

Fragen, Antworten und Kommentare zur aktuellen Vorlesung

Frage: Bei der Aufgabe mit dem Set mussten ja equals(.) und hashCode() überschrieben werden. Soll bei equals(.) nur die Id oder der gesamte Inhalt verglichen werden?

Antwort: Einfache Frage, komplexe Antwort, da es von der Situation abhängt. Das bedeutet, dass dies projektabhängig gelöst und dann einheitlich umgesetzt werden muss. Ein wichtiges Teilthema ist dabei, dass die Eindeutigkeit einer ID als Primärschlüssel garantiert sein muss. Das ist in monolithischen Programmen mit einer Klassenvariable als Zähler meist noch einfach umsetzbar, bei verteilten Programmen, in denen Entitäten fast gleichzeitig entstehen können oft sehr komplex. Meist werden Ids dann von Datenbanken generiert, was allerdings erst geschieht, wenn ein Objekt das erste Mal persistiert wird. Für persistierte Objekte ist damit ein reiner Vergleich der Id sinnvoll. Schwierig wird es, falls es möglich ist, dass vermeintlich neue Objekte angelegt werden, von denen zunächst unklar ist, ob dieses Objekt schon im System existieren. Hier kann ein fachlicher Vergleich im equals(.) sehr sinnvoll sein, könnte aber auch in einer anderen Methode erfolgen. Das gilt insbesondere auch, da es Fälle gibt, in denen von der Gleichheit von Objektvariablen nicht auf die Gleichheit von Objekten geschlossen werden kann (Entitäts-Semantik), wie bei Objekten einer Auktion, bei denen verschiedene gleichartige Objekte (z. B. Briefmarke) versteigert werden können.

HashCode() und Equals(.) werden bei vielen Methoden in Java genutzt, so dass hier über die gewünschte Gleichheit nachgedacht werden muss. Zusammenfassend kann nur festgehalten werden, dass für equals(.) immer die vier Basisregeln eingehalten werden müssen.

- (a) symmetrisch: `x.equals(x)`
- (b) reflexiv: `x.equals(y) == y.equals(x)`
- (c) transitiv: aus `x.equals(y)` und `y.equals(z)` folgt `x.equals(z)`
- (d) nichts ist null: `!x.equals(null)`

Frage: Im Klassendiagramm auf Folie 183 sind alle Assoziationen ungerichtet, sind sie alle bidirektional?

Antwort: Das Klassendiagramm stammt aus der Entwicklung von Klassendiagrammen, da werden Assoziationen oft zunächst als einfache Verbindungen eingezeichnet. Sie sind damit unterspezifiziert, also im Detail noch nicht festgelegt. Im nächsten Schritt wird über Multiplizitäten und Objektvariablen für die Assoziationen nachgedacht. Dieser Schritt ist in der Abbildung dargestellt. Da es oft nur eine Objektvariable auf der Assoziation gibt, ist damit die Richtung vorgegeben. Beim genauen Blick auf das Diagramm fällt auf, dass es anscheinend eine bidirektionale Beziehung geben soll. (Generell ist ja die Idee solche Beziehungen möglichst zu vermeiden).

Frage: Wann kann ich mit der Hausarbeit beginnen?

Prinzipiell können Sie mit den Vorbereitungen jederzeit, also auch vor einer Woche starten. Sie können sich jetzt schon eine Bibliothek oder Framework ansehen, sich einarbeiten und über ein eigenes Beispiel nachdenken. Sie können dann die nachfolgenden Veranstaltungen über Design-Pattern nutzt, um zu evaluieren, ob bzw. welche was mit ihrem Thema zu tun haben oder ob Sie ihre ersten Lösungen überarbeiten müssen. Da Ihre Hausarbeit alle Gebiete von OOAD abdeckt, kann das Thema A = Anforderungsanalyse, schon detailliert bearbeitet werden.

Bedenken Sie dabei, dass Ihnen eventuell im Dezember noch eine Person zugeordnet wird, wenn Ihre Gruppe nicht aus 4 Personen besteht.

Frage: Größere Software wird ja typischerweise inkrementell entwickelt, wie sieht es damit in der Hausarbeit aus?

Antwort: Generell entwickeln Sie wahrscheinlich von Anfang an inkrementell in Mikroschritten, da sie erst einen typischen Ablauf ausprogrammieren, den ausprobieren und erst nach einer Zufriedenheit sich um weitere Funktionalität kümmern. Ähnlich sieht es mit Makroschritten in Projekten aus. In der Hausarbeit werden Sie meist nur ein Inkrement hinbekommen, evtl. auch mehrere. Um den Schreibaufwand nicht unnötig zu erhöhen, werden in der Hausarbeit die erreichten Ergebnisse mit ihren fachlichen Entscheidungen als Folge von Ergebnissen dokumentiert, was dann wie ein Wasserfall aussieht. Es ist also ein Ergebnisbericht und kein Erlebnisbericht, was wann gemacht wurde. Vereinfacht formuliert führt eine Formulierung der Art „Dann haben wir uns gemeinsam die benötigten Klassen ausgedacht“ sofort zum massiven Notenabzug, da dies nicht Teil eines akademischen Ergebnisses sein kann. Wenn solche Erfahrungen relevant sind, stehen sie abgetrennt in einem persönlichen Fazit oder einem Unterkapitel zur Vorgehensweise, dann ohne „wir“.

Frage: In ihren Coding-Guidelines steht, dass break nur in switch-case und continue gar nicht verwendet werden soll. Warum und widerspricht das nicht dem Early-Exit?

Antwort: Ein zentrales Ziel ist es, dass Programme von anderen einfach gelesen werden können. Je weniger Abzweigungen im Code vorkommen, desto einfacher ist er lesbar. Da sich das Herausspringen aus Schleifen oft mit Booleschen Variablen mit klingenden Namen realisieren lässt, ist das die schönere Lösung. Early-Exit ist bei Methoden am Anfang Standard. Bevor Sie irgendwas ausführen prüfen Sie ob die Methode überhaupt sinnvoll aufgerufen wurde und ob es triviale Ergebnisse, z. B. für leere Listen gibt. In Schleifen finde ich es ok, innen mit return herauszuspringen, um so ein schnelles Ende zu ermöglichen (kann man auch anders sehen).

Die genannte Regel ist keine der Form „es muss immer so sein“, es ist aber sinnvoll. In Java darf eine Variable nicht „goto“ heißen, zum Glück ist es aber als Befehl nie umgesetzt worden. Folgendes Programm ist aber leider in Java lauffähig.

```
public static void main(String[] args) {
    label2: for (int j = 0; j < 10; j++) {
        label1: for (int i = 0; i < 10; i++) {
            if (j == 3) {
                break label2;
            }
            if (i == 2) {
                break label1;
            }
            System.out.println("i j " + i + " " + j);
        }
    }
    label3: System.out.println("Ende");
}
```

Die Ausgabe lautet:

i j 0 0
i j 1 0
i j 0 1
i j 1 1
i j 0 2
i j 1 2
Ende