SE1, Aufgabenblatt 2

Softwareentwicklung I – Wintersemester 2011/12

Quelltext bearbeiten, Methoden, aktuelle und formale Parameter, Variablen

MIN-CommSy-URL: ..... https://www.mincommsy.uni-hamburg.de/
Projektraum: ..... SE-1 CommSy WiSe 11/12
Ausgabedatum: ..... 27. Oktober 2011

Kernbegriffe

Den Programmtext einer Klasse nennen wir ihren *Quelltext* (engl.: source code) bzw. ihre *Klassendefinition* (engl.: class definition). Wir bearbeiten Quelltexte mit einem *Editor*. Um Objekte einer Klasse erzeugen zu können, muss der Quelltext der Klasse zuvor *übersetzt* (kompiliert, engl.: to compile) werden. Dabei wird der menschenlesbare Quelltext von einem *Compiler* in eine maschinenausführbare Form überführt.

Im Quelltext einer Klasse ist unter anderem beschrieben, wie die Methoden der Klasse realisiert sind. Eine Methode besteht aus einem *Kopf* (engl: header), der ihre Signatur definiert, und einem *Rumpf* (engl.: body). Der Rumpf enthält eine *Sequenz* von *Anweisungen* (engl.: statement) und *Deklarationen* (engl.: declarations). Anweisungen definieren die eigentlichen Aktionen eines Programms.

In Java wird im Kopf einer Methode auch ihre Aufrufbarkeit festgelegt. Die mit `public` deklarierten *öffentlichen Methoden* können als Dienstleistungen betrachtet werden, die durch *Klienten* aufrufbar sind, beispielsweise von Objekten anderer Klassen. Diese Methoden bilden die Schnittstelle, die in BlueJ auch interaktiv aufrufbar ist. Die mit `private` deklarierten *privaten Methoden* hingegen sind nur innerhalb der definierenden Klasse aufrufbar.

Die Signatur einer Methode legt u.a. ihre *Parameter* fest (genauer: deren Anzahl, Reihenfolge und jeweiligen Typ). In Java gibt es *primitive Typen* wie `int`, `float` und `boolean`, ein Typ kann aber auch ein *Objekttyp* sein (wie `String` oder `Kreis`). Wir unterscheiden *aktuelle Parameter* (inzwischen etablierte, aber falsche Übersetzung des engl.: actual parameter) von *formalen Parametern* (engl.: formal parameter): Die tatsächlichen Werte, die an den möglicherweise zahlreichen Aufrufstellen einer Methode angegeben werden, werden aktuelle Parameter genannt. Die formalen Parameter hingegen, die im Kopf einer Methode aufgeführt sind, haben Namen. Mit diesen Namen kann im Rumpf einer Methode auf die Parameter zugegriffen werden. Für die *Parameterübergabe* sagen wir auch, dass bei einem Methodenaufruf die aktuellen Parameter des Aufrufers an die formalen Parameter der aufgerufenen Methode *gebunden* werden. Als aktuelle Parameter können häufig *Literale* verwendet werden. Ein Literal ist eine Zeichenfolge (wie `132` oder `"gelb"`), die einen Wert repräsentiert und deren Struktur dem Compiler bekannt ist.

Variablen (engl.: variable) sind Speicherplätze für Werte. Jede Variable in Java muss *deklariert* werden, durch Angabe eines Namens (*Bezeichner,* engl.: identifier) und eines Typs. *Lokale Variablen* werden innerhalb einer Methode deklariert. Eine lokale Variable existiert nur während der Ausführung der Methode und ist nur innerhalb der Methode zugreifbar. Die formalen Parameter einer Methode sind ebenfalls Variablen (sie werden im Kopf der Methode deklariert). Man spricht von *Variableninitialisierung*, wenn einer Variablen erstmalig ein Wert zugewiesen wird. Bei einer lokalen Variablen geschieht dies in der Methode, in der sie definiert ist. Bei einem formalen Parameter geschieht dies über die Bindung der aktuellen Parameter beim Methodenaufruf.

In einem objektorientierten System werden alle Aktionen durch Methodenaufrufe ausgelöst. Die Methoden eines Objektes werden üblicherweise in der *Punktnotation* (engl.: dot notation) aufgerufen: Zuerst wird das aufzurufende Objekt benannt, dann, getrennt durch einen Punkt, die aufzurufende Methode. Dann folgen die aktuellen Parameter, falls die Signatur der Methode Parameter definiert. Innerhalb einer Klassendefinition können die Methoden der eigenen Klasse auch direkt aufgerufen werden (ohne Objektname und Punkt).

Lernziele

Klassendefinition (Java-Quelltext) verstehen, verändern und übersetzen können; Fehlermeldungen des Compilers verstehen; sequenzielle Ausführung verstehen; Methodenaufrufe programmieren können; Parameter und lokale Variablen verstehen und anwenden können; Teile einer Methode in eine neue Methode auslagern können.

Aufgabe 2.1 Erster Kontakt mit Quelltext

2.1.1 Öffnet das Projekt *Blatt02_Zeichnung*. Erzeugt ein Exemplar der Klasse `Zeichner` und ruft die Methode `zeichne` auf.

2.1.2 Jetzt wird es ernst: Öffnet den Quelltext der Klasse `Zeichner` durch einen Doppelklick auf die Klasse. Im Editor könnt ihr die Implementierung der Methode `public void zeichne()` erkennen. Beschreibt eurem Betreuer mit den Begriffen aus dem Einleitungstext, was ihr dort seht.

2.1.3 Nehmt ein beliebiges Semikolon weg und übersetzt die Klasse. Warum tritt ein Fehler auf und wie wird er angezeigt? **Schreibt dies mit wenigen Worten nieder.**

2.1.4 Verändert die Implementierung von `zeichne`, so dass ein anderes Bild gezeichnet wird. Dazu könnt ihr die Methoden verwenden, die ihr bisher interaktiv aufgerufen habt. Hinweis: Nach jeder Veränderung muss der Quelltext neu übersetzt werden; dabei werden die alten Exemplare gelöscht, weil die „Blaupause" zum Erzeugen neuer Exemplare geändert wurde.

2.1.5 Was ist eine Variablendeklaration, und aus welchen Teilen besteht sie? **Erklärt dies schriftlich** am Beispiel `String strasse;`.

Aufgabe 2.2 Verändern der Schnittstelle des Zeichners

2.2.1 Öffnet das Projekt *Zeichnung* erneut und speichert eine Kopie als *Zeichnung2*. Arbeitet von nun an mit dem Projekt *Zeichnung2*.

2.2.2 Die Farbe der Hauswand soll nun interaktiv bei einem Methodenaufruf angegeben werden können. Hierzu müsst ihr in der Methode `zeichne` einen Parameter einbauen, der den übergebenen Farbnamen speichert und den ihr an passender Stelle für das Setzen der Wandfarbe verwendet.

Ein Beispiel für eine Methode mit Parameter ist zum Beispiel `farbeAendern` in der Klasse `Kreis`.

Eine allgemeine Struktur einer Methode, die einen Parameter akzeptiert, sieht wie folgt aus:

```
/**
 * Methodenkommentar
 * @param <Parametername> Erklärung zum Paramter
 */
public void <Methodenname>(<Parametertyp> <Parametername>)
{ ...
}
```

Wer möchte, kann weitere Parameter definieren, beispielsweise für die Farben der Sonne und des Fensters.

2.2.3 Ihr habt durch Einführung des formalen Parameters die Schnittstelle der Klasse `Zeichner` verändert. Dokumentiert dies, indem ihr *Schnittstellenkommentare* schreibt (in der Klasse `Kreis` findet ihr Beispiele für dokumentierte Methoden mit Parametern). Überprüft, ob eure Kommentare bei einem interaktiven Aufruf erscheinen.

2.2.4 Was ist der Unterschied zwischen formalen und aktuellen Parametern? **Erläutert ihn schriftlich** mit euren eigenen Worten.

2.2.5 *Zusatzaufgabe*: Was versteht man unter dem Kopf einer Methode? Was ist die Signatur einer Methode? Worin besteht der Unterschied? Wofür benötigt man die Signatur einer Methode? **(schriftlich)**

Aufgabe 2.3 Vertraut werden mit BlueJ

2.3.1 Experimentiert mit der Entwicklungsumgebung BlueJ. Wie kann man z.B. im Editor die Zeilennummern anzeigen lassen? Was bedeutet es, wenn manche Klasse gestreift erscheinen? Wie lassen sich alle Klassen auf einmal übersetzen? Was ist der Unterschied zwischen „Übersetzen" und „Paket neu übersetzen"? Wie kann man sich die Projektdokumentation anzeigen lassen?

2.3.2 Im Editorfenster von BlueJ könnt ihr zwischen „*Implementierung* bzw. *Quelltext*" und „*Schnittstelle* bzw. *Dokumentation*" wählen (Auswahlbox rechts oben). Die Schnittstellensicht wird von einem Programm namens *javadoc* erzeugt, das den Quelltext analysiert und die relevanten Informationen extrahiert und in einer eigenen HTML-Darstellung aufbereitet. Javadoc ist Bestandteil des JDK (Java Development Kit) und nichts BlueJ-spezifisches. Welche Elemente der Klasse werden im Schnittstellen-Modus angezeigt und was ist verborgen?

2.3.3 In welchen Situationen benötigt man die Schnittstellensicht einer Klasse? Warum ist sie hilfreich?

Aufgabe 2.4 Verbessern der internen Struktur

2.4.1 Die Methode `zeichne` ist inzwischen vermutlich recht unübersichtlich und lang geworden. Sie soll jetzt übersichtlicher werden, indem ihr die Funktionalität auf weitere Methoden aufteilt. Definiert dazu Methoden wie `zeichneDach` und/oder `zeichneWand` und ruft diese von der zentralen `zeichne`-Methode aus auf. Ist es nötig, diese Methoden als `public` zu deklarieren? Wenn ja, warum? Wenn nein, weshalb wäre es dann sinnvoll, sie als `private` zu deklarieren?

2.4.2 Schreibt eine Methode `zeichneHaeuser`, die (neben der Sonne, eventuell einem Himmel etc.) drei oder mehr Häuser gleicher „Architektur" mit (möglichst) unterschiedlichen Wandfarben und an (möglichst) verschiedenen Stellen zeichnet. Versucht, durch die Aufteilung und die Übergabe von Parametern Arbeit zu sparen und redundante Aufrufe zu reduzieren.