technische universität dortmund

Fakultät für Informatik
Lehrstuhl 11

Mart Hagedoorn
Guangping Li

Sommersemester 2024
24. Mai

DAP2 Praktikum – Blatt 7

Abgabe: KW 22

Studienleistung

- Zum Bestehen des Praktikums muss jeder Teilnehmer*innen die folgenden Leistungen erbringen:
  - Es müssen mindestens 50 Prozent der Punkte in den Kurzaufgaben erreicht werden.
  - Es müssen mindestens 50 Prozent der Punkte in den Langaufgaben erreicht werden.
- Im Krankheitsfall kann ein Testat bei Vorlage eines Attests in der folgenden Woche nachgeholt werden.
- Wenn ein Praktikumstermin auf einen Feiertag fällt, müssen Sie sich an einem beliebigen anderen Praktikumstermin in der gleichen Woche testieren lassen.
- **Hinweis:** Notieren Sie sich Ihre Punkte nach jedem Testat! Dies dient der eigenen Kontrolle. (Ihr Punktestand kann Ihnen während des Semesters nicht genannt werden.)

Wichtige Information (im Moodle verfügbar)

- Beachten Sie die Erklärung des **Ablaufs (Blatt A)**.
- Beachten Sie die **Regeln und Hinweise (Blatt R)** in der aktuellsten Version!
- Beachten Sie die **Hilfestellungen (Blatt H)** in der aktuellsten Version!

Kurzaufgabe 7.1: Hashing (6 Punkte)

In dieser Aufgabe sollen Sie eine simple Hashtabelle implementieren. Zunächst können Sie davon ausgehen, dass die Schlüssel vom Typ `Integer` sind. In Aufgabe 7.2 verallgemeinern Sie die Implementierung für beliebige Datentypen.

- Ein Objekt vom Typ `B7A1` verwaltet eine Menge von Schlüssel-Wert-Paaren, wobei Schlüssel und Werte vom Typ `Integer` sind. Legen Sie in `B7A1` eine private innere Klasse `Pair` an, sodass ein `Pair`-Objekt genau ein Schlüssel-Wert-Paar beinhaltet.
- Die Hashtabelle ist als Array (oder `ArrayList`) mit n Einträgen organisiert, wobei jeder Eintrag eine verkettete Liste von `Pair`-Objekten enthält. (Die verketteten Listen dienen der Kollisionsauflösung die Hashtabelle soll so funktionieren, wie in der Vorlesung beschrieben.) Der Konstruktor von `B7A1` erwartet n als einzigen Parameter.
- Erstmal können Sie einfach die naive Hashfunktion (`key` mod n) verwenden. Diese Funktion ist sehr schlecht: Stammen die Schlüssel etwa aus dem Universum $\{n \cdot k \mid k \in \mathbb{N}\}$, dann werden alle Paare in der gleichen verketteten List abgelegt. Legen Sie eine private Hilfsfuntkion an, welche aus dem Schlüssel die Adresse der verketteten Liste bestimmt:

  ```
  private int addressOfList(Integer key)      // welche Liste ist zuständig?
  ```

  Hinweis: Der Operator `%` ist in Java streng genommen nicht der Modulo-Operator und kann ein negatives Ergebnis zurückgeben. Implementieren Sie stattdessen einen eigenen Modulo Operator: z.B. `Math.floorMod(key, n)`.
- Die Grundoperationen der Hashtabelle sind Einfügen, Auslesen, und Entfernen. Wenn beim Einfügen bereits ein Paar mit gleichem Schlüssel existiert, wird der Wert einfach überschrieben. Wenn beim Auslesen kein Paar mit dem gewünschten Schlüssel existiert, wird `null` zurückgegeben. Wenn beim Löschen kein Paar mit dem gewünschten Schlüssel existiert, wird `false` zurückgegeben (und ansonsten `true`).

  ```
  public void insert(Integer key, Integer value)          // füge Paar ein
  public Integer get(Integer key)                       // gebe Wert zurück
  public boolean remove(Integer key)                      // entferne Paar
  ```

  Bestimmen Sie die Adresse der zuständigen Liste immer mit `addressOfList`. Abgesehen vom Konstruktor und den Grundoperationen hat die Klasse **keine öffentlichen Methoden oder Attribute!**

Testausgaben des Programms:

```
java B7A1.java <Enter>
8 <Enter>
Enter command (i to insert, g to get, r to remove, q to quit): i 8 7 <Enter>
Inserted (8, 7)
Enter command (i to insert, g to get, r to remove, q to quit): g 9 <Enter>
Key not found.
Enter command (i to insert, g to get, r to remove, q to quit): g 8 <Enter>
Key: 8, Value: 7
Enter command (i to insert, g to get, r to remove, q to quit): r 8 <Enter>
Key 8 removed: true
Enter command (i to insert, g to get, r to remove, q to quit): g 8 <Enter>
Key not found.
Enter command (i to insert, g to get, r to remove, q to quit): q <Enter>
```

Hinweis: Sie dürfen die Bibliotheken `ArrayList`, `LinkedList`, `List`, und `ListIterator` aus `java.util`, sowie die Methode `java.lang.Math.floorMod` verwenden.

Hinweis: Sie sollen als Datentyp der Schlüssel und Werte wirklich die Klasse `Integer` verwenden, und nicht den primitiven Typen `int`. Mit der Wrapper-Klasse haben Sie zahlreiche Vorteile, z.B. dass Sie hilfreiche Methoden auf `Integer`-Objekten aufrufen können (beispielsweise `toString()`), dass Sie Referenzen auf `Integer`-Objekte an Methoden übergeben können, und insbesondere dass Sie die Wrapper-Klasse mit generische Klassen und Methoden verwenden können. Dies wird wichtig für Aufgabe 7.2 sein. Weitere Informationen zu den Wrapper-Klassen der primitiven Typen finden sie z.B. hier:

`https://www.w3schools.com/java/java_wrapper_classes.asp`

Kurzaufgabe 7.2: Generische Klassen (2 Punkte)

In dieser Aufgabe verallgemeinern Sie ihre Implementierung aus Aufgabe 7.1, sodass Sie nicht nur Schlüssel und Werte vom Typ `Integer` verwenden. Dazu machen Sie die Klasse `B7A1` zur generischen Klasse `B7A2<K, V>`, in welcher Schlüssel vom Typ `K` und Werte vom Typ `V` sind (hier sind die Namen von `K` und `V` eine Namenskonvention für Schlüssel-Wert-Paare, englisch *Key-Value-Pairs*). Die generischen Typen `K` und `V` sind Parameter, die bei der Instanziierung der Klasse durch beliebige Typen ersetzt werden können. Für diesen Aufgabe initialisieren Sie die Typen von Schlüssel und Wert mit `String`.

Natürlich können Sie jetzt nicht mehr einfach (`key` mod m) als Hashfunktion verwenden, denn der Modulo-Operator ist nicht für alle möglichen Schlüsseltypen `K` definiert. Verwenden Sie stattdessen erstmals (`key.hashCode()` mod m). In Java erbt jede Klasse von der Klasse `Object`, und die Klasse `Object` stellt die Methode `hashCode` bereit. Die Funktionalität ist wie folgt:

- Der Aufruf von `hashCode()` gibt einen `int`-Wert zurück. Wird die Methode wiederholt für das gleiche Objekt aufgerufen, wird jedes mal der gleiche Wert zurück gegeben.
- Zwei Objekte `a` und `b` mit `a.equals(b)==true` müssen den gleichen `hashCode` haben.
- Zwei Objekte `a` und `b` mit `a.equals(b)==false` dürfen den gleichen `hashCode` haben.

Viele Klassen überschreiben die `hashCode`-Funktion. Informieren Sie sich im Internet darüber, wie `hashCode` für die Wrapper-Typen der primitiven Klassen funktioniert. Handelt es sich dabei um "gute" Hashfunktionen?

Testausgaben des Programms:

```
java B7A2.java <Enter>
8 <Enter>
Enter command (i to insert, g to get, r to remove, q to quit): i 8 eight
Inserted (8, eight)
Enter command (i to insert, g to get, r to remove, q to quit): i ten 2
Inserted (ten, 2)
Enter command (i to insert, g to get, r to remove, q to quit): g 8
Key: 8, Value: eight
Enter command (i to insert, g to get, r to remove, q to quit): g ten
Key: ten, Value: 2
Enter command (i to insert, g to get, r to remove, q to quit): r 8
Key 8 removed: true
Enter command (i to insert, g to get, r to remove, q to quit): g 8
Key not found.
Enter command (i to insert, g to get, r to remove, q to quit): q
```

Hinweis: Generics kennen Sie hoffentlich bereits aus DAP1. Ansonsten finden Sie zahlreiche gute Tutorials und Beispiele im Internet, z.B.:

`https://www.geeksforgeeks.org/generics-in-java`
`https://www.w3schools.blog/generics-class-java`