Weitere Datenstrukturen: HashMap

Lernziele:

- Sie verstehen das Prinzip einer HashFunktion
- Sie kennen die *Map*-Datenstruktur und können es anwenden

1 Das Abbilden von Wertepaaren

Idee: Wir wollen jeweils ein Paar von Werten abspeichern. Unser erster Ansatz könnte ein Array sein, wo wir in jeder Stelle zwei Werte speichern:

{«yoghurt»;1.20, «brot»;2.30, «apfel»;0.80; ….. }

Wie Sie selber feststellen, wäre das vermutlich ein 2-dimensionaler Array. Wir würden bei einer Suche den jeweiligen Preis in O(log n) Zeit finden (falls die Liste sortiert ist).

Wenn wir aber jedes Element direkt finden möchten (also O(1) Zeit), dann brauchen wir einen «fixen» Draht zwischen dem Produkt und Preis. Die Abfrage eines Begriffs soll sofort den Preis liefern.

1.1 Verwendung von HashFunction = Hash Table

Dieser «fixer» Draht ist eine HashFunktion. Dh. Es wird zB. ein String eingegeben und man erhält direkt einen Wert. Die Funktion «mapped» einen Wert zu einem anderen. Oder anders ausgedrückt: ein Schlüssel wird mit einem Wert fix definiert.

Die Hash Funktion bildet stets einen Wert zu einem bestimmten Index. Wir haben somit eine Hash Tabelle (oder «Hash Table»).

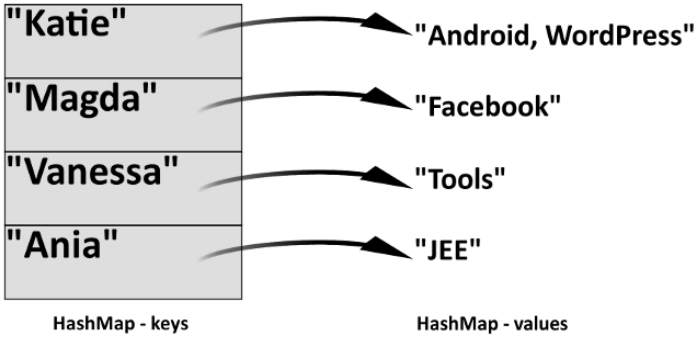
In Java werden Hash Tables «HashMap» genannt, in anderen Sprachen sind sie auch als «Dictionary» bekannt.

2 Die *HashMap* Datenstruktur

Der Hauptzweck der *HashMap* Datenstruktur ist es, *Werte* (*values*) mit *Schlüsseln* (*key*) zu verbinden. Sie unterstützt üblicherweise 2 Operationen:

- *einfügen* (*put*) eines neuen Paares
- *suchen* (*get*) nach dem Wert, der mit einem gegebenen Schlüssel verbunden ist

Diese Struktur wird sehr oft eingesetzt. Ein klassisches Beispiel ist das Telephonbuch (oder auch Kontakte auf Ihrem Smartphone).



Dabei muss darauf geachtet werden, dass es nicht zu doppelten Einträgen (doppelte Schlüssel) kommt. Somit wird folgende Regel implementiert:

- Jeder Schlüssel ist immer nur mit einem Wert verbunden
- Wenn ein Client ein Schlüssel-Wert-Paar einfügt, in der dieser Schlüssel (und ein damit verbundener Wert) bereits vorhanden ist, dann ersetzt der neue Wert den alten.

In Java wird das Interface *Map* in vielen verschiedenen Klassen implementiert (siehe dazu Java API). Eine sehr oft verwendete Klasse ist der *HashMap*.

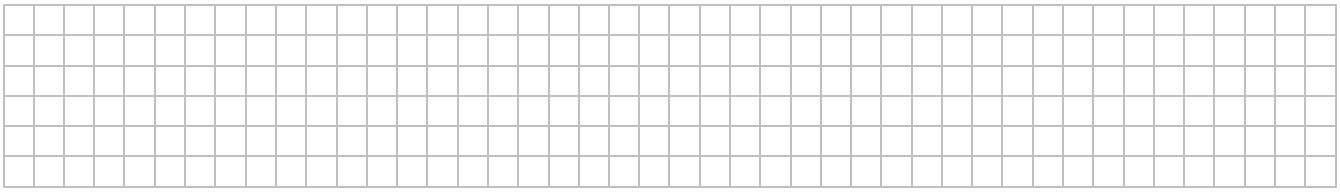
Code-Beispiel:

```
HashMap<String, String> phoneBook =   new HashMap<String, String>();

phoneBook.put(„Lisa Jones“, „(402) 4536 4674“);
phoneBook.put(„Prince Harry“, „(0044)79854 4512“);
```

2.1 Aufgabe: *HashMap* für grössere Datenmengen verwenden

Wir wollen die *HashMap* für eine Liste von US-Flughäfen verwenden (siehe airports.cvs).
Schreiben Sie eine Klasse, welche diese Liste in ein HashMap einliest.
Erweitern Sie die Klasse, so dass nach einem Flughafen Code gesucht werden kann.
(entweder via Konsole oder GUI)



2.3 Für Fortgeschrittene: Tech-Support System implementieren

Erstellen Sie ein Tech-Support System, welches anhand von bestimmten Schlüsselwörtern, dem User eine „Hilfe" anbietet. Diese Applikation sollte folgende Klassen beinhalten:

InputReader: diese Klasse benützt die *Scanner*-Klasse, um die Eingabe einzulesen.

Responder: diese Klasse hat ein *HashMap* mit Schlüsselwörtern (*key*) und entsprechenden Antworten (*value*).

SupportSystem: hier wird Eingabe und Ausgabe gesteuert.
Diese Klasse benützt den *InputReader* für die Eingabe und den *Responder* für eine Antwort.

Ein Auszug der Klasse mit der Methode *start()*:

```
public class SupportSystem
{
    private InputReader reader;
    private Responder responder;

    …
    public void start()
    {
        boolean finished = false;

        printWelcome();

        while(!finished) {
```

```
            String input = reader.getInput();

            if(input.startsWith("bye")) {
                finished = true;
            }
            else {
                String response = responder.generateResponse();
                System.out.println(response);
            }
        }
        printGoodbye();
    }
```