Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik III

Übungen zur Vorlesung Softwaretechnologie

- Wintersemester 2019/20 -
Dr. Günter Kniesel

Übungsblatt 1

Zu bearbeiten bis: 11.10.2019, 16 Uhr

Fangen Sie frühzeitig mit der Bearbeitung des Zettels an, damit wir Ihnen bei Bedarf helfen können!

Fragen richten Sie bitte an swt-tutoren@lists.iai.uni-bonn.de oder swt-vorlesung@lists.iai.uni-bonn.de. Über die erste Liste erreichen Sie die Tutoren und Herrn Dr. Kniesel, über die Zweite die Tutoren und Ihre Kommilitonen. Fragen an Herrn Dr. Kniesel senden Sie bitte *nur* über die Tutoren-Liste, nicht an ihn direkt, denn so haben Sie viel bessere Chancen auf eine schnelle Antwort. Das meiste können die Tutoren selbst unmittelbar beantworten und ansonsten können sie in der wöchentlichen Teambesprechung auf offene Fragen hinweisen.

Praktische Aufgaben

Diese Aufgaben sollen von allen Gruppenmitgliedern einzeln bearbeitet werden, da sie Sie in die Arbeit mit Git einführen, Sie die benötigte Software installieren sollen und weitere organisatorische Aspekte behandelt werden.
Die Lösungen zu den praktischen Aufgaben dieses Blattes sollten Sie selbstständig erarbeiten und bei Problemen mit Ihrem Tutor lösen.

Aufgabe 1. *Getting started* **8 Punkte**

a) **(1 Punkt)** Melden Sie sich **bis zum 08.10.2019, 23:59** via TVS zu den Tutorien an. Siehe Hinweise unter http://sewiki.iai.uni-bonn.de/teaching/lectures/se/2019/tutorien/

b) **(1 Punkt)** Für die praktischen Übungsaufgaben werden wir hauptsächlich die freie Entwicklungsumgebung Eclipse verwenden. Auf den Pool-Rechnern ist Eclipse installiert. Wir empfehlen Ihnen aber, sich die aktuelle Java- und Eclipse-Version auch auf Ihrem privaten Rechner einzurichten. Siehe https://sewiki.iai.uni-bonn.de/teaching/lectures/se/2019/infrastruktur/casetool#Eclipse

c) **(4 Punkte)** Die Bearbeitung und Einreichung der Aufgaben in den Übungsgruppen erfolgt ausschließlich über das Konfigurationsmanagement-System Git. Dieses wird in der ersten Vorlesungswoche vorgestellt.

- Installieren Sie auf Ihrem Rechner ein Git-System von http://git-scm.com/downloads
- Installieren Sie ein graphisches Front-End für Git. Siehe dazu die Hinweise auf

  https://sewiki.iai.uni-bonn.de/teaching/lectures/se/2019/infrastruktur/git/
- Zur Autorisierung beim Server brauchen Sie einen öffentlichen Schlüssel. Eine Anleitung zur Schlüssel-Erzeugung finden sie unter: https://sewiki.iai.uni-bonn.de/teaching/lectures/se/2019/infrastruktur/git/anleitung_sshkeygen
- Schicken Sie den Schlüssel bis spätestens Dienstag den **08.10.2019, 23:59** Uhr an die Liste der SWT-Tutoren (swt-tutoren@lists.iai.uni-bonn.de)!

d) **(2 Punkte)** Die Tutorien finden in den Räumen U1.041 und U1.042 statt. Für den Zutritt benötigen Sie den Zahlencode in der Datei „*Pool-Zugangspin.txt*“, welche im ReadOnly-Repository liegt: ssh://gitolite-se-swt@git.iai.uni-bonn.de/swt2019_readonly

Bitte stellen Sie sicher, dass Sie ein Informatik-Konto haben bzw. es rechtzeitig verlängert haben.

Siehe https://gsg.bit.uni-bonn.de/doku.php?id=de:accounts:students:manage für entsprechende Anleitungen (für Informatiker, Nebenfächler, FFF-ler, …).

Aufgabe 2. *Hello Git* **6 Punkte**

Zum Einüben des Umgangs mit Git arbeiten Sie gemeinsam mit Ihren Kommilitonen an einem Gedicht. Beginnen Sie damit, den privaten Schlüssel für den Dummy-User "swt.anonymous" von

https://sewiki.iai.uni-bonn.de/teaching/lectures/se/2019/infrastruktur/git/start
herunterzuladen. Anschließend müssen Sie:

1. das Repository **ssh://gitolite-se-swt@git.iai.uni-bonn.de/swt2019_for_all** „clonen“.
2. die Datei **gedicht.txt** editieren (Schreiben Sie eine oder ein paar neue Zeilen / Strophen zu dem Gedicht dazu – es kommt nicht auf dichterische Perfektion an ;-) Folgen Sie dabei der Anleitung am Anfang der Datei und tragen Sie Ihre neuen Zeilen unten in die Datei ein.
3. am Schluss die Änderungen „commiten“
4. evtl. weitere Änderungen durchführen und commiten,
5. wenn Sie meinen fertig zu sein, ein „pull“ durchführen (d.h. zwischenzeitliche Änderungen ihrer Kollegen in Ihre lokale Kopie übernehmen). Dabei treten evtl. Konflikte auf, die Sie lokal lösen können / müssen. In diesem Fall müssen Sie ein neues „commit“ der Konfliktlösungen anlegen.
6. am Schluss „pushen“ (d.h. alle lokalen Commits ins Git-Repository übertragen).

Dokumentieren Sie Ihre Vorgehensweise stichpunktartig in einer Text-Datei (.txt). Notieren Sie auch eventuelle Fehlermeldungen oder andere Abweichungen von dem hier beschriebenen Plan. Auf dem nächsten Übungsblatt werden Sie diese Datei dann in Ihr jeweiliges **Gruppen-Repository** hochladen.

Theoretische Aufgaben

Aufgabe 3. *Objektorientiert Grundbegriffe: Typen und Subtypen* **8 Punkte**

Erklären Sie kurz die folgenden Begriffe:

a) **(1 Punkt)** Ein Typ in Java ist entweder ein primitiver Typ (int, float, …) oder ein Objekttyp. Objekttypen werden durch Klassen oder Interfaces definiert (wobei technisch Interfaces auch Klassen sind, d.h. Instanzen der Metaklasse „Class“. → https://www.tutorialspoint.com/java/java_basic_datatypes.html

b) **(2 Punkte)** Ein generischer Typ ist ein Typ in dessen Definition Typvariablen benutzt werden. Eine Typvariable wird durch Angabe in spitzen Klammern hinter dem Typ-Namen definiert. Sie ist ein Platzhalter für einen Typ.
→ https://www.tutorialspoint.com/java/java_generics.htm

c) **(1 Punkt)** Aus einem generischen Typ kann man einen nicht-generischen Typ herleiten indem man alle Typvariablen durch Objekttypen ersetzt. Der aus der Ersetzung resultierende Typ heißt „parametrisierter Typ“.

d) **(2 Punkte)** Ein nicht-generischer bzw. nicht-parametrisierter Typ T_2 ist in Java Untertyp von T_1 genau dann, wenn er via „extends“ oder „implements“ direkt oder transitiv von T1 abgeleitet ist.
→ https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html

e) **(2 Punkte)** Das Liskovsche Ersetzbarkeitsprinzip (Liskov substitution principle) besagt, dass Instanzen eines Untertyps von T in jedem Kontext verwendbar sind / sein müssen, in denen Instanzen von T erwartet werden.
→ https://en.wikipedia.org/wiki/Liskov_substitution_principle

Aufgabe 4. *Objektorientiert Grundbegriffe: Generische Typen* **8 Punkte**

Entscheiden Sie für jede der folgenden Aussagen, ob sie wahr oder falsch sind und ersetzen Sie die „…“ durch korrekte Ergänzungen.

a) **(6 Punkte)** Sei T_2 ein Untertyp von T_1 und Set<X> ein generischer Typ.

1) Der Typ Set<T_1> ist ein legaler Typ. Wahr

2) Der Typ Set<T_2> ist ein legaler Typ. Wahr

3) Jeder Typ Set<T> ist genau dann ein legaler Typ, wenn T ein Untertyp von X ist. Falsch, X ist nur ein Platzhalter. Man kann von X nicht erben.

4) Der Typ Set<T_1> ist ein Untertyp von Typ Set<T_2>. Falsch

5) Der Typ Set<T_2> ist ein Untertyp von Typ Set<T_1>. Falsch

b) **(2 Punkte)** Sei List<X> ein Obertyp von LinkedList<X>. Dann ist die Zuweisung
List<String> ys = new LinkedList<String>();

a. ☒ legal, weil die Raw-Types in einer Subtyp-Beziehung sind und die sich entsprechenden Typparameter jeweils mit dem gleichen Typ instanziiert sind.
b. ☐ illegal, weil …

Aufgabe 5. *Abstrakte und finale Typen* **11,5 Punkte**

a) **(1 Punkt)** Wie unterscheidet sich die Subklasse einer abstrakten Klasse von der Subklasse eines Interfaces?

Subklassen von abstrakten Klassen

- Erben von der Oberklasse: **extends**
- Implementieren nicht zwangsläufig alle Methoden aus der Oberklasse (aber auf jeden Fall alle abstrakten!)
- Können nur von einer abstrakten Klasse erben

Subklassen von Interfaces

- Implementieren die Interfaces: **implements**
- Implementieren alle Methoden der Oberklasse (alle Methoden eines Interfaces sind abstrakt)
- Können mehrere Interfaces implementieren

b) **(4,5 Punkte)** Prüfen Sie die folgenden Aussagen auf ihren Wahrheitsgehalt. Falls die Aussage nicht oder nur teilweise stimmt, erläutern Sie warum.

1) In einer abstrakten Klasse können Felder deklariert werden. Wahr
2) In einem Interface können Felder deklariert werden. Wahr, Konstanten (**final**)
3) In einer abstrakten Klasse sind alle Methoden abstrakt. Falsch, es gibt auch konkrete Methoden, welche bereits eine Implementierung haben.
4) Eine abstrakte Methode besteht nur aus dem Methodenkopf, also ohne Implementierung. Wahr
5) Abstrakte Methoden können „private“ sein. Falsch, Subklassen können diese Methoden sonst nicht überschreiben.
6) Instanzen von abstrakten Klassen sind abstrakte Objekte. Falsch, man kann abstrakte Klassen nicht Instanziieren.
7) Da in Java keine multiple Vererbung möglich ist, kann eine Klasse nur von einer anderen konkreten Klasse, einer abstrakten Klasse oder einem Interface erben. Falsch, da Klassen nicht von Interfaces erben, sondern diese (auch mehrere) implementieren.
8) Eine abstrakte Klasse kann nicht von einer konkreten Klasse erben. Falsch
9) Eine abstrakte Klasse kann nicht von Interfaces erben. Wahr, da eine Klasse nicht von einem Interface erbt, sondern es implementiert.

c) **(1,5 Punkte)** Was ist ein finaler Typ in Java? Ein finaler Typ kann nicht mehr verändert werden.
Für eine Klasse heißt das, dass andere Klassen nicht von ihr erben dürfen.
Für Methoden heißt das, dass Sie in Subklassen (sofern die Klasse selbst nicht final ist!) nicht überschrieben werden können.
Für eine Variable heißt das, dass sie nur ein Mal initialisiert werden darf.

d) **(2,5 Punkte)** Prüfen Sie auch die folgenden Aussagen wieder auf ihren Wahrheitsgehalt. Falls die Aussage nicht oder nur teilweise stimmt, erläutern Sie warum.

1) Eine finale Klasse enthält standardmäßig nur finale Methoden. Wahr
2) Eine finale Methode enthält keine Implementierung. Falsch, gerade finale Methoden müssen Implementierungen enthalten, da sie sonst nutzlos wären.
3) Es gibt auch finale Variablen. Wahr, „Konstanten“
4) Eine finale Variable muss bei ihrer Deklaration mit einem Wert belegt werden. Falsch, die Unveränderbarkeit greift erst bei der ersten Initialisierung, nicht unbedingt schon bei der Deklaration.
5) Die Instanzvariablen eines Objekts eines abstrakten finalen Typens sind ebenfalls abstrakt und final. Falsch. Es ist völlig sinnlos, einen Typen abstrakt und final zu deklarieren, da man diesen Typen weder instanziieren (wegen abstrakt) noch von ihr erben (wegen final) kann. Folglich gäbe es keine Objekte dieses Typens.

e) **(2 Punkte)** Ist es sinnvoll, manche Methoden, die vom Konstruktor aufgerufen werden als final zu deklarieren? Begründen Sie! Falls Instanzen der Klasse, die den Konstruktor aufrufen, sich stets bei der Initialisierung gleich verhalten müssen, ist es sinnvoll. Durch das Setzen auf final können die Methoden, die im Konstruktor aufgerufen werden, nämlich nicht überschrieben werden und verhalten sich somit stets gleich.

∑ 41,5 Punkte