Technische Universität München
Fakultät für Informatik
Prof. Dr. A. Knoll

SS 2006
Übungsblatt 10
14. Juli 2006

Übungen zu Einführung in die Informatik I

Aufgabe 41 Modellierung und Implementation eines Bücherregals

In Aufgabe 42 von Übungsblatt 10 haben wir bereits die Klasse `Buch` modelliert und implementiert. Nun wollen wir dazu ein Bücherregal realisieren.

a) *„Mein Bücherregal kann leer sein oder eine Anzahl von Büchern enthalten. Wenn es voll ist, kann es erweitert werden. In mein Bücherregal kann man ein neues Buch einstellen, nach einem Buchtitel suchen und ein Buch mit einem gewünschten Buchtitel entnehmen. Außerdem hat mein Bücherregal ein Verzeichnis der enthaltenen Buchtitel."*

Identifizieren Sie aus dieser Beschreibung eines Bücherregals die Dienste, die eine Klasse `BuecherRegal` zur Verfügung stellen sollte, überlegen Sie sich, welche Attribute zur Bereitstellung dieser Dienste nötig sind, und geben Sie einen detaillierten Entwurf der Klasse in UML an.

b) **(P)** Implementieren Sie die Klasse `BuecherRegal` in Java. Stützen Sie sich dabei auf den Reihungstyp `Buch[]` und entscheiden Sie sich für eine von mehreren möglichen Varianten bei den Fragen,

- wie Sie ein leeres Regal füllen (z. B. von links, von rechts oder ohne eine solche Ordnung),
- ob das Regal nach der Entnahme von Büchern Lücken aufweisen soll oder nicht und wann Sie gegebenenfalls entstandene Lücken wieder schließen,
- ob beim vollen Regal die Erweiterung explizit angefordert werden muss oder ob die Erweiterung automatisch beim Einstellen des nächsten Buches erfolgt.

Aufgabe 42 Implementation des Bücherregals auf sortierter Reihung

In der vorhergehenden Aufgabe haben wir die Klasse `BuecherRegal` modelliert und implementiert. Nun wollen wir für diese Klasse eine neue Implementation angeben, die sich auf Reihungen abstützt, in denen die Bücher lexikographisch nach dem Titel sortiert werden. Die neue Implementation soll den Klassennamen `BuecherRegalSortiert` erhalten.

a) Ändern Sie die Implementation der Operationen `stelleEin()`, `findeBuchZumTitel()` und `entferneBuchZumTitel()` so, dass die Sortiertheit der Bücher sichergestellt bzw. zum schnellen Auffinden der Bücher ausgenutzt wird.

b) Implementieren Sie eine weitere Operation `druckeAutorenTitelVerzeichnis()`, die zu jedem Buch den Autor und den Titel ausdruckt. Dabei sollen die Autoren in lexikographischer Reihenfolge erscheinen. Falls es mehrere Bücher zum selben Autor gibt, sollen diese nach dem Titel sortiert ausgedruckt werden.

Hinweise:

- Für Vergleiche von Objekten der Klasse `String` steht Ihnen in Java die Instanz–Methode `int compareTo(String s)` zur Verfügung: Sind `string1` und `string2` zwei Objekte vom Typ `String`, dann liefert der Aufruf `string1.compareTo(string2)` einen Wert kleiner, gleich oder größer Null, falls `string1` lexikographisch kleiner, gleich oder größer als `string2` ist.
- Auf der Homepage des Programmierpraktikums finden Sie zu diesem Übungsblatt eine Datei `Buecher.txt`, die eine Folge von Aufrufen der Methode `stelleEin()` enthält. Sie können den Inhalt dieser Datei an eine geeignete Stelle Ihrer `main`–Methode kopieren, um eine „realistische Umgebung“ zum Testen Ihrer Methoden zu erhalten, ohne sebst unzählige Autorennamen und Buchtitel eingeben zu müssen.

Aufgabe 43 (H) Java & UML: Prozessverwaltung in Betriebssystemen (10 Punkte)

Moderne multiuser- und multitasking-fähige Betriebssysteme führen aus Sicht des Benutzers eine grosse Anzahl von Programmen quasigleichzeitig aus. Dies wird bewerkstelligt, indem der Prozessor zyklisch jedes in Ausführung befindliche Programm (Prozess) jeweils für eine kurze Zeit abarbeitet. Die Zuteilung ablaufbereiter Prozesse dem Prozessor übernimmt ein bestimmter Teil des Betriebssystems, der sog. Scheduler. In dieser Aufgabe soll ein stark vereinfachtes Modell des Prozesswechsels (Schedulings) in Java und UML realisiert werden. Nehmen Sie folgende Spezifikation als Grundlage Ihres Designs:

Ein Prozess wird durch seinen Namen und durch die zur Erledigung notwendigen Anzahl von Prozessortakten (in der Praxis unbekannt!) charakterisiert. Er ist beim Aufruf mit 90%-er Wahrscheinlichkeit ablaufbereit. Falls er nicht ablaufbereit ist (wartet beispielsweise auf E/A), so wird zum nächsten Prozess gewechselt. Während seines Ablaufs wird er mit 25%-er Wahrscheinlichkeit vor Ablauf seiner vom Scheduler zugeteilten Prozessortakte unterbrochen, und es kommt zum Prozesswechsel. Während des Ablaufs gibt jeder Prozess in jedem Takt eine kurze Textmeldung ($''tick''$) aus. Nach Verbrauch zugeteilter Prozessortakte oder bei Unterbrechung, signalisiert der Prozess dem Scheduler durch einen geeigneten Rückgabewert, ob er seine Aufgabe komplett abgearbeitet hat, oder nochmal aufgerufen werden muss.

Das Betriebssystem instantiiert einen Scheduler, übergibt ihm einige ablaufbereite Prozesse und stösst die Abarbeitung der Prozesse durch den Scheduler an. Ein Scheduler wird zum Einen über die maximal erlaubte Anzahl von Prozessen, und zum Anderen über die Anzahl von maximal zur Verfügung stehenden Prozessortakten bis zu einem Prozesswechsel charakterisiert. Der Scheduler arbeitet solange seine Prozessliste ab, bis jeder Prozess die zur Erledigung notwendigen Anzahl von Prozessortakten zugeteilt bekommen hat.

Aufgabenstellung:

a) Stellen Sie die Klassen `Betriebssystem`, `Scheduler` und `Prozess` in UML dar und definieren Sie sie in Java. Geben Sie die laut Spezifikation notwendigen Konstruktoren an.

b) Simulieren Sie den Prozesswechsel für drei Prozesse mit unterschiedlichen Laufzeiten. Vergewissern Sie sich, dass Ihr Programm die korrekte Anzahl ”ticks” ausgibt und terminiert.