

SE1, Aufgabenblatt 9

Softwareentwicklung I – Wintersemester 2016/17

Benannte Schnittstellen

Moodle-URL: uhh.de/se1
Feedback zum Übungsblatt: uhh.de/se1-feedback
Projektraum Softwareentwicklung 1- WiSe 2016/17
Ausgabewoche 15. Dezember 2016

Kernbegriffe

In Java gibt es einen Mechanismus, mit dem ausschließlich der Umgang mit den Objekten einer Klasse modelliert werden kann, ohne Details ihrer Umsetzung festzulegen: *benannte Schnittstellen* (engl. *named interfaces*). Mit benannten Schnittstellen kann die Schnittstelle einer Klasse (die durch ihre öffentlichen Methoden definiert ist) explizit im Programm formuliert werden, indem in einem eigenen Konstrukt (mit dem Schlüsselwort `interface` statt `class`) nur die Köpfe der öffentlichen Methoden (ohne ihre Rümpfe) aufgeführt werden. Mit Hilfe benannter Schnittstellen können wir also von konkreten Implementationsdetails abstrahieren. Zur besseren Unterscheidung werden wir im Folgenden die benannten Schnittstellen von Java als *Interfaces* bezeichnen.

Genau wie eine Klasse definiert ein Interface einen Typ mit Operationen. Da ein Interface nur Methodenköpfe und keinerlei Anweisungen enthält, nennen wir die Methoden in einem Interface von nun an konsequent *Operationen* und sprechen nur noch von Methoden, wenn in Klassen Rümpfe mit Anweisungen vorliegen. Ein Interface kann keine Methodenrümpfe enthalten und keine Konstruktoren definieren, wir können also keine Objekte von Interfaces erzeugen. Implementiert eine Klasse ein Interface, so wird dies in der Klassendefinition mit dem Schlüsselwort `implements` deklariert. Verwendet werden die Objekte dieser Klasse dann üblicherweise unter dem Interface-Typ.




Bei einer Referenzvariablen unterscheiden wir zwischen ihrem *statischen Typ* und ihrem *dynamischen Typ*. Der statische Typ wird bei ihrer Deklaration festgelegt und steht bereits zur Übersetzungszeit fest. Der dynamische Typ hängt von der Klasse des Objektes ab, auf das die Referenzvariable zur Laufzeit verweist. Er ist dynamisch in zweierlei Hinsicht: Er kann erst zur Laufzeit ermittelt werden, und er kann sich während der Laufzeit ändern.

Eine Referenzvariable, die (noch) an kein Objekt gebunden ist, hat den besonderen dynamischen Typ „null-Typ“. Die Wertemenge dieses besonderen „null-Typs“ umfasst nur ein einziges Element: die null-Referenz (JLS §3.10.7).

Lernziele

Schnittstellen mit Interfaces formulieren können; die Unterschiede zwischen Klassen und Interfaces kennen; den statischen vom dynamischen Typ einer Variablen unterscheiden können.

Aufgabe 9.1 Verzeichnisse durchwandern und Dateien verarbeiten

-  9.1.1 Öffnet das Projekt *Dateisystem*. Wie viele verschiedene Pfeilarten könnt ihr im BlueJ-Klassendiagramm identifizieren? Was bedeuten diese verschiedenen Pfeilarten? **Schriftlich.**
-  9.1.2 Erzeugt ein Exemplar der Klasse `VerzeichnisWanderer`. Wenn ihr die Methode `start` aufrufen möchtet, benötigt ihr noch einen `DateiVerarbeiter` als Parameter für den Aufruf. Man kann aber keine `DateiVerarbeiter` erzeugen. Warum nicht? Was müsst ihr stattdessen machen? **Schriftlich.**
-  9.1.3 Was ist der statische Typ der Variable `_dateiVerarbeiter` in der Klasse `VerzeichnisWanderer`? Was ist ihr dynamischer Typ? **Schriftlich.**
- 9.1.4 Schreibt eine Klasse `TextdateiFilter`, die das Interface `DateiVerarbeiter` derart implementiert, dass nur Textdateien (`.txt`) aufgelistet werden. Ob in Kurz- oder Langform ist dabei egal. Schaut in der API der Klasse `java.lang.String` nach, welche Methoden bei dieser Aufgabe helfen könnten.
- 9.1.5 **Zusatzaufgabe:** Schreibt eine vierte Klasse, die das Interface `DateiVerarbeiter` implementiert. Diese Klasse soll einen Konstruktor `Filter(String, DateiVerarbeiter)` anbieten und allgemein nach der im ersten Parameter spezifizierten Erweiterung filtern und alle passenden Dateien an den im zweiten Parameter spezifizierten `DateiVerarbeiter` weiterreichen. Anwendungsbeispiel:

```
VerzeichnisWanderer wanderer = new VerzeichnisWanderer();  
DateiVerarbeiter lang = new LangAuflister();  
DateiVerarbeiter filter = new Filter("java", lang);  
wanderer.start(filter);
```

Aufgabe 9.2 Prägende Informatiker

- 9.2.1 Öffnet das Projekt *Informatiker* und studiert das Interface `Vergleicher`.
- 9.2.2 Erzeugt jeweils ein Exemplar der Klasse `PraegendeInformatiker` und der Klasse `PerNachname`. Ruft auf dem ersten Exemplar die Operation `schreibeGeordnet` auf übergeben das zweite Exemplar als Parameter. Daraufhin sollten die Informatiker per Nachnamen geordnet auf der Konsole erscheinen.
- 9.2.3 Die Klasse `PerNachname` vergleicht offenbar anhand des Nachnamens. Wozu mag wohl die Klasse `PerAlter` gut sein? Probiert sie aus!
- 9.2.4 Möglicherweise ist euch aufgefallen, dass gleichaltrige Personen untereinander keine dem Benutzer sinnvoll erscheinende Reihenfolge haben. Wäre es nicht praktisch, wenn man mehrere Vergleicher miteinander kombinieren könnte? Also erst das Alter vergleichen, und bei gleichem Alter den Nachnamen? Genau dafür gibt es die Klasse `Zweistufig`. Erstellt ein Exemplar und übergeben als Parameter jeweils ein Exemplar der Klassen `PerAlter` und `PerNachname`. Wenn ihr nun das Exemplar der Klasse `Zweistufig` an `schreibeGeordnet` übergeben, sollte die Liste deutlich sinnvoller aussehen.
- 9.2.5 Schreibt eine Klasse `PerVorname`, welche die Vornamen der Personen miteinander vergleicht.
- 9.2.6 Schreibt eine Klasse `PerGeschlecht`, welche Frauen vor Männern einstuft.
- 9.2.7 Könnt ihr durch geschickten Einsatz der Klasse `Zweistufig` auch 3 Vergleicher hintereinander schalten?
- 9.2.8 **Zusatzaufgabe:** Schreibt eine Klasse `Umgekehrt`, welche sich genau umgekehrt zu einem anderen Vergleich verhält. Beispielsweise soll `new Umgekehrt(new PerAlter())` junge Personen vor alten einstufen. Als Inspiration kann hierbei die (deutlich kompliziertere) Klasse `Zweistufig` dienen.

Aufgabe 9.3 Ein Sack voll Zahlen

Öffnet das Projekt *Zahlensaecke* und schaut euch das Interface `Zahlensack` an. Die drei implementierenden Klassen setzen die im Interface geforderte Funktionalität mit verschiedenen Mitteln um. Was in den Klassen genau passiert, ist für uns nicht wichtig, wir wollen sie nur benutzen.

- 9.3.1 Schreibt eine Klasse `Lotto` mit einer Methode `sechsAus49()`, welche sechs verschiedene Zufallszahlen im Bereich 1-49 auf die Konsole schreibt. Verwendet dazu innerhalb der Methode einen `Zahlensack`.
- 9.3.2 Wir wollen die verschiedenen Implementationen auf deren Effizienz überprüfen. Dazu gibt es bereits eine (unvollständige) Klasse `Effizienzvergleicher`, die in der Methode `vergleiche` Exemplare der drei Klassen erzeugt und diese an eine Methode `vermesse` weiterleitet. Ergänzt den fehlenden Code in der Methode `vermesse`! Greift dazu auf `long System.nanoTime()` zurück, welche die aktuelle Zeit in Nanosekunden liefert. Um vernünftige Messergebnisse zu erhalten ist es notwendig, sehr viele Methodenaufrufe durchzuführen. Verwendet dazu eine Zählschleife. Anschließend soll das Messergebnis einfach auf die Konsole geschrieben werden, sinnvollerweise in der Einheit ms. Was beobachtet ihr, wenn ihr `vergleiche` mit verschiedenen Werten für `groesse` aufruft, z.B. 1000/10000/100000?
- 9.3.3 Setzt einen Haltepunkt in der Methode `vermesse` und erklärt eurem Betreuer anhand des formalen Parameters `sack`, wo man den statischen und den dynamischen Typ sehen kann.



- 9.3.4 **Erklärt schriftlich** die Begriffe *statischer Typ* und *dynamischer Typ* am Beispiel der lokalen Variable `zs` aus der Methode `vergleiche`.

Aufgabe 9.4 Comparator

- 9.4.1 Das Vergleichen zweier Objekte wie im Interface `Vergleicher` (Aufgabe 9.2) ist ein solch zentraler Gedanke, dass dafür in Java bereits das Interface `Comparator` im Paket `java.util` mitgeliefert wird:

<https://docs.oracle.com/javase/7/docs/api/java/util/Comparator.html>

Damit `Comparator` für beliebige Typen von Objekten funktionieren kann, gibt man in spitzen Klammern an, um welchen Typ es sich jeweils handelt. In unserem Fall würde man also `Comparator<Person>` schreiben. Dieses Parametrisieren von Typen mit anderen Typen wird nächste Woche im Detail erläutert. Legt eine Kopie des Projekts *Informatiker* an, indem ihr in *BlueJ Projekt / Speichern unter...* auswählt und dann einen beliebigen Namen wählt, z.B. *Informatiker2*.

Entfernt das Interface `Vergleicher` und verwendet stattdessen den Typ `Comparator<Person>`. Dazu müsst ihr in jedem Quelltext, der diesen Typ verwendet, folgendes in die erste Zeile schreiben:

```
import java.util.Comparator;
```