SE1, Aufgabenblatt 5

Softwareentwicklung I – Wintersemester 2011/12

Objekte benutzen Objekte

MIN-CommSy-URL: ..... https://www.mincommsy.uni-hamburg.de/

CommSy-Projektraum ..... SE1 CommSy WiSe 11/12

Ausgabewoche ..... 17. November 2011

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 5.1 Überweisungsmanager

5.1.1. Erinnert ihr euch an die Klasse `Konto`? Diese Woche gibt es wieder eine Kontoklasse. Diese findet ihr in der Vorlage *Ueberweisungsmanager* im CommSy-Projekraum. 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.

5.1.2. 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` eingebt? Wie könnt ihr diesen Fehler verhindern? **Haltet eure Erkenntnisse schriftlich fest.**

5.1.3 *Zusatzaufgabe:* Was ist die Punktnotation, wofür wird sie verwendet, wie ist sie aufgebaut?

Aufgabe 5.2 Tic Tac Toe

Kurz bevor eine junge Softwarefirma die Arbeiten an der Umsetzung des Spiels *Tic Tac Toe* beenden konnte, ist sie leider pleite gegangen. Ihr wurdet nun mit der Aufgabe betraut, das Spiel fertig zu stellen. Zur Erläuterung: Bei Tic Tac Toe spielen zwei Spieler auf einem Spielfeld der Größe 3x3 = 9 Positionen. Die Spieler besetzen abwechselnd eine freie Position (Spieler 1 mit einem X, Spieler 2 mit einem O), bis ein Spieler drei Positionen in einer Reihe besetzt hat (horizontal, vertikal oder diagonal) und gewinnt, oder das Spielfeld voll ist.

Ihr müsst lediglich die Klasse `TicTacToe` bearbeiten, um das Spiel fertig zu stellen, alle anderen Klassen sind bereits fertig implementiert.

5.2.1 Öffnet das Projekt *TicTacToe* und erzeugt ein Exemplar der Klasse `TicTacToeGUI`. Es öffnet sich ein Fenster mit einer grafischen Darstellung des Spielfelds. Ein Klicken auf die Knöpfe hat noch keine sichtbare Wirkung. Schaut euch den Quelltext der Klasse `TicTacToe` an, um einen Überblick zu bekommen. Lest die Schnittstellenkommentare der Methoden und versucht, den Zweck der Methoden zu verstehen.

5.2.2 Euch ist möglicherweise aufgefallen, dass immer derselbe Spieler dran ist (in der Methode `gibAktuellenSpieler`). Das soll als erstes geändert werden.

Führt eine weitere Exemplarvariable ein, die sich merkt, wer der aktuelle Spieler ist. Initialisiert diese geeignet im Konstruktor und passt `gibAktuellenSpieler` entsprechend an. Implementiert auch die private, bisher leere Methode `wechsleSpieler`.

Testet eure Änderungen, indem ihr wieder ein Exemplar der Klasse `TicTacToeGUI` erstellt und euch davon überzeugt, dass durch einen Klick der aktuelle Spieler in der Anzeige gewechselt wird. Die Position wird dabei natürlich noch nicht belegt, denn das haben wir ja noch nicht implementiert.

5.2.3 Als nächstes wollen wir erreichen, dass die Spieler Positionen auf dem Spielfeld besetzen können. Schaut euch dazu die Methode `besetzePosition` an. In dieser Methode soll die komplette Spiellogik stattfinden. Nachdem sichergestellt wird, dass das Spiel noch nicht zu Ende und die gewünschte Position noch frei ist, fehlt eine Anweisung, die dafür sorgt, dass die Position besetzt wird. Dazu müsst ihr die gleichnamige Methode `besetzePosition` an dem Objekt der Klasse `Spielfeld` aufrufen.

Aus Klient-und-Dienstleister-Sicht spricht man hier von *Delegation*: wir stellen einen Dienst (Position besetzen) zur Verfügung, indem wir die Anfrage des Klienten an einen anderen Dienstleister (das Spielfeld) weiterleiten (*delegieren*), der diese Aufgabe für uns erledigt.

Testet wieder anhand der GUI, ob jetzt Positionen besetzt werden können.

5.2.4 Identifiziert eine weitere Methode, die Delegation benutzt.

5.2.5 Schaut euch die Methode `istSpielZuEnde` an. Solange `gibGewinner` den Wert -1 zurückliefert, gilt das Spiel als noch nicht beendet. Ein Blick in den Rumpf von `gibGewinner` offenbart, dass immer -1 zurückgeliefert wird. Deswegen lässt uns die GUI auch dann noch weiter spielen, wenn das Spielfeld bereits voll ist.

Ihr braucht offenbar eine weitere Exemplarvariable, die sich merkt, wer das Spiel gewonnen hat bzw. ob es überhaupt noch läuft. Initialisiert diese neue Exemplarvariable geeignet im Konstruktor und gebt ihren Wert in `gibGewinner` zurück.

In der Methode `besetzePosition` müsst ihr nun noch eine Anweisung einfügen, die ausgeführt wird, wenn das Spielfeld voll ist. Tipp: Lest den Schnittstellenkommentar der Methode `gibGewinner`. Testet eure Änderungen wieder mit der GUI. Jetzt sollte nach neun Klicks eine Meldung kommen, dass das Spiel unentschieden ausgegangen ist.

5.2.6 In `besetzePosition` muss jetzt nur noch darauf reagiert werden, wenn ein Spieler gewonnen hat. Fügt hier eine Anweisung hinzu, die den aktuellen Spieler zum Gewinner erklärt.

Das eigentliche Erkennen der Gewinnsituation ist noch nicht implementiert. Für das Erkennen von drei Positionen in einer Reihe, die der aktuelle Spieler besetzt hat, ist aber bereits eine leere Methode `aktuellerSpielerBesitzt` vorgegeben, die ihr sinnvoll implementieren sollt.

Nun müsst ihr in der Methode `hatAktuellerSpielerGewonnen` nur noch alle möglichen Reihen durchgehen, die zu einem Sieg führen. Fangt zunächst mit nur einer einzigen Reihe an, z.B. mit der obersten Zeile. Testet wieder mit der GUI und erweitert die Methode sukzessive, bis ihr alle acht Reihen (drei horizontale, drei vertikale und zwei diagonale) berücksichtigt. Testet das fertige Spiel.

5.2.7 Erstellt ein Objektdiagramm eines laufenden Tic-Tac-Toe-Spiels (auf Papier, ohne die GUI-Klasse). Grenzt die beiden Begriffe „Objektdiagramm“ und „Klassendiagramm“ **schriftlich** gegeneinander ab.

Aufgabe 5.3 Uhrenanzeige

5.3.1 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 Methodenrümpfe der Klasse sind leer. Implementiert nun alle Methoden im Sinne der Kommentare. Testet eure fertige Klasse gründlich!

5.3.2 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 `gibUhrzeit`, `setzeUhrzeit` und `taktsignalGeben`. Letztere soll die `Uhrenanzeige` um eine Minute weiterschalten (die `Uhrenanzeige` soll interaktiv weitergeschaltet werden können, sie ist keine selbstständig laufende Uhr).

5.3.3 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?

5.3.4 Erstellt ein Objektdiagramm zu einer `Uhrenanzeige` (auf Papier).