RWTH Aachen
Lehrgebiet Theoretische Informatik
Frohn–Reidl–Rossmanith–Sánchez–Ströder

WS 2013/14
Übungsblatt 6
02.12.2013

Übung zur Vorlesung Programmierung

Aufgabe T12

Implemetieren sie eine Klasse *Set*, um eine Menge zu modellieren. Benutzen Sie dafür unverändert die gegebenen Klassen *AbstractIterableSet* und *SimpleFunctionalSet*. Diese Klassen folgen der Idee aus Aufgabe T7, in welcher Mengen durch eine Liste von Einfüge- und Löschoperationen dargestellt werden. Ihre neue Implementierung soll folgendermaßen aussehen:

a) Implementieren Sie die generische Klasse *EmptySet*$\langle E \rangle$ als Unterklasse von *SimpleFunctionalSet*$\langle E \rangle$, um die leere Menge darzustellen.

b) Implementieren Sie die generische Klasse *AddSet*$\langle E \rangle$ als Unterklasse von *SimpleFunctionalSet*$\langle E \rangle$, um das Hinzufügen eines Elements zu modellieren.

c) Implementieren Sie die generische Klasse *RemoveSet*$\langle E \rangle$ als Unterklasse von *SimpleFunctionalSet*$\langle E \rangle$, um das Löschen eines Elements zu modellieren.

d) Schreiben Sie eine generische Klasse *FunctionalSet*$\langle E \rangle$ als Unterklasse von *AbstractIterableSet*$\langle E \rangle$. Da die Oberklasse das Interface *java.util.Set* implementiert, muss Ihre Klasse die noch fehlenden Methoden *add*, *clear*, *contains*, *iterator* und *remove* implementieren. Ihre Implementierung soll das Hinzufügen beliebiger Elemente (inklusive *null*) unterstützen. Entwerfen Sie für die *iterator* Methode eine weitere generische Klasse *FunctionalSetIterator*$\langle E \rangle$, welche das Interface *java.util.Iterator*$\langle E \rangle$ implementiert.

Achten Sie darauf, dass Sie passende Konstruktoren explizit angeben müssen, und beachten Sie die Prinzipien der Datenkapselung.
Zum Testen Ihrer Implementierung können Sie die Datei *SetTest.java* von unserer Webseite benutzen. Der Aufruf Ihrer *main*-Methode sollte folgende Ausgabe auf der Konsole erzeugen:

```
{}
{1}
{2,1}
{3,2,1}
{3,2,1}
{3,2}
{3}
{3}
{}
{null}
{1}
```

```
{2,1}
{2,1}
{2,1}
{2}
{2}
{5,2,3,null}
{5,2,3,null}
{5,3,null}
{5,3,null}
{5,3}
```

Aufgabe T13

Ein *Binärbaum* ist eine Datenstruktur, in welcher Zahlen (oder andere geordnete Elemente) gespeichert werden. Jeder Baumknoten enthält dabei ein Element x sowie zwei Referenzen zu Unterbäumen; im linken Teilbaum sind dabei Elemente gespeichert, die strikt kleiner als x sind, im rechten Teilbaum diejenigen, die strikt größer als x sind. Jedes Element kann höchstens einmal im Baum enthalten sein. Implementieren Sie folgende Methoden der Klassen *BinaryTree* und *BinaryTreeNode*:

- **boolean** *contains*(**int** v): gibt *true* genau dann zurück, wenn der Binärbaum das Element v enthält.
- **void** *insert*(**int** v): fügt das Element v in den Baum ein, wenn es nicht schon darin enthalten ist.

```
public class BinaryTree {
  private BinaryTreeNode root;
  public BinaryTree() {
    root = null;
  }
  public boolean contains(int v) {// TODO }
  public void insert(int v) {// TODO }
}
private class BinaryTreeNode {
  private int value;
  private BinaryTreeNode left, right;
  public BinaryTreeNode(int v) {
    value = v;
    left = right = null;
  }
  public boolean contains(int v) {// TODO }
  public void insert(int v) {// TODO }
}
```

Aufgabe H12 (5 Punkte)[1]

Sie finden eine leichte Abwandlung der Listen-Klasse aus der Vorlesung auf unserer Webseite unter dem Namen *IntList*. Entwerfen Sie ein Interface *IntListVisitor* und erweitern Sie die Klasse *IntList* um eine Methode **void** *accept*(*IntListVisitor visitor*), um im Sinne des Visitor-Patterns ein solches Objekt entgegenzunehmen.

Implementieren Sie anschließend eine Klasse *SummingIntListVisitor*, welche dieses Interface implementiert, um die Summe einer Liste mit Hilfe der *accept*-Methode zu berechnen.

Aufgabe H13 (2 + 15 Punkte)[1]

In dieser Aufgabe geht es um die Implementierung einer Datenstruktur für Mengen, welche in das bestehende Collections Framework eingebettet werden soll. Sie benötigen dafür die auf unserer Webseite verfügbare abstrakte Klasse *AbstractIterableSet*. Alle in dieser Aufgabe zu implementierenden Klassen sollen nicht abstrakt sein und die vorgegebene abstrakte Klasse soll nicht verändert werden.

Die hier zu implementierende Mengenstruktur basiert auf einer Liste von Elementen mit einer *active* Markierung vom Typ **boolean**. Nur solche Elemente sind in der Menge enthalten, deren *active* Markierung den Wert *true* hat. Um Elemente aus der Menge zu löschen, werden keine Objekte aus dieser Liste entfernt, sondern lediglich die entsprechende Markierung auf *false* gesetz. Um Duplikate zu vermeiden, soll beim Einfügen die Liste durchsucht werden und für ein ggf. bereits gelöschtes Element, das wieder eingefügt wird, schlicht die entsprechende Markierung wieder auf *true* gesetzt werden. Beispielsweise kann die Menge $\{1, 2, 3\}$ als die Liste 3 *true*, 2 *true*, 1 *true* dargestellt werden. Löscht man nun das Element 2, so ergibt sich die Liste 3 *true*, 2 *false*, 1 *true*. Ein erneutes Einfügen des Elementes 2 liefert wieder die ursprüngliche Liste. Einfügen von bereits vorhanden Elementen oder Löschen von nicht vohandenen Elementen soll die Datenstruktur unverändert lassen—lediglich die Methode *clear* soll tatsächlich Objekte aus der Liste entfernen.

a) Schreiben Sie eine generische Klasse *SetNode*$\langle E \rangle$ mit drei Attributen *active* vom Typ **boolean**, *element* vom Typ E und *next* vom Typ *SetNode*$\langle E \rangle$. Implementieren Sie Getter-Methoden für alle Attribute und ermöglichen Sie beliebige Veränderungen am Attribut *active* (beachten Sie dabei die Prinzipien der Datenkapselung).

b) Schreiben Sie eine generische Klasse *FlagSet*$\langle E \rangle$ als Unterklasse der bereitgestellten Klasse *AbstractIterableSet*$\langle E \rangle$ mit genau einem Attribut *head* vom Typ *SetNode*$\langle E \rangle$. Da die Oberklasse das Interface *java.util.Set* implementiert, benötigt Ihre Klasse die noch fehlenden Methoden *add*, *clear*, *contains*, *iterator* und *remove*. Nutzen Sie dazu die in der vorherigen Teilaufgabe implementierte Klasse *SetNode*. Ihre Menge soll das Hinzufügen von beliebigen Elementen (inklusive *null*) unterstützen und nur dann ein neues *SetNode*-Objekt anlegen, wenn ein Element in eine Menge eingefügt wird, das noch nie in dieser Menge enthalten war (bis zurück zur Erstellung der Menge oder zum letzten Aufruf von *clear*). Benutzen Sie die *active* Markierungen in den *SetNode*-Objekten, um, wie am Anfang dieser Aufgabe beschrieben, den Inhalt der Menge zu verwalten. Implementieren Sie für die Methode *iterator* eine weitere generische Klasse *FlagSetIterator*$\langle E \rangle$, welche das Interface *java.util.Iterator*$\langle E \rangle$ und somit die Methoden *hasNext*, *next* und *remove* sinnvoll implementiert.

Die gewünschte Funktionalität der Methoden aus den Interfaces *Set* und *Iterator* schlagen Sie bitte in der Java-API nach.

Zum Testen Ihrer Implementierung können Sie ebenfalls die Datei *SetTest.java* von unserer Webseite benutzen, indem Sie alle Vorkommen von *FunctionalSet* durch *FlagSet* ersetzen. Der Aufruf Ihrer *main* Methode sollte die gleiche Ausgabe wie in der Tutoraufgabe T12 erzeugen.

Hinweis: Für den Iterator könnte es hilfreich sein, das nächste zurückzuliefernde und das zuletzt zurückgelieferte Element zwischenzuspeichern.

Abgabe zum 10.12.2013

[1] Bitte Quelltext für die Abgabe ausdrucken und zusätzlich per E-mail an den jeweiligen Tutor senden.