

SE1, Aufgabenblatt 6

Softwareentwicklung I – Wintersemester 2015/16

Objekte benutzen Objekte

MIN-CommSy-URL: <https://www.mincommsy.uni-hamburg.de/>
CommSy-Projektraum SE1 CommSy WiSe 15/16
Ausgabewoche 19. November 2015

Kernbegriffe

Objekte können Methoden an anderen Objekten aufrufen. Das aufrufende Objekt wird als *Klient* bezeichnet, das aufgerufene als *Dienstleister*. Der Dienstleister definiert in den Signaturen seiner Methoden, welche Parameter für eine Methode akzeptiert werden, d.h. deren Anzahl, Reihenfolge und jeweiligen Typ.

In Java gibt es neben den primitiven Typen *Referenztypen*. Jede in Java geschriebene Klasse definiert einen Referenztyp. Eine Variable mit dem Typ einer Klasse (eine *Referenzvariable*, engl.: reference variable) enthält kein Objekt, sondern lediglich eine *Referenz* (engl.: reference) auf ein Objekt dieser Klasse. Über den Variablenbezeichner können mit der *Punktnotation* die Methoden des referenzierten Objekts aufgerufen werden.

Eine Referenzvariable, die auf kein Objekt verweist, hält den Wert des speziellen Literals `null`. Wird versucht, über eine solche `null`-Referenz eine Methode aufzurufen, dann wird die Programmausführung mit einer so genannten *Ausnahme* (engl.: exception), hier einer `NullPointerException`, abgebrochen. Eine Ausnahme wird immer dann geworfen, wenn während der Ausführung eines Programms ein Fehler festgestellt wird.

In Java können mehrere Methoden einer Klasse den gleichen Namen haben, solange sich ihre Signaturen unterscheiden. Wird auf diese Weise ein Name für mehrere Methoden verwendet, so bezeichnet man den Methodennamen als *überladen* (engl.: overloaded). Gleichnamige Methoden sollten auch eine möglichst ähnliche Semantik haben. Nach den gleichen Regeln kann eine Klasse mehrere Konstruktoren definieren.

Die *Unified Modeling Language (UML)* ist eine grafische Sprache zur Beschreibung von Softwaresystemen. Die Notation der UML umfasst Diagramme für die Darstellung verschiedener Ansichten auf ein System. Unter anderem gibt es *Klassendiagramme* (engl.: class diagrams) und *Objektdiagramme* (engl. object diagrams). Ein Klassendiagramm beschreibt die statische Struktur eines Systems, indem es die Beziehungen zwischen seinen Klassen darstellt. Ein Objektdiagramm gibt Antwort auf die Frage, welche Struktur ein Teil des Systems zu einem bestimmten Zeitpunkt während der Laufzeit besitzt („Schnappschuss“); es zeigt üblicherweise Objekte mit den Belegungen ihrer Felder (Werte primitiver Typen oder Referenzen).


Lernziele


Sicheres Umgehen mit Objektreferenzen; Verstehen von möglichen Fehlersituationen und Ausnahmen; Verstehen und Zeichnen einfacher UML-Diagramme

Aufgabe 6.1 Überweisungsmanager

6.1.1 erinnert ihr euch an die Klasse `Konto`? Diese Woche gibt es wieder eine Kontoklasse. Diese findet ihr in der Vorlage *Ueberweisungsmanager* im CommSy-Projektraum. Fügt dem Projekt nun eine neue Klasse *Ueberweisungsmanager* hinzu. Ein Exemplar dieser Klasse soll einen Betrag von einem Konto auf ein anderes überweisen können. Dazu soll die Klasse eine Methode `ueberweisen(Konto quellKonto, Konto zielKonto, int betrag)` definieren. Implementiert in ihr das gewünschte Verhalten.


Um bei einem interaktiven Methodenaufruf Objekte zu übergeben, müssen diese zuvor erzeugt worden sein. Per Klick auf die Referenzen in der Objektleiste können diese leicht in den BlueJ-Dialog übertragen werden.

 6.1.2 Testet euren Überweisungsmanager interaktiv mit BlueJ. Schaut euch dabei mit dem Objekt-Inspektor die internen Zustände eurer Konten an. Was passiert, wenn ihr anstelle eines Kontos einfach `null` eingibt? Wie könnt ihr diesen Fehler verhindern? **Haltet eure Erkenntnisse schriftlich fest.**


 6.1.3 Was ist die Punktnotation, wofür wird sie verwendet, wie ist sie aufgebaut? **Schriftlich.**

Aufgabe 6.2 Uhrenanzeige

6.2.1 Öffnet das BlueJ-Projekt *Uhrenanzeige*. Darin findet ihr eine (unvollständige und deshalb nicht übersetzbare) Klasse `Nummernanzeige`, die ihr im Editor öffnen und euch erarbeiten sollt. Lest insbesondere den Klassenkommentar sehr genau durch, um die gewünschte Funktionalität der Klasse zu verstehen. Alle Methodenrumpfe der Klasse sind leer. Implementiert nun alle Methoden im Sinne der Kommentare. Testet eure fertige Klasse gründlich!

- 6.2.2 Fügt nun eine Klasse `Uhrenanzeige` hinzu, die die Anzeige einer Digitaluhr mit Stunden und Minuten implementiert. In der Implementierung der Klasse `Uhrenanzeige` sollt ihr die Klasse `Nummernanzeige` verwenden (ohne sie zu verändern und ohne ihren Quelltext zu kopieren). Das bedeutet, dass ihr in der `Uhrenanzeige` Exemplare der `Nummernanzeige` erzeugt und dass ihr mit der Punktnotation Methoden der `Nummernanzeige` aufruft. Implementiert in der `Uhrenanzeige` die Methoden `gibUhrzeitAlsString`, `setzeUhrzeit` und `taktsignalGeben`. Letztere soll die `Uhrenanzeige` um eine Minute weiterschalten (die `Uhrenanzeige` wird interaktiv weitergeschaltet, sie ist keine selbstständig laufende Uhr).
- 6.2.3 Erweitert die Klasse `Uhrenanzeige` um einen Konstruktor, mit dem die gewünschte Uhrzeit gleich beim Erzeugen eines Exemplars dieser Klasse mit angegeben werden kann. Wieso könnt ihr mehr als einen Konstruktor haben? Wie nennt man das dahinter liegende Konzept? Könnt ihr ein Beispiel finden, bei dem es auch für Methoden sinnvoll ist?
-  6.2.4 Erstellt ein UML-Objektdiagramm (Methoden sind optional) zu einer `Uhrenanzeige` (auf Papier).

Aufgabe 6.3 Animierte Analog-Uhr

- 6.3.1 Öffnet das Projekt `AnalogUhr` und erzeugt ein Exemplar der Klasse `AnalogUhr`. Daraufhin öffnet sich ein neues Fenster mit einer optisch ansprechenden, animierten Analog-Uhr, die um Mitternacht startet.
-  6.3.2 Erstellt ausgehend von einem Exemplar der Klasse `AnalogUhr` ein UML-Objektdiagramm (ohne Methoden), um euch zu verdeutlichen, wie die Objekte miteinander verbunden sind. Für das Diagramm sind nur die ersten vier Zustandsfelder innerhalb von `AnalogUhr` relevant. Die restlichen Zustandsfelder verweisen auf Exemplare von Klassen aus Javas Standardbibliothek (`Graphics2D`, `JPanel` und `JFrame`).
- 6.3.3 Diskutiert basierend auf dem Objektdiagramm folgende Fragen mit eurem Betreuer:
- Welchen Sinn hat das `Zeitmesser`-Objekt? Was bedeuten die beiden `long`-Zustandsfelder?
 - Welchen Sinn haben die Zustandsfelder `_radius1` und `_radius2`, und warum sind die überall anders?
 - Welchen Wert hat `_grob._zeitmesser`? Warum ist dieser spezielle Wert hier sinnvoll?
 - Ist es problematisch, dass der grobe Divisor den Wert 0 hat? Durch 0 dividieren ist ja keine gute Idee...
- 6.3.4 Bisher gibt es nur einen Sekundenzeiger und einen Minutenzeiger. Baut einen Stundenzeiger ein!
- 6.3.5 Bisher sind auf dem Ziffernblatt nur die groben Stundenmarkierungen zu sehen. Ergänzt das Ziffernblatt um jeweils 4 feine Minutenstriche nach jedem Stundenstrich!

Aufgabe 6.4 Wie spät ist es eigentlich?

Sicher ist euch aufgefallen, dass die analoge Uhr immer um Mitternacht startet. Recherchiert eigenständig, wie man in Java die Sekunden seit Mitternacht berechnet und baut den Zeitmesser entsprechend um.