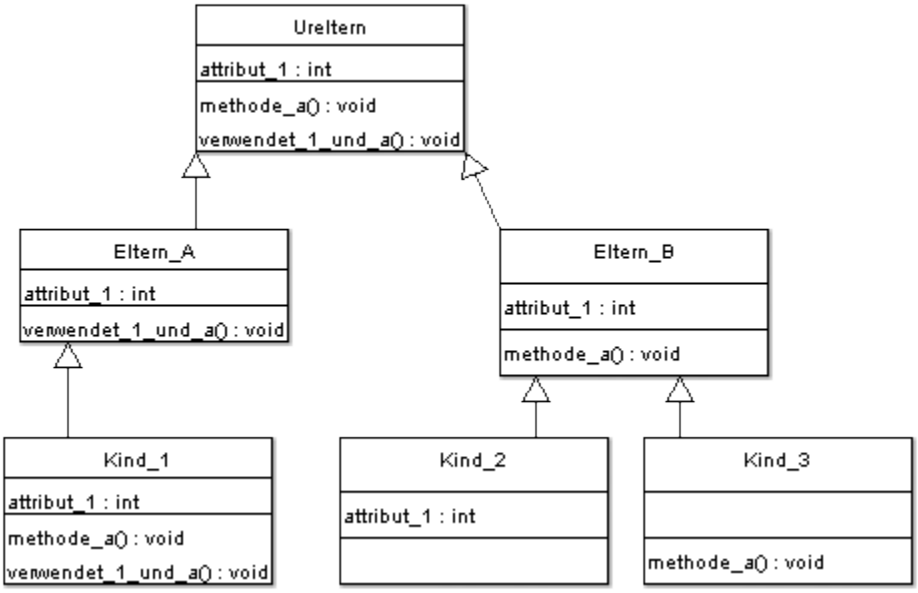
Die Sache mit dem Überschreiben

Das Überschreiben (override – nicht verwechseln mit overload – Überladen) ist nicht ganz so einfach, wie es im Abschnitt zu den Entwurfsmustern abgehandelt wird. Dort heißt es:

Die Klasse Gruppe allerdings muss diese Methoden auch inhaltlich implementieren. Das Bemerkenswerte ist nun, dass ein Gruppenobjekt „wissen" muss, dass es nicht die Methode aus der Klasse Moebel verwenden darf, sondern dass es die zu ihr gehörende Methode verwenden muss. Man nennt diesen Vorgang Überschreiben (override).

Um die zu lösenden Aufgaben deutlich zu machen, betrachten wir einmal folgenden fiktiven Vererbungsbaum[1]:



Nehmen wir einmal an, wir erzeugen eine Instanz der Klasse **`Kind_1`**. Wir wollen das Objekt mit **`kind_1`** bezeichnen. Welche Attribute und Methoden sind für dieses Objekt gültig? Die Antwort ist ganz einfach, es sind die in der Klassendefinition angegebenen **`attribut_1()`**, **`methode_a()`** und **`verwendet_1_und_a()`**. Bei dieser Konstellation fragt man sich, weshalb hier überhaupt Vererbung eingesetzt wurde. Sie wäre bei **`Kind_1`** nur sinnvoll, wenn es weitere Attribute und Methoden gibt, die vererbt werden konnten.

Anders sieht es aus, wenn wir ein Objekt **`kind_2`** von **`Kind_2`** erzeugen. Wenden wir nun die Methode **`verwendet_1_und_a()`** an, die in der Klasse **`Ureltern`** definiert wurde, dann greift diese nicht auf das in der Klasse **`Ureltern`** definierte Attribut **`attribut_1()`** zu, sondern auf **`attribut_1()`** von der zum Objekt gehörenden Klasse **`Kind_2`**. Andererseits verwendet diese aber – da **`Kind_2`** diese nicht selbst implementiert – die Methode **`methode_a()`**, die in der Klasse **`Eltern_B`** definiert wurde.

Ähnlich, nur dass statt der Methode von **`Eltern_B`** die eigene Methode verwendet wird, geht es bei einer Instanz **`kind_3`** von **`Kind_3`**. Hier wird auf das Attribut von **`Eltern_B`** zurückgegriffen.

Woher aber weiß die in **`Ureltern`** definierte Methode **`verwendet_1_und_a()`** von **`kind_2`** aber, dass sie das Attribut aus **`Kind_2`** verwenden muss, jedoch für **`kind_3`** das Attribut der eigenen Klasse, dafür aber die Methode **`methode_a()`** von **`Kind_3`**?

Die Lösung ist eine jedem Objekt zugeordnete Tabelle von Adressen der Attribute und

1 Dabei ist hier keine abstrakte Klasse eingebaut, weil daraus keine neuen Erkenntnisse zu gewinnen sind.

Methoden, die dieses verwenden muss. Man kann sich das so vorstellen, dass beim Erzeugen eines Objektes zunächst alle die Adressen für eine Urelternobjekt angelegt werden, dann die des Elternobjektes und dann die des Kindobjektes. Wird dabei ein Attribut oder eine Methode erneut definiert, dann überschreibt (ersetzt) sie einfach den bisherigen Wert.

Das Verständnis der Vererbung „von oben nach unten“ ist in der Regel sehr leicht: Die erbenden Klassen „kennen eben“ das, was sie erben. Schwieriger ist zu verstehen – s.o. - weshalb scheinbar auch die Methoden der vererbende Klasse die Attribute und Methoden der erbenden kennen. Hier ist „scheinbar“ wichtig. Die Kenntnis wird trotzdem von oben nach unten weiter gereicht, nur eben neu zugeordnet. Das erkennt man daran, dass eine Methode einer vererbenden Klasse natürlich nicht auf eine Methode zugreifen kann, die ihr nicht bekannt ist, da sie in ihr noch nicht definiert wurde – und sei es denn, wie bei einer abstrakten Klasse – zumindest mit ihrer Signatur.

Aufgabe:

Untersuchen Sie für weitere Objekte zu allen Klassen des o.a. Vererbungsbaumes, wie und welche Attribute und Methoden sie verwenden.