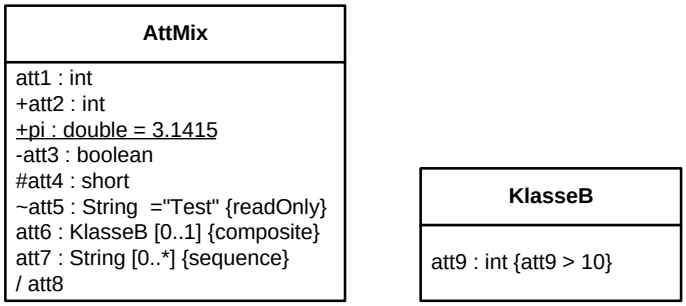
SOPHIST

Codebeispiele für Attribute

Kapitel 4: Klassendiagramm



Umsetzung in C++

C++ fordert die Deklaration aller Attribute innerhalb des Rumpfes der Klassendefinition. Die Sichtbarkeit der einzelnen Attribute wird durch die Schlüsselwörter `public`, `private` oder `protected` angegeben, welche durch Doppelpunkt von den so gekennzeichneten Attributen abgetrennt sind. Wird die Angabe weggelassen, so gilt `private` als Vorgabe.

Da C++ über kein explizites Paketkonzept verfügt, wird konsequenterweise das Konzept der UML-Sichtbarkeitseinschränkung `package` nicht umgesetzt.

C++ verbietet die Initialisierung von Variablen innerhalb der Klassendeklaration, weshalb weder `att2` noch `pi` der vorgesehene Wert zugewiesen werden kann. Als Alternativlösung kann die gewünschte Initialisierung im Rahmen des Konstruktors oder, wie im folgenden Beispiel, außerhalb der Klassendeklaration vorgenommen werden. Für dieses Vorgehen ist jedoch die Sichtbarkeit, welche vorgabegemäß auf `private` gesetzt ist, zwingend in `public` abzuändern, da ansonsten kein Zugriff auf `pi` erfolgen könnte.

Die Typisierung des Attributs `att6` erfolgt durch Angabe eines Verweises (auch als *Referenz* oder *Pointer* bezeichnet) auf die (im Beispiel weggelassene) `KlasseB`. Ist dieser Klasse keine Ausprägung zugeordnet, so kann `att6` mit dem Pseudowert `NULL` belegt werden.

Für die Realisierung des mengenwertigen Attributs `att7` wird auf die C++-Standard Template Library (STL) [Bre98] zurückgegriffen. Sie stellt mit dem vector-Konstrukt einen Mechanismus zur Ablage einer nach oben nicht beschränkten Menge von Ausprägungen eines frei festlegbaren Typs zur Verfügung. Im Beispiel wurde als Typ dieser Menge gemäß den Festlegungen des Klassendiagramms der STL-Typ `string` gewählt.

```
class AttMix {
     int att1;
     KlasseB *att6;
      vector<string> att7;
     public:
           int att2;
           static double pi;
           static const string att5;
     private:
           bool att3;
     protected:
```

```
            short att4;
};
const string AttMix::att5("Test");
double AttMix::pi=3.1415;
```

Umsetzung in Java

Java gestattet die Definition von Attributen, die in der Terminologie der Programmiersprache als „Felder“ bezeichnet werden, innerhalb des durch geschweifte Klammern umschlossenen Rumpfs einer Klasse.

Das Beispiel zeigt die Umsetzung der verschiedenen Deklarationsformen, zunächst die möglichen Sichtbarkeitseinschränkungen. Ein Attribut wie `att1`, für das die Sichtbarkeit nicht spezifiziert ist, ist für die Klasse selbst, all ihre Unterklassen sowie alle Klassen, die sich im selben Paket wie die deklarierende Klasse befinden, sicht- und zugreifbar. Dies entspricht in etwa der Sichtbarkeitseinschränkung `package`, welche nicht als explizites Java-Sprachkonstrukt unterstützt wird.

Das Attribut `att2` wird bereits bei seiner Erzeugung mit einem Wert vorbelegt, sodass jede Ausprägung der Klasse `AttMix` automatisch nach ihrer Erzeugung bereits einen Wert für `att2` besitzt.

Ein Klassenattribut, das heißt ein Attribut, dessen Wert für die Klasse selbst und alle ihre Ausprägungen in gleicher Weise zugreifbar ist, wird durch `pi` beispielhaft veranschaulicht.

`att3` und `att4` zeigen Umsetzungen, die in eindeutiger Weise dem Klassendiagramm entsprechen.

Das Zeichenkettenattribut `att5` illustriert die Realisierung der Einschränkung `readOnly`, die in Java mit dem Schlüsselwort `final` korrespondiert. Gleichzeitig wird die Zeichenkette mit dem konstanten Wert `Test` belegt. Die Sichtbarkeitseinschränkung wurde weggelassen, da die Vorgabebelegung der Programmiersprache Java dem UML-Wert `package` entspricht.

Das Attribut `att6` ist vom Datentyp `KlasseB` und kann daher Objekte dieser Klasse aufnehmen. Durch das Multiplizitätsintervall `[0..1]` wird die Forderung erhoben, auch die Ablage keines Wertes explizit auszudrücken. Sie lässt sich in Java durch Zuweisung des gesonderten Wertes `null` erreichen.

Die Anlage von `att7` zeigt eine mögliche Realisierung für ein mengenwertiges Attribut (gefordert durch das Multiplizitätsintervall `[0..*]`), dessen Inhaltswerte vom Typ `String` geordnet (gefordert durch die Eigenschaft `ordered`) verwaltet werden. Das Beispiel verwendet hierzu die ab Java-Version 1.5 zur Verfügung stehenden generischen Mechanismen, die es erlauben, den Inhaltstyp des Typs `Vector` einzuschränken. `Vector` selbst dient hierbei zur Umsetzung des geforderten geordneten, mengenwertigen Attributs, da diese Standardklasse eine dynamische Aufzählung zur Verfügung stellt.

Das abgeleitete Attribut `att8` wird nicht in ein Java-Feld umgesetzt, da sein Wert aus den Inhalten anderer Attribute oder sonstiger Systemkomponenten dynamisch zur Laufzeit berechnet werden kann. Das Beispiel zeigt eine Lösung unter Nutzung einer get-Methode an. Diese würden im Falle des Aufrufs den benötigten Wert berechnen und zurückliefern, statt auf einen separaten Speicherplatz für das genannte Attribut zuzugreifen.

Da für `att8` im Modell der Abbildung 3.12 kein Datentyp angegeben wurde, Java jedoch die Angabe zwingend fordert, nimmt das Beispiel hier `Object` an, da dies die Rückgabe beliebiger Objektausprägungen zulässt.

```
class AttMix {
     int att1;
     public int att2;
     public static double pi=3.1415;
     private boolean att3;
     protected short att4;
     final String att5 = "Test";
```

```
	KlasseB att6;
	java.util.Vector<String> att7;

	Object getAtt8() {
		//Berechne Wert für att8
		return wert;
	}
}
```