3. Programmieraufgabe

Programmierparadigmen

LVA-Nr. 194.023
2024/2025 W
TU Wien

Themen:

Zusicherungen,
Paradigmenübersicht,
Analyse des eigenen Programms

Ausgabe:

04. 11. 2024

Abgabe (Deadline):

11. 11. 2024, 14:00 Uhr

Abgabeverzeichnis:

`Aufgabe1-3`

Programmaufruf:

`java Test`

Grundlage:

Skriptum,
Kapitel 2 und
Abschnitte 3.1 und 3.2

Welche Aufgabe zu lösen ist

Ihre Lösung der 2. Programmieraufgabe ist um Folgendes zu erweitern:

- Zusicherungen und Analyse des Programms,
- einen funktionalen Teil,
- einen parallelen oder nebenläufigen Teil.

Zusicherungen und Analyse: Falls noch nicht geschehen, kennzeichnen Sie alle Teile Ihrer Lösung der 2. Aufgabe so, dass klar ist, welcher Teil als prozedural und welcher als objektorientiert betrachtet wird.

Versehen Sie alle objektorientierten Programmteile mit Kommentaren, die Zusicherungen entsprechend Design-by-Contract in natürlicher Sprache (deutsch oder englisch) darstellen. Falls nötig, schreiben Sie vorhandene Kommentare so um, dass sie als Zusicherungen lesbar werden. Organisatorische Kommentare (Namen der Autoren, Kommentare beginnend mit `STYLE:`, ...) bleiben davon unberührt. Lassen Sie organisatorische Kommentare, die nicht klar als solche zu erkennen sind, mit `NOTE:` beginnen. Alle notwendigen Zusicherungen sollen explizit im Programmcode stehen. Abhängigkeiten zwischen Programmteilen sollen durch Zusicherungen nicht unnötig verstärkt werden.

Vergewissern Sie sich, dass Clients im objektorientierten Teil nur das voraussetzen, was Server durch Zusicherungen versprechen, und Server nur das, was Clients versprechen, also alle Verträge 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 Kommentare, die mit `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 mindestens drei Stellen im objektorientierten Teil Ihres Programms, an denen

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

mind. 3 `"GOOD: ..."`
im objektorientierten Teil

Schreiben Sie an jede dieser Stellen einen Kommentar, der mit `GOOD:` beginnt. Beschreiben Sie darin kurz, warum Sie diese Stelle gewählt und welche Überlegungen zur guten Lösung geführt haben.

Finden Sie mindestens drei Stellen im objektorientierten Teil, an denen der Klassenzusammenhalt oder die Objektkopplung nicht ideal ist oder durch Verzicht auf dynamisches Binden der Code unübersichtlicher wurde. Schreiben Sie an jede dieser Stellen einen Kommentar, der mit `BAD:` beginnt und beschreiben Sie kurz, warum Sie diese Stelle gewählt haben. Beschreiben Sie auch kurz, wie diese Schwachstelle zustandegekommen sein und wie eine bessere Lösung aussehen könnte.

mind. 3 `"BAD: ..."` im objektorientierten Teil

Finden Sie analog dazu je mindestens drei Stellen im prozeduralen Teil, die gut bzw. nicht gut gelungen sind und markieren Sie sie mit je einem beschreibenden Kommentar, der mit `GOOD:` bzw. `BAD:` beginnt. Gut gelungen sind Stellen, an denen der Kontrollfluss leicht nachvollziehbar ist oder globale Variablen sowie schwer durchschaubare Aliase vermieden werden (und dies nicht automatisch aus den natürlichen Gegebenheiten folgt). Schlecht gelungen ist genau das Gegenteil.

mind. 3 `"GOOD: ..."` und mind. 3 `"BAD: ..."` im prozeduralen Teil

Funktionaler Teil: Erweitern Sie Ihr Programm um neue Funktionalität entsprechend der 2. Aufgabe (oder eine neue, alternative Implementierung bestehender Funktionalität) in einem funktionalen Stil. Binden Sie die neuen Programmteile so ein, dass Sie bei einem Aufruf von `java Test` ausgeführt werden. Der Umfang der neuen Programmteile ist nicht genau bestimmt, soll aber ausreichen, um Erfahrungen in der funktionalen Programmierung zu sammeln. Etwa hundert Zeilen sind eine Untergrenze. Wählen Sie selbst, welchen funktionalen Programmierstil Sie verwenden, rekursive Funktionen oder einen applikativen Stil basierend auf Java-8-Streams (eventuell in Kombination). Achten Sie bitte auf referentielle Transparenz. Versuchen Sie, die Möglichkeiten der funktionalen Programmierung zu nutzen, das heißt, algorithmisch eher komplexe Programmteile möglichst kompakt und dennoch in hoher Qualität bereitzustellen. Verwenden Sie bitte auch (selbst geschriebene oder bereits im System vorhandene) Funktionen höherer Ordnung. Setzen Sie gezielt Lambdas ein. Kennzeichnen Sie alle neuen funktionalen Programmteile durch Kommentare beginnend mit `STYLE:` als solche. Diese Kommentare sollen auch beschreiben, wie die funktionalen Teile mit dem Rest des Programms zusammenarbeiten, sodass referentielle Transparenz gewahrt bleibt.

Paralleler oder nebenläufiger Teil: Erweitern Sie Ihr Programm um neue Funktionalität entsprechend der 2. Aufgabe (oder eine neue, alternative Implementierung bestehender Funktionalität), wobei entweder parallele oder nebenläufige Programmierung zum Einsatz kommt. Wählen Sie die Funktionalität so, dass sie mit den Zielen des gewählten Paradigmas gut zusammenpasst und achten Sie darauf, diese Ziele möglichst gut zu erreichen. Sorgen Sie dafür, dass entsprechende Programmteile bei einem Aufruf von `java Test` ausgeführt werden. Der Umfang der neuen Programmteile soll ausreichend groß sein, um Erfahrungen mit dem Thema zu sammeln (100 Zeilen Minimum). Kennzeichnen Sie alle neuen Programmteile durch Kommentare beginnend mit `STYLE:` als solche. Diese Kommentare sollen auch beschreiben, welche konkreten Ziele mit dem Einsatz des parallelen oder nebenläufigen Paradigmas erreicht werden sollen, auf welche Weise das gelingt und wie der Rest des Programms diese Ziele unterstützt oder ihnen im Weg steht.

Aufgabenaufteilung: Die Datei `Test.java` soll am Anfang als Kommentar eine kurze, aber verständliche Beschreibung der Aufteilung der Arbeiten auf die einzelnen Gruppenmitglieder enthalten, nicht nur für frühere Aufgaben, sondern auch für diese. Nicht jedes Gruppenmitglied muss an jedem Aufgabenteil mitarbeiten, obwohl das vorteilhaft wäre.

Wie die Aufgabe zu lösen ist

nur Kommentare ändern

Lassen Sie den Programmcode aus der 2. Aufgabe (abgesehen von Kommentaren) so weit es geht unverändert, fügen Sie nur neue funktionale und parallele oder nebenläufige Teile hinzu. Schreiben Sie nicht nur Kommentare wie sie Ihnen in den Sinn kommen, sondern beschreiben Sie im objektorientierten Teil mittels Vorbedingungen, Nachbedingungen, Invarianten und History-Constraints bewusst einhaltbare Verträge zwischen Client und Server. Festlegungen des Kontexts in der Modularisierungseinheit helfen bei der Formulierung der Zusicherungen. Verträge sollen alles enthalten, was Client und Server voneinander wissen müssen. Unnötige Bedingungen sind zu vermeiden, um Abhängigkeiten schwach zu halten. Meist spiegeln Verträge Überlegungen wider, die Sie bei der Programmerstellung im Kopf hatten. Wenn ursprüngliche Überlegungen nach Änderungen nicht mehr zutreffen, sollen Verträge den endgültigen Stand darstellen. Programmteile in anderen Paradigmen haben andere Arten von Kommentaren. In Prozeduren verdeutlichen Kommentare den Kontrollfluss, in applikativen Teilen die dahinter stehende Idee, auf reinen Funktionen häufig kontextunabhängige Vor- und Nachbedingungen.

objektorientiert: Kommentare sind Zusicherungen

andere Paradigmen, andere Form von Kommentaren

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, 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 korrekt erscheint. Das geht aber auf Kosten der Brauchbarkeit und Wartbarkeit, weil im Vertrag sinnlose Details festgeschrieben werden. Schreiben Sie lieber mit `ERROR:` beginnende Kommentare. Solche Kommentare wirken sich nicht auf Ihre Beurteilung aus, helfen Ihnen aber, für Sie typische Fehler, die Sie unter Zeitdruck machen, besser zu verstehen.

Verträge einfach halten

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

Auch bei Untertypbeziehungen treten häufig Fehler in Zusicherungen auf. Vergewissern Sie sich, dass im Untertyp Vorbedingungen nur schwächer und Nachbedingungen sowie Invarianten stärker sind als im Obertyp und auch History-Constraints ähnliche, aber anders formulierte Bedingungen erfüllen. Fehler sind mit `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 gut oder schlecht auffallen, durch mit `GOOD:` oder `BAD:` beginnende Kommentare. Das Gleiche gilt für prozedurale Programmteile, wobei die Qualitätskriterien jedoch andere sind. Sie müssen mindestens je drei Stellen (insgesamt also mindestens 12) so kennzeichnen. Wenn Sie mit bestimmten Suchkriterien keine passenden Stellen finden, müssen Sie Ihre Suchkriterien entsprechend anpassen.

Erweiterungen um funktionale und parallele oder nebenläufige Programmteile werden idealerweise so gewählt, dass möglichst geringe Abhängigkeiten von bestehendem Code entstehen. Jedenfalls sollten vorher entsprechende Abschnitte im Skriptum gelesen werden, damit klar ist, welche Eigenschaften die neuen Programmteile haben sollen.

Abhängigkeiten vermeiden

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 Verbindung und befolgen Sie entsprechende 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 ebenso geübt werden, wie das Programmieren in für Sie vielleicht ungewohnten Paradigmen. Ein wichtiger Aspekt dabei ist die Eigenständigkeit: Sie sollen Softwareverträge und Programmierstile aufgrund eigener Überlegungen entwickeln statt anhand von Vorgaben oder Beispielsammlungen. Das ist zwar ein schwieriger Weg, aber einer, der Ihnen hilft, auch in außergewöhnlichen Situationen zurechtzukommen.

Erweiterungen um funktionale und parallele oder nebenläufige Programmteile haben natürlich zum Ziel, entsprechende Programmiertechniken zu üben und erste Anhaltspunkte dafür zu bekommen, wie sich das Programmieren in diesen Paradigmen „anfühlt“. Diese Teile der Aufgabe sind bewusst sehr offen formuliert. Sie sollen nicht nur den Umgang mit den Techniken lernen, sondern auch selbst herausfinden, wofür und auf welche Weise sich ein Einsatz lohnen könnte.

Aufgabe 3 ist die letzte Aufgabe, die zur Einstiegsphase zählt. Hier können Sie noch einmal Neues ausprobieren. Tutor_innen werden Sie gegebenenfalls auf schwere Fehler aufmerksam machen und Nachbesserungen verlangen. Diese Gelegenheit sollen Sie frühzeitig nutzen, um den Umgang mit neuen, schwierigen Konzepten zu üben. Ab der nächsten Aufgabe werden Unzulänglichkeiten ohne Korrekturmöglichkeit zu Punkteabzügen führen.