Hashtabellen und Suchbäume

- Die Schnittstelle `Set` der Mengen
- Die Schnittstelle `Map` der endlichen Abbildungen
- Hashtabellen
- Binäre Suchbäume
- Balancierte Bäume (AVL-Bäume)
- B-Bäume

Mengen

Grundlegende Operationen auf einer Menge:

- Element hinzufügen
- Element entfernen
- Test auf Elementschaft
- Alle Elemente ausgeben (in beliebiger Ordnung)

In Java gibt es die Schnittstelle `Set`, die diese und andere Methoden definiert.

Sie wird durch die Klassen `HashSet` und `TreeSet` implementiert.

Anwendungsbeispiel

```
import java.util.*;
import javax.swing.JOptionPane;

public class SetTest
{
    public static void main(String[] args) {
        Set names = new HashSet();

        boolean done = false;
        while (!done) {
            String input = JOptionPane.showInputDialog(
                                "Add Name, Cancel when done.");
            if (input == null)
                done = true;
            else {
                names.add(input);
                print(names);}}
```

Anwendungsbeispiel

```
        done = false;
        while (!done) {
            String input = JOptionPane.showInputDialog(
                                "Remove Name, Cancel when done.");
            if (input == null)
                done = true;
            else {
                names.remove(input);
                print(names);
            }
        }
        System.exit(0);
    }
```

Anwendungsbeispiel

```
    private static void print(Set s) {
        Iterator iter = s.iterator();
        System.out.print("{");
        while (iter.hasNext()) {
            System.out.print(iter.next());
            System.out.print(" ");
        }
        System.out.println("}");
    }
}
```

NB: Außer beim Konstruktor verwenden wir die *Schnittstelle* `Set`, nicht die *Klasse* `HashSet`. So müssen wir nur eine Zeile ändern um auf `TreeSet` umzusteigen.

Abbildungen

Eine endliche Abbildung (engl. (finite) *map*) ist eine Zuordnung von *Schlüsseln* (*keys*) zu *Werten* (*values*).

- Zu einer Abbildung können Bindungen hinzugefügt oder entfernt werden.
- Zu jedem Schlüssel in der Definitionsmenge der Abbildung kann der zugehörge Wert ausgegeben werden.
- Zu jedem Schlüssel gibt es nur einen Wert. Fügt man eine neue Bindung mit demselben Schlüssel hinzu, so wird der alte Wert überschrieben.

Die Schnittstelle `Map` stellt diese und andere Funktionen zur Verfügung. Sie wird implementiert u.a. durch die Klassen `HashMap` und `TreeMap`.

Anwendungsbeispiel

```
import java.util.*;
import javax.swing.JOptionPane;
import java.awt.Color;

public class MapTest
{
    public static void main(String[] args) {
        Map favoriteColors = new HashMap();
        favoriteColors.put("Juliet", Color.pink);
        favoriteColors.put("Romeo", Color.green);
        favoriteColors.put("Adam", Color.blue);
        favoriteColors.put("Eve", Color.pink);
        print(favoriteColors);
        favoriteColors.put("Adam", Color.yellow);
        favoriteColors.remove("Romeo");
        print(favoriteColors);
    }
```

Anwendungsbeispiel

```
    private static void print(Map m) {
        Set keySet = m.keySet();
        Iterator iter = keySet.iterator();
        while (iter.hasNext()) {
            Object key = iter.next();
            Object value = m.get(key);
            System.out.println(key + "->" + value);
        }
    }
}
```

Hashwerte

Die Klasse `Object` enthält eine Methode

```
int hashCode();
```

Sie liefert zu jedem Objekt einen Integer, den *Hashcode* oder *Hashwert*.

Die Spezifikation von `hashCode` besagt, dass zwei im Sinne von `equals` gleiche Objekte denselben Hashwert haben.

Es ist aber erlaubt, dass zwei verschiedene Objekte denselben Hashwert haben. Das ist kein Wunder, denn es gibt ja "nur" 2^{32} `int` Werte.

Allerdings sorgt eine gute Implementierung von `hashCode` dafür, dass die Hashwerte möglichst breit gestreut (*to hash* = fein hacken) werden. Bei "zufälliger" Wahl eines Objekts einer festen Klasse sollen alle Hashwerte "gleich wahrscheinlich" sein.

Implementierung von `Set` als Hashtabelle

Eine Möglichkeit, eine Menge zu implementieren, besteht darin, ein Array `s` einer bestimmten Größe `SIZE` vorzusehen und ein Element `x` im Fach `x.hashCode() % SIZE` abzulegen.

Das geht eine Weile gut, funktioniert aber nicht, wenn wir zwei Elemente ablegen möchten, deren Hashwerte gleich modulo `SIZE` sind.

In diesem Falle liegt eine *Kollision* vor.

Um Kollisionen zu begegnen kann man in jedem Fach eine verkettete Liste von Objekten vorsehen.

Für `get` und `put` muss man zunächst das Fach bestimmen und dann die dort befindliche Liste linear durchsuchen.

Sind Kollisionen selten, so bleiben diese Listen recht kurz und der Aufwand hält sich in Grenzen.

Genaue stochastische Analyse erfolgt in “Effiziente Algorithmen”.

Implementierung von `Map` als Hashtabelle

Praktisch dieselbe Datenstruktur kann man auch für Abbildungen verwenden:

Die Bindung $k \mapsto x$ wird im Fach `k.hashCode() % SIZE`

Dadurch ist sichergestellt, dass zu jedem Schlüssel nur ein Eintrag vorhanden ist.