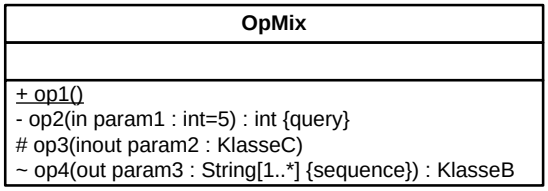
SOPHIST

Codebeispiele für Operationen

Kapitel 4: Klassendiagramm



Umsetzung in C++

Die C++-Syntax fordert zur Realisierung für jede Methode, die eine Operation aus dem Klassendiagramm der Abbildung 3.15 implementiert, die Festlegung eines Rückgabewertes. Per Konvention wird in C++ für alle Methoden, die keinen Rückgabewert definieren, das Schlüsselwort `void` gesetzt.

Die Umsetzung der Sichtbarkeiten ist identisch mit der Umsetzung der Attribute aus dem vorhergehenden Kapitel.

Ähnlich wie in der Darstellung des Klassendiagramms müssen für jede Methode, selbst wenn sie über keine Parameterliste verfügt, die begrenzenden runden Klammern angegeben werden. Daher wird die Operation `op1`, die über keine Parameter und keinen Rückgabewert verfügt, als `void op1()` umgesetzt. Auf den „Signatur“ genannten Operationsanteil folgt, von geschweiften Klammern umschlossen, die Methode als Implementierung der Operation.

Klassenoperationen, wie `op1`, werden durch das dem Rückgabetyp vorangestellte Schlüsselwort `static` gekennzeichnet.

Für Operation `op2` wurde mit `param1` ein Übergabeparameter definiert, der innerhalb der implementierenden Methode ausschließlich lesend verarbeitet werden darf. C++ setzt dies mit dem Schlüsselwort `const` um, das den Compiler zur Überwachung dieser Vereinbarung veranlasst. Wird im Rumpf der Methode `op2` versucht, schreibend auf den Inhalt von `param1` zuzugreifen, so erfolgt eine Fehlermeldung bereits zum Übersetzungszeitpunkt.

Darüber hinaus definiert das Klassendiagramm, dass auf die Übergabe eines Wertes für diesen Parameter vollkommen verzichtet werden kann. In diesem Falle wird ein solcher Aufruf durch das Laufzeitsystem so behandelt, als wäre der Aufruf mit Übergabe des Wertes 5 für den Parameter `param1` erfolgt. Zusätzlich definiert die Operation einen Rückgabewert vom Typ `int`. Eine Ausprägung dieses Typs muss nach Abarbeitung der Methode mittels der Anweisung `return` an den Aufrufer zurückgegeben werden.

Zusätzlich wird der Compiler durch das Schlüsselwort `const` angewiesen zu überwachen, dass diese Operation keinerlei Zustandsveränderungen am System, etwa durch das Schreiben von Attributen, vornimmt. So wird die UML-Eigenschaft `readOnly` umgesetzt.

Die Operation `op3` definiert einen mit `param3` benannten Parameter, auf den innerhalb der Methode ausschließlich schreibend zugegriffen werden kann. Zur Umsetzung in C++ muss hierfür der Inhaltsoperator – kenntlich am Stern, der dem Typnamen nachgestellt ist – verwendet werden. Insofern kann in der `op3` implementierenden Methode schreibend auf die Inhalte von `param3` zugegriffen werden, so dass diese auch nach Abarbeitung der Methode sichtbar sind.

Ein Sprachmittel zur Sicherung vor lesenden Zugriffen, etwa zur Fehlervermeidung bei der Übergabe nicht initialisierter Speicherbereiche zur Aufnahme der Ergebniswerte, bietet C++ nicht.

```
class OpMix {
      KlasseB op4(vector<string> param3) {
            //Implementierung
            return wert;
}
      public:
            static void op1() {
            //Implementierung
            }
      private:
            int op2(const int param1=5) const {
            //Implementierung
                  return wert;
            }
      protected:
            void op3(KlasseC* param2) {
            //Implementierung
            }
};
```

Umsetzung in Java

Die Umsetzung in Java erfordert ebenfalls die zwingende Angabe des Rückgabetyps, auch wenn kein Wert an den Aufrufer übermittelt wird. Ist kein Rückgabetyp definiert, so wird formal das Schlüsselwort `void` eingesetzt.

Die Umsetzung der Sichtbarkeiten erfolgt wie bei Attributen durch Voranstellen des entsprechenden Schlüsselwortes.

Sowohl Operation `op2` als auch `op4` definieren einen Rückgabewert. Sein Typ, der im Klassendiagramm definiert ist, wird vor dem Operationsnamen geschrieben und durch mindestens ein Leerzeichen abgetrennt.

Klassenoperationen, wie `op1`, werden durch das dem Rückgabetyp vorangestellte Schlüsselwort `static` gekennzeichnet.

Generell können Sie in Java alle objektartigen Übergabeparameter im Methodenrumpf schreiben (*call by reference*), so dass Veränderungen auch im aufrufenden Kontext sichtbar sind. Lediglich skalare Ausprägungen (Zahlen und Wahrheitswerte) werden per Wert (*call by value*) übergeben. Daher ist die Umsetzung der Übergaberichtungen des UML-Klassendiagramms in Java teilweise aufwändig zu lösen.

Zwar verhalten sich die Parameter `param1` und `param2` aufgrund ihrer Typisierung als skalarer Wert beziehungsweise Objekt in der beabsichtigten Weise, jedoch gilt das nicht für alle Fälle. Für die korrekte Realisierung der UML-Übergaberichtungen in Java sollten daher „in“-Parameter als Ausprägungen der Java-Primitivtypen übergeben werden. Somit überwacht das Laufzeitsystem automatisch die Einhaltung des Schreibverbotes. Ist dies nicht möglich, etwa weil der Übergabeparameter eine interne Struktur aufweist, dann sollten Sie sicherstellen, dass im Rumpf der Methode keine Schreiboperationen, etwa durch Aufruf von set-Methoden, vorgenommen werden.

Als „out“ oder „inout“ gekennzeichnete Parameter müssen zwingend als Objekte realisiert werden, selbst wenn sie nur einen Primitivwert kapseln, da andernfalls die im Rumpf der Methode vorgenommenen Änderungen nach dem Verlassen der Methode verworfen würden.

Die Umsetzung der Vorgabebelegung des Parameters `param1` mit 5 ist in Java nicht möglich, da die Parameterliste des Methodenaufrufs immer in Reihenfolge und Länge mit der der definierten Operation übereinstimmen muss.

Zur Realisierung des Veränderungsverbotes (Übergaberichtung `in`) für den durch `param1` übergebenen Wert bietet Java das Schlüsselwort `final` an. Seine Definition stellt sicher, dass im Methoderumpf keine schreibende Vorgänge für den Parameter `param1` zugelassen werden. Das durch die Eigenschaft `readOnly` auf Operationsebene ausgesprochene generelle Verbot ändernder Zugriffe, lässt sich in Java nicht umsetzen. (Zwar ist die Angabe von `final` auch für Operationen zulässig, doch ist es auf dieser Ebene mit einer anderen Semantik belegt.)

Für die durch `out` formulierte Einschränkung, dass die Inhalte von `param3` im Verlauf der Verarbeitung von `op4` nur geschrieben, nicht jedoch gelesen werden dürfen, existiert kein Sprachmittel. Die Übergabe einer Menge von String-Objekten als Parameter `param3` ist durch Nutzung der Array-Syntax möglich.

```
public class OpMix {
      public static void op1() {
            //Implementierung
      }
      private int op2(final int param1) {
            //Implementierung
            return wert;
      }
      protected void op3(KlasseC param3) {
            //Implementierung
      }
      KlasseB op4(String param3[]) {
            //Implementierung
            return wert;
      }
}
```