Karlsruher Institut für Technologie

Programmieren – Sommersemester 2013
Forschungsgruppe Verifikation trifft Algorithmik
`http://baldur.iti.kit.edu/programmieren-ss13`
Dr. Carsten Sinz · Markus Iser · Florian Merz

Übungsblatt 3

Ausgabe: 27.05.2013 – 13:00
Abgabe: 10.06.2013 – 13:00

Allgemeine Hinweise

- Achten Sie darauf nicht zu lange Zeilen, Methoden und Dateien zu erstellen[1][2]
- Programmcode und Kommentare müssen in englischer Sprache verfasst sein
- Kommentieren Sie Ihren Code angemessen: So viel wie nötig, so wenig wie möglich
- Wählen Sie geeignete Sichtbarkeiten für Ihre Klassen, Methoden und Attribute
- Verwenden Sie keine Klassen der Java-Bibliotheken ausgenommen Klassen des Pakets `java.lang`
- Achten Sie auf fehlerfrei kompilierenden Programmcode[1]
- Halten Sie alle whitespace Regeln ein[1][2]
- Halten Sie die Regeln zu Variablen-, Methoden und Paketbenennung ein und wählen Sie aussagekräftige Namen[2]

Endliche und Begrenzte Mengen

In dieser Übung sollen sie eine Klasse `BoundedSet` implementieren, die zur Darstellung von Mengen im mathematischen Sinn verwendet werden kann, sofern die Menge entweder nur endlich viele Elemente enthält, oder nur endlich viele Elemente *nicht* enthält. Insbesondere sollen also auch Mengen der Form „Alle Elemente des Universums bis auf die Zahlen 1 und 3“ darstellbar sein. Ihre Implementierung soll nur Elemente des Typs `int` aufnehmen können.

A Endliche Mengen (10 Punkte)

Entwerfen Sie zunächst eine Klasse `FiniteSet`, welche eine *endliche* Menge modelliert. Verwenden Sie für die interne Darstellung in `FiniteSet` ein Array. Auf keinen Fall dürfen Sie Klassen aus `java.util` verwenden. Ihre Klasse soll (mindestens) die folgenden Methoden bieten:

`public FiniteSet(int maxSize)`
: Konstruktor, der die maximale Anzahl an Elementen festlegt, welche die Menge enthalten kann. Initial ist die erzeugte Menge leer.

[1] Der Praktomat wird die Abgabe zurückweisen, falls diese Regel verletzt ist.
[2] Die Einhaltung dieser Regel wird durch das checkstyle Tool überwacht. Diese sind auf `http://baldur.iti.kit.edu/programmieren-ss13` erhältlich.

`public void insert(int elem)`
: Fügt die Zahl `elem` in die Menge ein, falls diese noch nicht enthalten ist. Falls das Array zu klein ist, legen Sie ein neues um den Faktor zwei größeres Array an.

`public void remove(int elem)`
: Löscht die Zahl `elem` aus der Menge, sofern diese in ihr enthalten ist.

`public boolean contains(int elem)`
: Gibt `true` zurück, genau dann wenn `elem` in der Menge enthalten ist.

`public FiniteSet intersect(FiniteSet m)`
: Bildet den (mathematischen) Schnitt der aktuellen Menge mit `m` und gibt diesen *als neues* Objekt des Typs `FiniteSet` zurück. Die maximale Größe des erzeugten Objekts soll das Minimum der maximalen Größen der beteiligten Mengen sein.

`public FiniteSet union(FiniteSet m)`
: Bildet die (mathematische) Vereinigung der aktuellen Menge mit `m` und gibt diese *als neues* Objekt des Typs `FiniteSet` zurück. Die maximale Größe des erzeugten Objekts soll die Summe der maximalen Größen der beteiligten Mengen sein.

`public FiniteSet difference(FiniteSet m)`
: Bildet die Differenz der aktuellen Menge und der Menge `m` und gibt diese *als neues* Objekt des Typs `FiniteSet` zurück. Die maximale Größe des erzeugten Objekts soll die maximale Größe der aktuellen Menge sein.

`public boolean equals(FiniteSet m)`
: Prüft, ob die aktuelle Menge gleich der Menge `m` ist und gibt im positiven Fall `true` zurück. Gleichheit ist hierbei im mathematischen Sinne zu verstehen.

`public String toString()`
: Liefert eine `String`-Darstellung der aktuellen Menge.

B Begrenzte Mengen (8 Punkte)

Aufbauend auf `FiniteSet` entwerfen Sie nun die Klasse `BoundedSet`. Diese Klasse soll die gleiche Funktionalität bieten wie `FiniteSet`. Dabei ist zu beachten, dass die Parameter- und Rückgabetypen der Methoden entsprechen angepasst werden müssen. Ebenso werden die Größenbeschränkungen bei den binären Operationen abhängig davon, ob die beteiligten `BoundedSet`-Objekte momentan eine endliche Menge darstellen, oder deren Komplement. Für die Implementierung von `BoundedSet` verwenden Sie ein `FiniteSet`-Objekt und ein `boolean`-Flag, das anzeigt, ob das `FiniteSet`-Objekt gerade die in der Menge enthaltenen oder die von der Menge ausgeschlossenen Werte speichert.

Zusätzlich soll die folgende Methode implementiert werden:

`public void complement()`
: Invertiert die aktuelle Menge. Durch diese Operation wird also zwischen der „Nur die Elemente x, y, z“- und der „Alle Elemente bis auf x, y, z“-Darstellung umgeschaltet.

C Tests (2 Punkte)

Schreiben Sie eine Testklasse, die das Verhalten einiger `BoundedSet`-Objekte ausführlich testet. Geben Sie zusätzlich das Protokoll Ihres Testlaufes als `Tests.txt` mit ab.

Kleine Mengenlehre

In dieser Aufgabe Betrachten wir Mengen von ganzen Zahlen. Wir bezeichnen die ganzen Zahlen ($\mathbb{Z} = \{..., -2, -1, 0, 1, 2, 3, ...\}$) deswegen als unser Universum U.

Zur Erinnerung:

- Für die Differenz $M_1 - M_2$ zweier Mengen M_1 und M_2 gilt:

$$m \in (M_1 - M_2) \Longleftrightarrow m \in M_1 \wedge m \notin M_2$$

- Das Komplement $\overline{M}$ einer Menge M ist $\overline{M} = U - M$

Es gelten außerdem die folgenden Gleichungen für beliebige Mengen M_1 und M_2:

$$\begin{aligned} M_1 \cap M_2 &= M_2 \cap M_1 \\ M_1 \cup M_2 &= M_2 \cup M_1 \end{aligned}$$

$$\begin{aligned} \overline{M_1} \cap \overline{M_2} &= \overline{M_1 \cup M_2} \\ \overline{M_1} \cup \overline{M_2} &= \overline{M_1 \cap M_2} \\ \overline{M_1} \cap M_2 &= M_2 - M_1 \\ \overline{M_1} \cup M_2 &= \overline{M_1 - M_2} \end{aligned}$$

$$\begin{aligned} \overline{M_1} \cup \{a\} &= \overline{M_1 - \{a\}} \\ \overline{M_1} - \{a\} &= \overline{M_1 \cup \{a\}} \end{aligned}$$

$$\begin{aligned} \overline{M_1} - M_2 &= \overline{M_1 \cup M_2} \\ M_1 - \overline{M_2} &= M_1 \cap M_2 \\ \overline{M_1} - \overline{M_2} &= M_2 - M_1 \end{aligned}$$