3. Programmieraufgabe

Objektorientierte Programmiertechniken

LVA-Nr. 185.A01
2017/2018 W
TU Wien

Themen:

Zusicherungen, Analyse des eigenen Programms

Ausgabe:

25. 10. 2017

Abgabe (Deadline):

08. 11. 2017, 12:00 Uhr

Abgabeverzeichnis:

`Aufgabe1-3`

Programmaufruf:

`java Test`

Grundlage:

Skriptum, Schwerpunkt auf Abschnitt 2.2

Welche Aufgabe zu lösen ist

Versehen Sie den Programmtext Ihrer Lösung der 2. Programmieraufgabe (inklusive Testprogramm) mit Kommentaren, die Zusicherungen in natürlicher Sprache (Deutsch oder Englisch) darstellen. Entfernen Sie alle Kommentare, die keine Zusicherungen ausdrücken. Nur die weiter unten beschriebenen speziellen Kommentare sowie organisatorische Kommentare (etwa die Namen der Autoren) dürfen danach neben Zusicherungen vorkommen. Lassen Sie organisatorische Kommentare, die nicht sowieso ganz klar als solche zu erkennen sind, mit `ANMERKUNG:` oder `NOTE:` beginnen. Alle notwendigen Zusicherungen sollen explizit im Programmcode stehen. Abhängigkeiten zwischen Programmteilen sollen durch Zusicherungen aber nicht unnötig verstärkt werden.

Vergewissern Sie sich, dass Clients nur das voraussetzen, was Server durch Zusicherungen versprechen, und Server nur das, was Clients versprechen, also alle Verträge in Ihrer Software eingehalten werden. Prüfen Sie nach, ob Zusicherungen überall dort, wo Sie eine Vererbungsbeziehung haben (durch `extends` oder `implements`), die Bedingungen für die Ersetzbarkeit erfüllen. Falls Sie Inkonsistenzen bemerken, die sich durch andere Formulierungen der Zusicherungen nicht einfach beheben lassen – das heißt, Sie haben inhaltliche Fehler entdeckt – beschreiben Sie die Inkonsistenzen durch spezielle Kommentare, die mit `FEHLER:` oder `ERROR:` beginnen. Versuchen Sie, in wenigen Worten mögliche Ursachen sowie nötige Programmänderungen zur Fehlerkorrektur zu beschreiben. Lassen Sie den Programmtext selbst (abgesehen von Kommentaren) unverändert.

Finden Sie Stellen in Ihrem Programm, an denen

- der Klassenzusammenhalt besonders hoch ist und Sie sich andere einfache Lösungsvarianten mit deutlich niedrigerem Klassenzusammenhalt vorstellen können,
- oder die Objektkopplung besonders schwach ist und Sie sich andere einfache Lösungsvarianten mit deutlich stärkerer Objektkopplung vorstellen können,
- oder die Verwendung von dynamischem Binden den Programmcode besonders stark vereinfacht und die Wartbarkeit positiv beeinflusst.

Schreiben Sie an jede dieser Stellen einen Kommentar, der mit `GUT:` oder `GOOD:` beginnt. Beschreiben Sie darin kurz, warum Sie diese Stelle gewählt und welche Überlegungen zur guten Lösung geführt haben. Falls Sie nicht mindestens fünf solche Stellen finden, schreiben Sie an den Anfang der Datei `Test.java` einen Kommentar, in dem Sie wahrscheinliche Gründe dafür nennen.

mindestens 5 solche Kommentare

Finden Sie Stellen in Ihrem Programm, an denen Klassenzusammenhalt oder Objektkopplung besonders schlecht sind oder durch Verzicht auf dynamisches Binden der Code größer oder unübersichtlicher wurde. Schreiben Sie an jede dieser Stellen einen Kommentar, der mit `SCHLECHT:`

oder `BAD:` beginnt, und beschreiben Sie kurz, warum Sie diese Stelle gewählt haben. Beschreiben Sie auch kurz, wie diese Schwachstelle Ihrer Meinung nach zustandegekommen ist und wie eine bessere Lösung aussehen könnte. Falls Sie nicht mindestens fünf solche Stellen finden, schreiben Sie an den Anfang der Datei `Test.java` einen Kommentar, in dem Sie wahrscheinliche Gründe dafür nennen.

mindestens 5 solche Kommentare

Die Klasse `Test.java` soll als Kommentar eine kurze, aber verständliche Beschreibung der Aufteilung der Arbeiten auf die einzelnen Gruppenmitglieder enthalten (nicht nur für frühere Aufgaben, sondern auch für Aufgabe 3).

Wie die Aufgabe zu lösen ist

nur Kommentare ändern

Lassen Sie den Programmcode (abgesehen von Kommentaren) unverändert. Schreiben Sie nicht nur Kommentare wie sie Ihnen in den Sinn kommen, sondern beschreiben Sie über Vorbedingungen, Nachbedingungen, Invarianten und History-Constraints bewusst einhaltbare Verträge zwischen Client und Server. Die Verträge sollen alles enthalten, was Client und Server voneinander wissen müssen. Andererseits sollen Verträge unnötige Bedingungen vermeiden um die Abhängigkeit zwischen Client und Server nicht künstlich zu erhöhen. Im Idealfall spiegeln Verträge Überlegungen wider, die Sie bei der Programmerstellung im Kopf hatten. Wenn die ursprünglichen Überlegungen nach Änderungen nicht mehr zutreffen, sollen Verträge den endgültigen Stand darstellen.

Obwohl nicht verlangt, empfiehlt es sich, bei jeder Zusicherung zu vermerken um welche Art von Zusicherung es sich handelt. Das kann Vorteile bei der Überprüfung der Zusicherungen bringen.

Es kann passieren, dass ein Client vom Server etwas anderes erwartet, als dieser bietet, oder umgekehrt. Das ist ein Fehler im Programm, der entweder sofort oder erst nach späteren (dem Vertrag entsprechenden) Programmänderungen zu falschen Ergebnissen führt. Versuchen Sie nicht, solche Fehler durch Tricks in den Verträgen zu beseitigen. Oft ist es möglich, durch komplizierte Formulierungen die Zusicherungen so hinzubiegen, dass das Programm noch korrekt erscheint. Das geht aber fast immer nur auf Kosten der Brauchbarkeit und Wartbarkeit, weil im Vertrag für den Endanwender sinnlose Details und Eigenschaften festgeschrieben werden. Schreiben Sie lieber mit `FEHLER:` oder `ERROR:` beginnende Kommentare statt über Tricks Verträge zurechtzubiegen. Solche Kommentare haben garantiert keine Auswirkungen auf Ihre Beurteilung, können Ihnen aber helfen, für Sie typische Fehler, die Sie unter Zeitdruck sicherlich machen, besser zu verstehen.

Verträge einfach halten

Es ist empfehlenswert, dass Zusicherungen von der Person geschrieben werden, die auch den entsprechenden Programmcode (aus Sicht des Servers) geschrieben und die entsprechenden Überlegungen vielleicht noch im Kopf hat. Fehler werden aber eher von anderen Personen erkannt, die den Code aus der Sicht eines Clients betrachten. Am wahrscheinlichsten sind Fehler an Schnittstellen, an denen der Code eines Servers und eines Clients von verschiedenen Personen stammt. Diese Stellen sind besonders sorgfältig zu überprüfen.

Auch in Untertypbeziehungen treten häufig Fehler bei den Zusicherungen auf. Vergewissern Sie sich, dass im Untertyp Vorbedingungen nur

schwächer und Nachbedingungen sowie Invarianten stärker sein dürfen als im Obertyp und auch History-Constraints ähnliche, aber anders formulierte Bedingungen erfüllen müssen. Entsprechende Fehler sind ebenso mit `FEHLER:` oder `ERROR:` zu kennzeichnen.

Durch Zusicherungen festgelegte Verträge sind ein Qualitätsmerkmal, das hilft, die Korrektheit bei Ersetzung einzelner Programmteile zu erhalten. Weitere Qualitätsmerkmale können bei der Wartung ebenso hilfreich sein. Markieren Sie alle Stellen, die Ihnen bei der Suche nach passenden Verträgen als besonders gut oder schlecht auffallen, durch mit `GUT:` oder `SCHLECHT:` bzw. `GOOD:` oder `BAD:` beginnende Kommentare.

Falls Ihre Lösung der zweiten Aufgabe noch so unvollständig ist, dass Sie es nicht für sinnvoll erachten, die Lösung der dritten Aufgabe darauf aufzubauen, setzen Sie sich bitte mit Ihrer Tutorin oder Ihrem Tutor in Kontakt und befolgen Sie die von ihr oder ihm gegebenen Anweisungen. Im Zweifelsfall lösen Sie die Aufgabe so wie oben beschrieben.

Warum die Aufgabe diese Form hat

Programmieren ist mehr als nur das Schreiben von Code. Diese Aufgabe soll Ihnen durch eine festgelegte Vorgehensweise helfen, die Qualität Ihrer eigenen Programme vor allem hinsichtlich Wartbarkeit und Wiederverwendbarkeit abzuschätzen. Sie sollen ein Gefühl für Ihre eigenen Stärken und Schwächen bekommen.

Das Schreiben und Überprüfen von Zusicherungen, die Berücksichtigung von Zusicherungen in Untertypbeziehungen, sowie die Abschätzung von Klassenzusammenhalt und Objektkopplung sind Lernziele, die in dieser Aufgabe geübt werden. Ein wichtiger Aspekt dabei ist die Eigenständigkeit: Sie sollen Verträge in der Software aufgrund eigener Überlegungen entwickeln (jenen Überlegungen, die Sie im Idealfall bereits beim Schreiben des Programmcodes angestellt haben) statt anhand von Vorgaben oder Beispielsammlungen. Das ist zwar ein schwieriger Weg, aber einer, der Ihnen hilft, auch in außergewöhnlichen Situationen zurechtzukommen.