Human-Computer Interaction Group

Leibniz Universität Hannover

Vorlesung:
Priv.-Doz. Dr.-Ing. Matthias Becker
eMail: xmb@hci.uni-hannover.de

Übungsbetrieb:
Patric Plattner
eMail: patric.plattner@hci.uni-hannover.de

Aufgabenblatt vom **15.05.2019**
Abgabe bis **21.05.2019, 23:59**
Korrektur ab **23.05.2019**

5. Hausübung

Abstrakte Klassen, Interfaces

In diesem Abgabeblatt üben wir das Vererben von abstrakten Klassen und Interfaces. Dazu werdet ihr zwei kleinere Aufgaben bearbeiten, eine zu abstrakten Klassen und eine zu Interfaces. Die beiden Aufgaben sollen in zwei verschiedenen Ordnern auf der zip Datei liegen.

Aufgabe 1

In dieser Aufgabe sollt ihr eure Abgabe aus der letzten Woche nehmen und ein wenig abändern. In der letzten Woche habt ihr konkrete Vererbung geübt, indem ihr einen Binärbaum `IntBinTree` implementiert und dann einen Suchbaum `IntSearchTree` davon abgeleitet habt. Allerdings ist konkrete Vererbung nicht der beste Mechanismus um solch eine Beziehung zu modellieren. Bei konkreter Vererbung ist man nämlich darauf angewiesen, dass die Super-Klasse bereits Implementierungen hat, also braucht man eine Art Referenz-Implementierung. Allerdings ist das oftmals nicht wünschenswert. So würde es z.B. bedeuten, dass die Implementierung von `insert()` und `search()` in IntBinTree eine Eigenschaft eines Binärbaumes an sich sind, was so nicht stimmt. Die Implementierungen der beiden Methoden sind nur *eine* Möglichkeit. Die restlichen Methoden wie `getLeft/Right()` sind hingegen Eigenschaften eines Binärbaumes an sich, also sollte man diese auch in der Super-Klasse implementieren. Um sowas zu modellieren, gibt es eine andere Art der Vererbung: Vererbung von abstrakten Klassen.

Aufgabenstellung

In dieser Aufgabe sollt ihr eure Abgabe von letzter Woche noch einmal runterladen, falls ihr die Dateien schon gelöscht habt, oder ihr nutzt die Vorlage, die in der Zip Datei zu dieser Übungsaufgabe im Studip hochgeladen wurde. Eure Aufgabe ist wie folgt: Benennt eure Klasse `IntBinTree` in `IntRandTree` um. Erstellt dann eine neue abstrakte Klasse `IntBinTree`, die einen Konstruktor hat und die folgende Methoden implementiert:

- `IntBinTree getLeft()`
- `IntBinTree getRight()`
- `void setLeft(IntBinTree t)`
- `void setRight(IntBinTree t)`
- `int getContent()`

Alle Abgaben sind bis zu dem in der Kopfzeile vermerkten Datum in unserem UploadTool abzugeben. Eine Anleitung zum Hochladen findet ihr hier. Prüft ob eure Abgabe ordentlich hochgeladen wurde, indem ihr sie selber nach dem Hochladen runterladet. Solltet ihr es nicht schaffen, innerhalb des vorgegebenen Zeitraumes die Lösung korrekt abzugeben, wird dies zum Nichtbestehen der Studienleistung führen. Bei technischen Problemen mit dem UploadTool bitte eine Mail an patric.plattner@hci.uni-hannover.de schicken.

Vorlesung:
Priv.-Doz. Dr.-Ing. Matthias Becker
eMail: xmb@hci.uni-hannover.de

Übungsbetrieb:
Patric Plattner
eMail: patric.plattner@hci.uni-hannover.de

Aufgabenblatt vom **15.05.2019**
Abgabe bis **21.05.2019, 23:59**
Korrektur ab **23.05.2019**

- `String inOrder()`
- `String toString()`

Zusätzlich dazu soll die Klasse auch zwei Abstrakte Methoden haben:

- `boolean search(int i) //gibt true zurück falls i in baum`
- `void insert(int i)`

Dann bearbeitet ihr die Klassen IntRandTree und IntSearchTree so, dass sie von IntBinTree erben, einen Konstruktor haben und nur die beiden abstrakten Methoden implementieren.

Behaltet dabei die Package-Struktur bei und ändert eure Main Methode so, dass sie dasselbe macht wie in Übungsblatt 04, aber mit der neuen Klassenstruktur funktioniert. Passt die Javadoc Kommentare der abgeleiteten Klassen und schreibt Javadoc Kommentare für die neue Klasse.

Vorlesung:
Priv.-Doz. Dr.-Ing. Matthias Becker
eMail: xmb@hci.uni-hannover.de

Übungsbetrieb:
Patric Plattner
eMail: patric.plattner@hci.uni-hannover.de

Aufgabenblatt vom **15.05.2019**
Abgabe bis **21.05.2019, 23:59**
Korrektur ab **23.05.2019**

Aufgabe 2

Es gibt aber auch noch eine weitere Art der Vererbung: Das Implementieren von Interfaces. Interfaces unterscheiden sich von abstrakten Klassen, indem ein Interface keine Implementierungen beinhaltet[1]. Ein Interface ist nur eine Art „Vertrag“, der beschreibt, was für Methoden eine Klasse mindestens implementiert, ohne dabei Details über die Implementierung zu verraten. Wir werden in dieser Aufgabe ein Interface schreiben, welches uns Objekte auf verschiedene Arten und Weisen sortieren lassen kann.

Aufgabenstellung

Ihr sollt zunächst ein Enum `SortMode` implementieren, das 3 Einträge hat: `NAME, PRICE, ID`. Danach sollt ihr ein Interface `ISortable` schreiben, welches genau eine Methode vorgibt:

- `public String getSortString(SortMode mode);`

Nun schreibt ihr drei Klassen, die das Interface `ISortable` implementieren:

- `BluRay`
  - Eine BluRay hat einen Titel(String), einen Preis(int), einen Publisher(String) und einen ASIN Code(String).
- `CD`
  - Eine CD hat einen Albumtitel(String), einen Künstler(String), einen Publisher(String), einen Preis(int) und einen ASIN Code(String)
- `Book`
  - Ein Buch hat einen Titel(String), einen Author(String), einen Publisher(String), einen Preis(int) und eine ISBN-13(String)

Implementiert diese drei Klassen. Die Klassen sollen Getter für alle Felder (keine Setter), die eben gerade beschrieben wurden, haben. Schreibt zu jeder Klasse einen Konstruktor, der ein Argument pro Feld nimmt und damit die Felder initialisiert. Hier ein Beispiel:

```
Book(String title, String author, String publisher, int eurPrice, String ISBN)
```

Nun zu den Implementierungen der `getSortString()` Methode. Diese sollen je nach SortMode einen String zurückgeben, der auf eine bestimmte Art und Weise formatiert ist:

- `BluRay`
  - `SortMode NAME: „${title}-${publisher}“`

1 Es gibt zwar Default Implementierungen für Interfaces, aber diese ignorieren wir hier.

Vorlesung:
Priv.-Doz. Dr.-Ing. Matthias Becker
eMail: xmb@hci.uni-hannover.de

Übungsbetrieb:
Patric Plattner
eMail: patric.plattner@hci.uni-hannover.de

Aufgabenblatt vom **15.05.2019**
Abgabe bis **21.05.2019, 23:59**
Korrektur ab **23.05.2019**

- `SortMode PRICE: „${price}“ //mit führenden Nullen auf 6 Stellen auffüllen`
- `SortMode ID: „${ASIN}“`

- `CD`
  - `SortMode NAME: „${title}-${artist}-${publisher}“`
  - `SortMode PRICE: „${price}“ //mit führenden Nullen auf 6 Stellen auffüllen`
  - `SortMode ID: „${ASIN}“`
- `Book`
  - `SortMode NAME: „${title}-${author}-${publisher}“`
  - `SortMode PRICE: „${price}“ //mit führenden Nullen auf 6 Stellen auffüllen`
  - `SortMode ID: „${ISBN}“`

Dann soll eine Klasse Sort erstellt werden. Diese Klasse soll eine statische Methode sort mit folgender Signatur haben:

```
public static void sort(SortMode mode, ISortable[] toSort)
```

Die sort Methode soll `toSort` sortieren. Die Sortierung soll anhand der Sortierstrings von getSortString() vorgenommen werden. Ihr koennt Strings anhand der `String.compareTo()`[2] Methode vergleichen. Fuer das Sortieren koennt ihr einen beliebigen Algorithmus benutzen, wir schlagen hier aber einen vor:

```
Algorithm sort(String[] s) returns null
for int i = 0 to s.length() - 2
      for int j = i + 1 to s.length() - 1
            if s[i] > s[j] then
                  String tmp = s[i];
                  s[i] = s[j];
                  s[j] = tmp;
```

Erstellt dann eine `Main` Klasse mit `main` Methode, in der ihr ein Array an `ISortable` mit mindestens 10 Eintraegen erstellt und dann sortiert.

2 https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#compareTo-java.lang.String-

Vorlesung:
Priv.-Doz. Dr.-Ing. Matthias Becker
eMail: xmb@hci.uni-hannover.de

Übungsbetrieb:
Patric Plattner
eMail: patric.plattner@hci.uni-hannover.de

Aufgabenblatt vom **15.05.2019**
Abgabe bis **21.05.2019, 23:59**
Korrektur ab **23.05.2019**

Die Package-Struktur soll wie gewohnt so aussehen: `de.uni_hannover.hci.name`. Legt die einzelnen Klassen in sinnvolle Subpackages. Verseht eure Klassen und Methoden mit sinnvollen Javadoc Kommentaren.