HSLU Hochschule Luzern

Objektorientierte Programmierung (OOP)

Übung SW07: equals(), hashCode() und Comparable

Themen: Einführung Collections (D01), Gleichheit und Vergleichbarkeit (O09)
Zeitbedarf: ca. 180min.

Roland Gisler, Version 1.5.5 (HS24)

1 Implementation von `equals()` und `hashCode()`

1.1 Lernziele

- Sie sind in der Lage die verschiedenen Arten von „Gleichheit“ zu definieren.
- Sie können die Methoden **`equals()`** und **`hashCode()`** adäquat überschreiben.
- Sie kennen die Anforderungen der verschiedenen Contracts.
- Sie können die Implementationen effizient und angemessen testen.

1.2 Grundlagen

Grundlage bildet der Input **`O09_IP_ObjectEqualsCompare`**. Auch für diese Übung greifen wir teilweise wieder auf frühere Übungen zurück. **Wichtig**: Diese Übung bildet eine **sehr wichtige** Grundlage für die nächste Woche folgenden Übungen zu den Datenstrukturen (Collections)!

1.3 Aufgaben

a.) Modellieren Sie eine komplett neue Klasse **`Person`**. Diese soll eine eindeutige Identifikationsnummer (ID, vom Typ **`long`**), einen Nachnamen (**`String`**) und einen Vornamen (**`String`**) haben. Die ID soll immutable implementiert sein und «von Hand» vergeben werden. Alle Attribute können mit einem Konstruktor gesetzt werden. Getter- und Setter-Methoden sind sinngemäss zu implementieren. Zusätzliche, sinnvolle Attribute können Sie nach Belieben ergänzen.
Tipp: Skizzieren Sie diese Klasse vorgängig in einem einfachen Klassendiagramm!

b.) Testen Sie mindestens den Konstruktor mit einem Unit-Test (sofern Sie das nicht schon längst gemacht haben) und überprüfen Sie, ob die Attribute korrekt gesetzt werden.

c.) Haben Sie auch die **`toString()`**-Methode überschrieben?
Wenn nein, empfehle ich Ihnen, das ab sofort bei **jeder** Klasse zu machen. Denn so können wir eine Klasse schnell und einfach auf die Konsole (oder später dann sinnvoller auch in ein Logfile) ausgeben! Das vereinfacht u.a. die Fehlersuche sehr.

d.) Überlegen Sie sich grundsätzlich welche Art von Gleichheit bei der Klasse **`Person`** sinnvoll ist, und diskutieren Sie mit anderen Studierenden die verschiedenen Varianten.

e.) Implementieren Sie einen Testfall mit Hilfe des «EqualsVerifier» (siehe http://jqno.nl/equalsverifier/). Die notwendige Library (JAR-Datei) ist im verwendeten Projekt-Template bereits enthalten, Sie können sie somit direkt verwenden.
Hinweis: Bei der Ausführung des Tests wird dieser vorerst mit der Warnung failen, dass wir die von **`Object`** vererbte Implementation von **`equals()`** verwenden, was bekanntlich meistens nicht sinnvoll ist, hier aber (noch) den Tatsachen entspricht.

f.) Ergänzen Sie zusätzlich mindestens zwei Testfälle welche beim Aufruf von **`equals()`** einmal zu einem **`true`** und einmal zu einem **`false`** führen (also einmal «gleich» und einmal «ungleich». Führen Sie auch diese Tests sofort aus – haben Sie dieses Resultat erwartet? (Zur Erinnerung: **`Object`** definiert Gleichheit als Identität der Objekte!)

g.) Implementieren Sie nun schrittweise die **`equals()`**-Methode. Nehmen Sie dazu die Hinweise und das Schema aus dem Input zu Hilfe. Führen Sie die vorhandenen Tests immer wieder aus und beobachten Sie fortlaufend die Ergebnisse. Nehmen Sie auch die gute Dokumentation von «EqualsVerifier» zu Hilfe (siehe e), und beachten Sie, dass eine Warnung nicht zwingend ein Fehler sein muss!
Wichtig: Verwenden Sie **nicht** die Codegenerierung Ihrer IDE. Und wenn Sie nicht widerstehen können, dann **nur** um **detailliert** zu untersuchen, wie sich die Implementationen unterscheiden, und um die Differenzen restlos zu verstehen!

h.) Implementieren Sie sinnvolle Tests für die **`hashCode()`**-Methode: Wenn zwei Personen im Sinne von **`equals()`** gleich sind, müssen auch die Hashwerte übereinstimmen! Andernfalls sollten sie möglichst (aber müssen nicht zwingend) unterschiedlich sein.

i.) Überschreiben Sie nun auch die **`hashCode()`**-Methode mit einer passenden Implementation und testen Sie auch hier fortlaufend.

j.) Beachten Sie, dass «EqualsVerifier» seine Arbeit sehr ernst nimmt, und dass Sie mit entsprechenden Suppressions auch definieren können/müssen, dass eine bestimmte Implementation bewusst so gewollt ist! Die Fehlermeldungen und die gute Dokumentation von «EqualsVerifier» werden Ihnen dabei weiterhelfen.

k.) In den bisherigen Übungen haben Sie auch die Klassen **`Point`** und **`Temperatur`** erstellt. Ergänzen Sie diese ebenfalls mit passenden **`equals()`**- und **`hashCode()`**-Methoden! Gehen Sie dabei nach gleicher Systematik vor wie bei der Klasse **`Person`**!

l.) Knacknuss: Erinnern Sie sich an die Klasse **`Element`** und ihre drei Spezialisierungen **`Stickstoff`**, **`Blei`** und **`Quecksilber`**? Überlegen Sie sich in welcher Art bei diesen Klassen «Gleichheit» gegeben ist! Lassen Sie sich von einer Implementation und entsprechenden Tests nicht abhalten!

2 Implementation von `Comparable`

2.1 Lernziele

- Sie verstehen die Semantik der **`compareTo()`**-Methode von **`Comparable`**.
- Sie können eine adäquate Implementation vornehmen.
- Sie können die korrekte Funktion testen und beurteilen.

2.2 Grundlagen

Grundlage bildet auch hier der Input **`009_IP_ObjectEqualsCompare`**.

Wichtig: Die Implementationen dieser Übung bilden eine essentielle Grundlage, damit wir die erweiterten Klassen in späteren Übungen zum Thema Datenstrukturen als Elemente in den verschiedenen Collections nutzen können!

2.3 Aufgaben

a.) Alle Klassen, bei welchen wir in der ersten Aufgabe **`equals()`** und **`hashCode()`** implementiert haben, wollen wir nun auch noch «**`Comparable`**» machen. Überlegen Sie sich für jede einzelne Klasse (**`Person`**, **`Temperatur`**, **`Element`** etc.) die sinnvolle «natürliche» Ordnung.

b.) Schreiben Sie auch hier zuerst die Testfälle. Pro Klasse bzw. **`compareTo()`**-Methode gibt es in der Regel mindestens **drei** sinnvolle Testfälle. Welche sind das? ➔ Test First!

c.) Beginnen Sie mit der Implementation der **`compareTo()`**-Methode. Testen Sie auch hier fortlaufend, und beobachten Sie idealerweise, wie Ihre bereits vorhandenen Testfälle immer «grüner» werden.

d.) Schauen Sie sich das Interface **`Comparator`** an. Vorsicht, dieses wird in kleinen, eigenständigen Klassen implementiert! Versuchen Sie mal einen Comparator zu implementieren, welcher in der Lage ist **`Person`**en nach Nachname und Vorname aufsteigend zu ordnen!