Technische Universität München
Fakultät für Informatik
R. Kuntschke, I. Nagy,
Dr. M. Schneider

SS 2006
Lösungsvorschläge zu Blatt 10
14. Juli 2006

Übungen zu Einführung in die Informatik I

Aufgabe 41 Modellierung und Implementation eines Bücherregals (Lösungsvorschlag)

a) Zunächst identifizieren wir aus der Beschreibung die Dienste, die die Klasse `BuecherRegel` zur Verfügung stellen soll:

(i) *„Mein Bücherregal kann leer sein oder eine Anzahl von Büchern enthalten."*
Hier identifizieren wir einen Dienst `getAnzahlBuecher()`. Mit diesem Dienst kann auch festgestellt werden, ob das Regal leer ist (Abfrage auf `0`).

(ii) *„Wenn es voll ist, kann es erweitert werden."*
Hier identifizieren wir zwei Dienste: `istVoll()` und `erweitere()`.

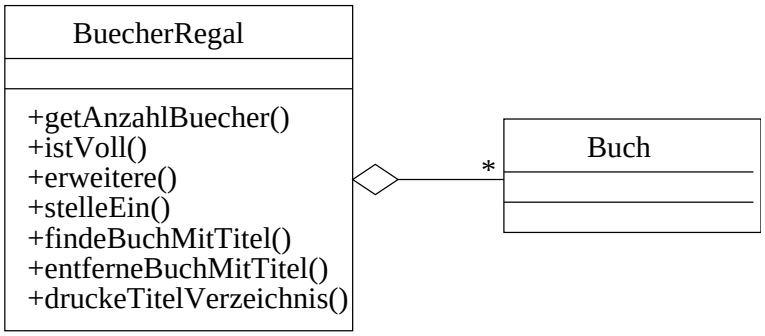
(iii) *„In mein Bücherregal kann man ein neues Buch einstellen, ..."*
Hier identifizieren wir einen Dienst `stelleEin()`.

(iii) *„... nach einem Buchtitel suchen ..."*
Hier präzisieren wir, dass wir nach einem Buch mit einem gewünschten Titel suchen, und identifizieren damit einen Dienst `findeBuchMitTitel()`.

(iv) *„... und ein Buch mit einem gewünschten Buchtitel entnehmen."*
Hier identifizieren wir einen Dienst `entnehmeBuchMitTitel()`.

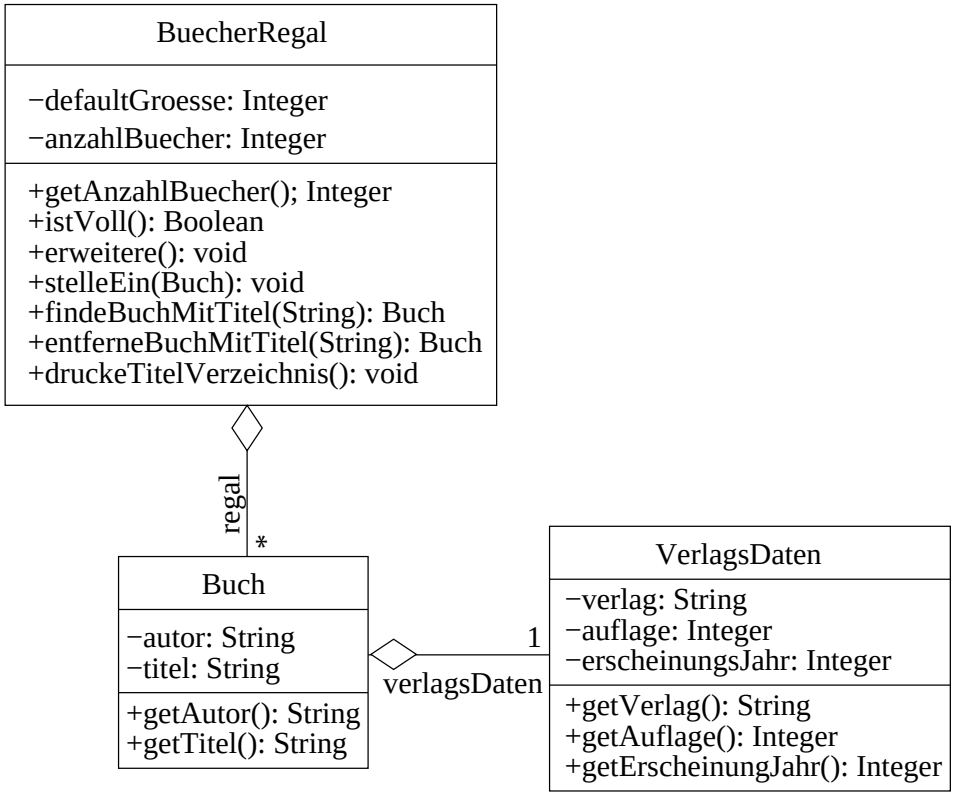
(v) *„Außerdem hat mein Bücherregal ein Verzeichnis der enthaltenen Buchtitel."*
Hier identifizieren wir schließlich einen Dienst `druckeTitelVerzeichnis()`.

Damit können wir nun ein erstes Modell in UML angeben, in dem nur diese Dienste aufgeführt sind:



Als Attribute der Klasse wird sicherlich `anzahlBuecher` benötigt, das angibt, wieviele Bücher aktuell im Regal stehen. Wir führen zusätzlich noch ein Attribut `defaultGroesse` ein, das angibt, wieviele Bücher in einem neuen Regal (d. h. vor der ersten Erweiterung) Platz finden sollen und um wieviele Plätze ein neues Regal erweitert werden soll.

Damit ergibt sich im detaiilierten Entwurf folgendes Klassendiagramm:



b) Zur Implementation des Bücherregals mittels einer Reihung `Buch[] regal` stehen im Prinzip folgende beiden Varianten zur Verfügung:

Variante 1: Die `anzahlBuecher` im Regal vorhandenen Bücher stehen auf den Indexpositionen `0, 1, ... , anzahlBuecher - 1`. Ein neues Buch wird, falls noch Platz vorhanden ist, auf Indexposition `anzahlBuecher` abgelegt. Die Lücke, die beim Entfernen eines Buches entsteht, wird geschlossen, indem man das letzte Buch (auf Indexposition `anzahlBuecher - 1`) auf diesen Platz vorzieht. Leere Plätze sind durch ihre Position bestimmt und müssen nicht eigens gekennzeichnet werden.

Diese Variante ist im nachfolgenden Java-Programm implementiert.

Variante 2: Die `anzahlBuecher` im Regal vorhandenen Bücher stehen irgendwo in der Reihung verteilt. Leere Indexpositionen sind durch `null` gekennzeichnet. Beim Einstellen eines Buches wird eine freie Indexposition gesucht, beim Entfernen eines Buches wird der entsprechende Platz durch `null` gekennzeichnet.

Zur Frage, ob das Erweitern automatisch angestoßen werden soll oder nicht, haben wir uns in der nachfolgenden Implementation für das automatische Erweitern beim Einstellen eines Buches entschieden. In diesem Fall muss die Operation `erweitere()` nicht zur Schnittstelle gehören und wurde im Javaprogramm auch mit `private` gekennzeichnet.

```
public class BuecherRegal {
   final private int defaultGroesse = 5;

   private Buch[] regal;
   private int anzahlBuecher;

   public BuecherRegal () {
      regal = new Buch[defaultGroesse];
      anzahlBuecher = 0;
   }
```

```
private void erweitere () {
        // groesseres Regal kreieren:
   Buch[] regalNeu = new Buch[regal.length + defaultGroesse];
        // Inhalt kopieren:
   for (int index=0; index<anzahlBuecher; index++) {
      regalNeu[index] = regal[index];
   }
        // aus neu mach alt:
   regal = regalNeu;
}

public boolean istVoll () {
   return anzahlBuecher == regal.length;
}

public int getAnzahlBuecher () {
   return anzahlBuecher;
}

public void stelleEin (Buch b) {
        // wenn voll dann erweitern:
   if ( istVoll() ) {
      erweitere();
   }
        // neues Buch hinten hinstellen:
   regal[anzahlBuecher] = b;
        // anzahlBuecher konsistent erhoehen:
   anzahlBuecher++;
}

public Buch findeBuchMitTitel (String titel) {
   for (int index=0; index<anzahlBuecher; index++) {
      if (regal[index].getTitel().equals(titel)) {
            // Buch ist gefunden:
         return regal[index];
      }
   }
            // Buch nicht enthalten:
   return null;
}

public Buch entferneBuchMitTitel (String titel) {
   for (int index=0; index<anzahlBuecher; index++) {
      if (regal[index].getTitel().equals(titel)) {
            // Buch ist gefunden:
         Buch buch = regal[index];
            // Platz durch hinterstes Buch belegen:
         regal[index] = regal[anzahlBuecher -1];
            // anzahlBuecher konsistent erniedrigen:
         anzahlBuecher--;
            // gefundenes Buch ausliefern:
         return buch;
      }
   }
```

```
                // Buch nicht enthalten:
        return null;
    }

    public void druckeTitelVerzeichnis () {
        System.out.println("Verzeichnis der enthaltenen Buchtitel:")
            ;
        for (int index=0; index<anzahlBuecher; index++) {
            System.out.println((index+1) + ". Buch: " + regal[index].
                getTitel());
        }
    }

}
```

Aufgabe 42 Implementation des Bücherregals auf sortierter Reihung (Lösungsvorschlag)

a) In der Methode `stelleEin()` wird zunächst die Stelle gesucht, wo das Buch eingefügt werden muss (hinter allen Büchern mit lexikographisch kleinerem Titel), dann werden alle Bücher mit größerem Titel um eine Stelle nach hinten verschoben und schließlich das neue Buch auf dem frei geweordenen Platz eingefügt:

```
public void stelleEin (Buch b) {
        // wenn voll dann erweitern:
    if ( istVoll() ) {
        erweitere();
    }
        // Suche den Index, an dem das neue Buch eingefuegt wird
            :
    String titel = b.getTitel();
    int index = 0;
        // Buecher mit lexikographisch kleinerem Titel
            ueberspringen:
    while (index < anzahlBuecher && regal[index].getTitel().
        compareTo(titel) < 0)  {
        index++;
    }
        // Jetzt ist die Einfuegestelle gefunden (kann auch ganz
            hinten sein)
    int einfuegeIndex = index;
        // Alle Buecher ab der Einfuegestelle um eins nach
            rechts verschieben;
        // Schleife muss dabei von hinten nach vorn laufen:
    for (index = anzahlBuecher; index > einfuegeIndex; index--)
        regal[index] = regal[index-1];
        // Jetzt kann das neue Buch eingestellt werden
    regal[einfuegeIndex] = b;
        // anzahlBuecher konsistent erhoehen:
    anzahlBuecher++;
}
```

Um in den Methoden `findeBuchMitTitel()` und `entferneBuchMitTitel()` schnell das gesuchte Buch zu finden, wird eine Hilfsmethode `findeIndexZumTitel()` eingeführt, die den Algorithmus der binären Suche aus der Vorlesung implementiert:

```
private int findeIndexZumTitel (String titel) {
   // Grenzen des Suchintervalls werden vorbesetzt:
   int links = 0;
   int rechts = anzahlBuecher - 1;

   // Solange das Suchintervall nicht leer ist:
   while (links <= rechts) {
      // Vergleiche das mittlere Element mit dem Titel:
      int index = (links + rechts) / 2;
      int compare = regal[index].getTitel().compareTo(titel);
      if (compare < 0) {
         // weiter rechts suchen:
         links = index + 1;
      }
      else if (compare > 0) {
         // weiter links suchen:
         rechts = index - 1;
      }
      else {
         // Titel gefunden:
         return index;
      }
   }
       // Titel kann nicht enthalten sein:
   return -1;
}
```

Damit können nun die Methoden `findeBuchMitTitel()` und `entferneBuchMitTitel()` angegeben werden:

```
public Buch findeBuchMitTitel (String titel) {
   int index = findeIndexZumTitel(titel);
   if (index == -1)
            // Buch nicht enthalten:
      return null;

   return regal[index];
}

public Buch entferneBuchMitTitel (String titel) {
   int buchIndex = findeIndexZumTitel(titel);
   if (buchIndex == -1)
            // Buch nicht enthalten:
      return null;

            // Buch ist gefunden:
   Buch buch = regal[buchIndex];

      // Alle Buecher rechts vom Buch um eins nach links
         verschieben;
      // Schleife muss dabei von vorn nach hinten laufen:
   for (int index = buchIndex; index < anzahlBuecher; index++)
      regal[index] = regal[index+1];

            // anzahlBuecher konsistent erniedrigen:
```

```
        anzahlBuecher--;
                // gefundenes Buch ausliefern:
        return buch;
    }
```

b) Zunächst wird eine Hilfsmethode `compareAutorTitel()` angegeben, die Autoren und gegebenenfalls Titel zweier Bücher vergleicht und feststellt, welches der beiden Bücher vor dem anderen stehen muss. Das Ergebnis der Methode ist analog zur Methode `compareTo()` auf der Klasse `String` zu interpretieren.

```
    private int compareAutorTitel(Buch b1, Buch b2) {
        int result = b1.getAutor().compareTo(b2.getAutor());
                // Bei gleichem Autor den Titel vergleichen
        if (result == 0) {
            return b1.getTitel().compareTo(b2.getTitel());
        }
        return result;
    }
```

In der Methode `druckeAutorTitelVerzeichnis()` werden die Referenzen auf die Bücher zunächst in eine zweite Reihung `autorenReihung` kopiert. Diese Reihung wird mit dem Sortieralgorithmus *selection sort* sortiert. Anschließend wird die Reihung durchlaufen und dabei zu jedem Buch Autor und Titel ausgedruckt.

```
    public void druckeAutorTitelVerzeichnis () {
        // eine neue Reihung fuer neue Referenzen auf die Buecher:
        Buch[] autorenReihung = new Buch[anzahlBuecher];

        // die Verweise auf die Buecher werden kopiert:
        for (int index = 0; index<anzahlBuecher; index++)
            autorenReihung[index] = regal[index];

        // Selection Sort aus der Vorlesung:
        for (int out = 1; out < anzahlBuecher; out++) {
            Buch temp = autorenReihung[out];
            int in = out;
            while (in > 0 && compareAutorTitel(autorenReihung[in-1],
                temp) > 0) {
                autorenReihung[in] = autorenReihung[in-1];
                in = in-1;
            }
            autorenReihung[in] = temp;
        }

        System.out.println("Verzeichnis der Autoren und Buchtitel:")
            ;
        for (int index=0; index<anzahlBuecher; index++) {
            System.out.println(autorenReihung[index].getAutor() + ": "
                +
                                autorenReihung[index].getTitel());
        }
    }
```

Eine andere Variante wäre es, die Reihung `autorenReihung` als neues Attribut der Klasse einzuführen. Dann wäre es Aufgabe der Methoden `stelleEin()` und `entferneBuchMitTitel()`, diese Reihung konsistent zu `regal` mit zu verwalten.

Wenn man wie mit der Reihung `autorenReihung` eine weitere Zugriffsmöglichkeit auf Daten erhält, so nennt man dies auch einen weiteren *Index* für die Daten. Diese Technik wird häufig bei Datenbanken benutzt, um schnell Daten nach unterschiedlichen Kriterien aufzufinden. Auch dort wird der weitere Index entweder dann erzeugt, wenn er benötigt wird, oder er wird konsistent mitgeführt. Es muss im Einzelfall entschieden werden, welche Variante die günstigere ist.