Lehrstuhl für Angewandte Informatik I
Prof. Dr. Bernhard Westfechtel
Sandra Greiner

WS 2017/18
18. Januar 2018

Konzepte der Programmierung

Übung 12
Vererbung, Schnittstellen und Generizität

Abgabe: 01.02.2018

Hinweis: Alle Aufgaben auf diesem Übungsblatt lassen sich mit der `Main`-Klasse, die im e-Learning zur Verfügung gestellt wird, überprüfen.

Aufgabe 12.1 (Songverwaltung, Erweiterung)
Auf den letzten beiden Übungsblättern haben Sie jeweils eine doppelt-verkette Liste erstellt. Einmal wurden in den Listenelementen Plätzchendosen (`Dosenliste`) gespeichert und einmal wurden darin Songs (`Playlist`) gespeichert. Was machen Sie, wenn Sie eine Liste, die gleich aufgebaut ist, für Bücher erstellen wollen? Oder wie gehen Sie vor, wenn Sie DVDs genauso abspeichern wollen?

Eine erste Lösung wäre, jeweils die Liste zu kopieren und statt des bisherigen Eintragstyp in der Klasse `Element`, das jeweilig andere Artefakt zu nehmen, also z.B. ein Buch oder die DVD. Diese Lösung ist zwar möglich, widerspricht jedoch einer gut implementierten Programmarchitektur. Zudem ist das Vorgehen fehleranfällig: Stellen Sie sich vor, Sie vergessen die Implementierung an manchen Stellen zu ändern.

Besser ist es, eine **einzige** Klasse `Liste` zu implementieren, die verschiedene Eintragstypen unterstützt. Dafür eignet sich das Prinzip der **Generizität**. Wir implementieren in dieser Aufgabe deswegen eine **generische** doppelt-verkettete Liste.

Folgendes ist dafür nötig:

- Implementieren Sie eine Klasse `Liste` mit einem Typparameter `T`. Darin wird genauso wie auf den letzten Blättern eine Klasse `Element` definiert, die nun aber abhängt von dem Typparameter `T`. Ersetzen Sie darin jeweils den bisherigen Datentyp (Plätzchendose oder Song) durch den Parameter `T`.
- Bieten Sie (mindestens) die folgenden Methoden wieder an, die sich genauso verhalten sollen, wie in der (den) bisherigen Implementierung(en):
  - `gibGroesse(...)`
  - `fuegeEin(...)` an einer Position
  - `loesche(...)` an einer Position

Mit diesen Änderungen ist es möglich, verschiedene Listen zu erstellen. Überprüfen Sie Ihre Implementierung mit der gegebenen `Main`-Klasse. Integrieren Sie dazu auch die Klassen `Song` (vom letzten Übungsblatt) und `Buch` in Ihr BlueJ-Projekt.

(**3 Punkte**)

Aufgabe 12.2 (Pakete, Schnittstellen, Generizität)
Diese Aufgabe dient als gute Übung, die im Titel genannten Konzepte nochmals gemeinsam anzuwenden. Da die Aufgabe sehr umfangreich ist, jedoch zentrale Komponenten der Vorlesung einübt, ist eine Bearbeitung (und deren Abgabe zur Korrektur) äußerst ratsam.

Gegeben ist die folgende generische Schnittstelle:

```
1  package kdp1718.uebung12.datenstrukturen.schnittstelle;
2
3  /**
4  * Schnittstelle fuer assoziative Datenstrukturen, die Werte vom Typ W
5  * unter einem Schluessel vom Typ S ablegen. Auf dem Wertebereich von
6  * S ist eine totale Ordnung definiert.
7  */
8  public interface AssoziativerZugriff<S extends Comparable<S>, W> {
9
10     /**
11     * Gibt zurueck, ob dem uebergebenen Schluessel ein Wert zugeordnet ist.
12     */
13     public boolean enthaelt(S schluessel);
14
15     /**
16     * Gibt den Wert zurueck, der dem gegebenen Schluessel zugeordnet ist. Falls
17     * Kein Wert zugeordnet ist, wird null zurueckgegeben.
18     */
19     public W gib(S schluessel);
20
21     /**
22     * Legt den gegebenen Wert unter dem gegebenen Schluessel ab. Ein eventuell
23     * schon existierender Wert wird ueberschrieben und zurueckgegeben. War dem
24     * Schluessel zuvor kein Wert zugeordnet, wird null zurueckgegeben.
25     */
26     public W fuegeEin(S schluessel, W neuerWert);
27 }
28
```

Machen Sie sich zunächst mit der gegebenen Schnittstelle und der Schnittstelle `Comparable<T>` `aus der Java-Klassenbibliothek` [1] vertraut. Arbeiten Sie sich außerdem in den Lösungsvorschlag aus Aufgabe 10.3 (Binäre Suchbäume als assoziative Datenstruktur) ein. Implementieren Sie die folgenden Klassen und Pakete. Verwenden Sie **explizite Importe**, wann immer möglich.

(a) Erstellen Sie ein neues BlueJ-Projekt und fügen Sie die obige Schnittstelle in ein neues **Paket** `kdp1718.uebung12.datenstrukturen.schnittstelle` ein.

(b) Erstellen Sie in Ihrem BlueJ-Projekt ein neues Paket `kdp1718.uebung12.datenstrukturen.implementierung` mit einer generischen Klasse `BinaerBaum`, die die Schnittstelle `AssoziativerZugriff` unter Nutzung der gleichen Parameter, die in der Schnittstelle verwendet werden, implementiert. Orientieren Sie sich bei der Implementierung am Lösungsvorschlag aus Aufgabe 10.3, wo als Schlüsseltyp implizit `String` (die ISBN) und als Wertetyp `String` (der Titel des Buchs) angenommen wurde.
Hinweis: Anstatt eines numerischen Vergleichs (`<`, `>`) sollten Sie eine bestimmte Methode der Schnittstelle `Comparable` verwenden. Zur Prüfung der Gleichheit der Schlüssel verwenden Sie nicht den Operator `==`, sondern die Objektmethode `equals`.

[1] http://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html

(c) Erstellen Sie im Paket `kdp1718.uebung12.stundenplan` eine neue Klasse `Termin`, die das Interface `Comparable<Termin>` implementiert. Sie soll zwei private Objektfelder (Wochentag, Stunde), einen passenden Konstruktor und lesende Zugriffsmethoden besitzen. Definieren Sie für den Wochentag einen verschachtelten Aufzählungstypen mit den Literalen `MONTAG` bis einschließlich `FREITAG`. Implementieren Sie alle von der Schnittstelle deklarierten Methoden in sinnvoller Weise.
Hinweis: Mit der Objektmethode `name()` können Sie den Namen (z.B. `MONTAG`), mit `ordinal()` den Ordnungswert eines Enum-Literals erfragen (`MONTAG` = 0 etc.).

(d) Erstellen Sie im Paket `kdp1718.uebung12.stundenplan` eine Klasse `Lehrveranstaltung`, die sich aus drei privaten Objektfeldern vom Typ `String` für den Namen, den Dozenten und den Raum einer Lehrveranstaltung zusammensetzt und auch einen passenden Konstruktor und lesende Zugriffsmethoden beinhaltet.

(e) Erstellen Sie im Paket `kdp1718.uebung12.stundenplan` eine Klasse `Stundenplan`, die von `BinaerBaum<Termin, Lehrveranstaltung>` erbt. Außerdem soll die `toString()`-Methode so definiert sein, dass der Stundenplan in zeitlicher Reihenfolge (s. Beispiel unten) ausgegeben wird. Definieren Sie hierfür eine geeignete **rekursive** Hilfsmethode.

(f) Erstellen Sie im Paket `kdp1718.uebung12.stundenplan.beispiel` eine Klasse `Beispiel` mit einem **Hauptprogramm**, in dem ein leerer Stundenplan erzeugt wird. Anschließend sollen mehrere Lehrveranstaltungen, die Sie besuchen, in zeitlich zufälliger Reihenfolge eingefügt werden. Schließlich soll Ihr Stundenplan auf der Standardausgabe ausgegeben werden.

Beispiel-Ausgabe

```
1  MONTAG 8 Uhr: Rechnerarchitektur und Rechnernetze, Prof. Rauber, H33
2  MITTWOCH 8 Uhr: Konzepte der Programmierung, Prof. Westfechtel, H33
3  MITTWOCH 14 Uhr: Rechnerarchitektur und Rechnernetze, Prof. Rauber, H33
4  DONNERSTAG 16 Uhr: Konzepte der Programmierung, Prof. Westfechtel, H33
5
```

(1+4+3+2+2+2 = 14 Punkte)

Aufgabe 12.3 (Newtonverfahren mit Interfaces)
In dieser Aufgabe ermöglichen wir mithilfe von Interfaces die Berechnung des Newtonverfahrens. Mit dem Newtonschen Näherungsverfahren lassen sich Näherungen für die Nullstellen einer Funktion finden.[2]

Dazu berechnet man eine Folge von Näherungswerten anhand folgender Iterationsgleichung:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Zusätzlich benötigt man noch einen Startwert x_0 aus der Nähe der vermuteten Nullstelle und die Funktion $f : \mathbb{R} \to \mathbb{R}$ muss in der Nähe der Nullstelle eine von 0 verschiedene Ableitung f' besitzen.
Erstellen Sie dafür zunächst ein Paket `newton`, in dem sie folgende Komponenten einfügen:

(a) ein Interface `Funktion`, welches eine Methode `berechneFunktionswert(...)` definiert. Ihr wird eine Kommazahl für x übergeben und sie gibt auch eine Kommazahl zurück.

[2]Siehe auch: http://de.wikipedia.org/wiki/Newton-Verfahren

Sowie ein Interface `DiffbareFunktion`, bei dem es sich um eine spezielle Funktion handelt. Es definiert daher zusätzlich die Methode `berechneAbleitung(...)`, die ebenfalls eine Kommazahl zurückgibt und x übergeben bekommt.

(b) eine Klasse `Polynom`, welche das Interface `DiffbareFunktion` implementiert. Aus Gründern der Vereinfachung nehmen wir hier als einziges Polynom die Funktion

$$f(x) = 3x^3 - 5x^2 + 8$$

an.

(c) ein Klasse `Newton` mit einer Klassenmethode `berechneNullstelle(...)`, die ein Objekt vom Typ `DiffbareFunktion` und den Startwert x_0 übergeben bekommt. Die Methode implementiert das Newton-Verfahren unter Verwendung der Methoden `berechneFunktionswert(...)` und `berechneAbleitung(...)`. Die Iteration soll solange durchgeführt werden, bis $|x_{n+1} - x_n| \leq \epsilon$ für ein vorgegebenes ϵ gilt. Da die Konvergenz des Newton-Verfahrens nicht immer garantiert ist, müssen Sie die Iteration spätestens nach einer maximalen Anzahl an Iterationen abbrechen.
Definieren Sie in der Klasse daher noch geeignete **Konstanten** für ϵ und die maximale Anzahl an Iterationen und belegen Sie sie mit sinnvollen Werten.

(d) Wieso ist eine Lösung mit abstrakten Klassen an dieser Stelle ungeschickt? Erklären Sie die Unterschiede und Gemeinsamkeiten von Interfaces und abstrakten Klassen.

(**1+2+4+2 = 9 Punkte**)

Aufgabe 12.4
In Aufgabe 11.5 haben wir zwei verschiedene Implementierungen für die Datenstruktur Schlange implementiert. Die Java Klassenbibliothek bietet selbst ein generisches Interface für Listen an: `java.util.List<E>`[3]),
wobei der Typparameter bei der Nutzung durch den aktuellen Datentyp ersetzt wird.

Wiederum finden sich davon verschiedene Implementierungen, deren Details für uns nicht besonders relevant sind. Man kann im Normalfall davon ausgehen, dass die Implementierung fehlerfrei funktioniert. Machen Sie sich mit dem List-Interface, insbesondere seinen Methoden, vertraut. Machen Sie sich klar, welche Funktionen man darauf aufrufen kann.

Implementieren Sie nun eine Klasse `SchlangeStandardBib`, die ebenfalls die Datenstruktur Schlange implementiert (das Interface `Schlange` aus Aufgabe 11.5). Allerdings soll die komplette Implementierung durch eine Liste der Standardbibiliothek implementiert sein, d.h. die Funktionalität wird weitergereicht an Methodenaufrufe der Standardklasse. Implementieren Sie dazu ein Objektfeld vom Typ `List` und initialisieren Sie es mit einer leeren `LinkedList`[4].

Delegieren Sie die Implementierung der einzelnen Operationen an geeignete Methodenaufrufe der Liste. (**3 Punkte**)

[3] https://docs.oracle.com/javase/8/docs/api/java/util/List.html
[4] https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html