Hochschule Furtwangen
Fakultät I
Prof. Dr. Lothar Piepmeyer

Praktikum ‚Algorithmen und Datenstrukturen'

Aufgabenblatt 9

Aufgabe 1:
Sie haben sich zuletzt in Aufgabenblatt 7 mit der Entwicklung einer eigenen Klasse `LinkedList` beschäftigt. Im Kapitel ‚Iteratoren' haben wir das Interface ‚Container' erweitert. Ihre Klasse ‚LinkedList' soll jetzt dieses *erweiterte* Interface implementieren. Insbesondere soll der folgende Code

```
Container<Integer> numbers=new LinkedList<>();
numbers.add(4);
numbers.add(7);
numbers.add(1);
numbers.add(1);
for (int number : numbers) {
    System.out.println(number);
}
```

Die Ausgabe

```
1
1
7
4
```

haben. Hört sich komplizierter an, als es ist. Bei mir hat die Lösung gut 20 ordentliche Zeilen.

Aufgabe 2:
Übernehmen Sie aus dem Kapitel 'Hash-Verfahren' den Code für die Klasse `HashTable`. Für den Datentyp `LinkedList`, der ja von `HashTable` genutzt wird, übernehmen Sie Ihre eigene Klasse. Auch hier muss, wie in Aufgabe 1, das *erweiterte* Interface `Container` implementiert werden. Die Implementierung von `Iterable` ist dabei nicht im Skript ausgeführt und bleibt Ihnen überlassen. Um Ihnen eine Orientierung zu geben: bei mir sind das gut 30 ordentliche Zeilen.
Der folgende Code

```
HashTable<Integer> numbers=new HashTable<>();
for (int i = 0; i < 10; i+=2) {
    numbers.add(i);
}
for(int number : numbers) {
    System.out.println(number);
}
```

muss

```
0
2
4
6
8
```

ausgeben.

Aufgabe 3:
a. In dieser Aufgabe beschäftigen Sie sich mit dem Unterschied zwischen guten und schlechten Hash-Funktionen. Entwerfen Sie dazu zunächst eine Klasse `MyString`, die einen Text als (selbstverständlich) privates Attribut hat. Überschreiben Sie die Methode `hashCode` so, dass Sie die Länge des Textes liefert. Vergessen Sie nicht auch die equals-Funktion zu überschreiben. Die Methoden `equals` und `hashCode` kommen eigentlich immer zu zweit.
Der folgende Code

```
MyString hello = new MyString("Hello HFU");
```

```
MyString moreHello = new MyString("Hello HFU");
System.out.println(hello.equals(moreHello));
System.out.println(hello.hashCode());
```

muss also

```
true
9
```

ausgeben.

b. Erzeugen Sie eine Hash-Tabelle mit 10.000 zufälligen Texten vom Typ `String` und eine mit 10.000 zufälligen Texten vom Typ `MyString`. Für jede der beiden Tabellen messen Sie, wie lange es dauert, um zu prüfen, ob alle Texte der Tabelle auch wirklich in der Tabelle vorhanden sind. Das können Sie wie folgt machen:

```
for (MyString text : table) {
      if(!table.contains(text))
          throw new AssertionError();
}
```

Auch wenn das auf den ersten Blick sinnfrei erscheint, ergeben die gemessenen Zeiten interessante Einblicke in die Wichtigkeit guter Hash-Funktionen - und wer weiß, vielleicht haben Sie einen Fehler gemacht und der Code wirft bei Ihnen einen Exception.
Hinweis: Zufällige Texte erzeugen Sie einfach, in dem Sie mit der Klasse `Random` (siehe Aufgabenblatt 8) ganze Zahlen oder Fließkommazahlen erzeugen und diese dann in Text umwandeln. Das ist nach dem folgenden Muster möglich

```
String text=4711+"";
```