1.1 HashSet

1.1.1 Motivation

Im Gegensatz zu Listen kommen in Mengen keine gleichen Elemente mehrfach vor.
Es gibt also keine Duplikate.
Damit eine Menge ein Element (eines bestimmten Datentyps/Klasse TypX) nicht doppelt aufnimmt (z.B. mit der Methode add()), überprüft diese Methode, ob dieses Element schon vorkommt. Dazu werden die Methoden equals() und hashCode() benutzt:
Zuerst wird „auf die Schnelle“ überprüft, ob beide Objekte denselben hashCode (das ist ein int-Wert) besitzen.
Trifft dies zu, wird die langsamere equals-Methode bemüht, um die wirkliche Gleichheit zu testen.

1.1.1.1 Methode hashCode() und equals()

Die Methode hashCode liefert einen möglichst eindeutigen (aber nicht zwingend einmaligen) Wert in Form eines integer-Werts zur Identifikation des Inhalts eines Objekts zurück.
Er wird anhand der Attribute des Objekts berechnet.
1)
Wenn zwei Objekte einen unterschiedlichen hashCode besitzen, können Sie nicht gleich (bzgl. der Methode equals() !!) sein. Es völlig ausgeschlossen, dass die Objekte dennoch den selben Inhalt besitzen. Deshalb wird auf die umfangreiche Überprüfung mit equals verzichtet.

2)
Besitzen zwei Objekte einen identischen hashcode, können sie inhaltlich gleich (bzgl. der Methode equals() !!) sein, müssen aber nicht.
Außerdem sollte die Berechnung des hashCodes schnell möglich sein.

Konkret:
Ergibt der Vergleich zweier Objekte mit equals() den Wert true, dann muß der hashCode() der 2 Objekte gleich sein.
Ergibt der Vergleich zweier Objekte mit equals() den Wert false, dann kann der hashCode() der 2 Objekte gleich sein, muß aber nicht.

Beispiel:
Die Klasse Person besteht aus den 3 Attributen:
- String vorname,
- String nachname,
- int alter.

Der hashcode() wird z.B. nur aus vorname und nachname berechnet,
wie z.B. die Summe der Asciiwerte der Zeichen von vorname und nachname.
equals() wird auch nur genau aus vorname und nachname berechnet: Wenn diese jeweils gleich sind, wird genau dann true zurückgeliefert.
Man sieht:
Wenn equals() true zurückliefert, dann liefert hashcode() den gleichen Wert.
Wenn hashcode() den gleichen Wert liefert wie z.B. bei der Person mit
Vorname = "A" Nachname = "B" und der anderen Person:
Vorname = "B" Nachname = "A"
dann sind das verschiedene Personen mit dem gleichen hashcode().

1.1.1.2 Beispiel (Programmcode)

```
package demohashcode_10;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;


public class Startklasse {
    public static void main(String[] args) {
        ArrayList <Person> pl = new ArrayList <Person>();
        pl.add(new Person("A","B"));
        pl.add(new Person("B","A"));
        HashSet hs = new HashSet<Person>(pl);
        Iterator it = hs.iterator();
        // HashSet wird mit dem Iterator durchlaufen
        while (it.hasNext()){
            Person p;
            p=(Person)it.next();
            System.out.print(" Vorname="+p.getVorname());
            System.out.print(" Nachname="+p.getNachname());
            System.out.print(" hascode="+p.hashCode());
            System.out.println("");
        }
    }
}

class Person  {
    private String vorname;
    private String nachname;

    public Person(String vorname, String nachname) {
        this.vorname = vorname;
        this.nachname = nachname;
    }

    public String getVorname() {
        return vorname;
    }

    public void setVorname(String vorname) {
        this.vorname = vorname;
    }

    public String getNachname() {
        return nachname;
    }

    public void setNachname(String nachname) {
        this.nachname = nachname;
    }
```

```
    public boolean equals(Object obj){
        if(vorname.equals(((Person)obj).vorname) &&
           vorname.equals(((Person)obj).vorname)){
            return true;
        }
        else{
            return false;
        }
    }

    public int hashCode() {
        int i;
        int sum=0;
        for (i = 0; i < vorname.length(); i++) {
            sum = sum + (int)vorname.charAt(i);
        }
        for (i = 0; i < nachname.length(); i++) {
            sum = sum + (int)nachname.charAt(i);
        }
        return sum;
    }
}
```

1.2 Sortieren

1.2.1 Motivation

Wenn man Listen oder Felder sortieren will, kann man die Methode sort() benutzen.
Der zugehörige Algoritmus muß intern natürlich jeweils zwei Objekte vergleichen können.
Sie braucht also eine Methode "vergleiche()", mit deren Hilfe sie die Liste sortieren kann.
Wenn man z.B. eine List von Personen sortieren will, muß die Methode "vergleiche()" berechnen, wann eine Person beim Vergleich kleiner als eine andere Person ist.
Dazu könnte man z.B. den Nachnamen als lexikografische Vergleichsmöglichkeit nutzen (wie im Telefonbuch).
Dies kann durch ein Interface erzwungen werden, das z.B. die Klasse Person implementiert.
Dieses Interface verlangt dann, daß in der Klasse Person eine bestimmte Vergleichsmethode implementiert wird.

1.2.1.1 Beispiel (Programmcode)

```
package demosortieren_10;
import java.util.*;

public class Startklasse {
    public static void main(String[] args) {
        int i;
        ArrayList <Person> pl = new ArrayList <Person>();
        pl.add(new Person("Albert","Einstein"));
        pl.add(new Person("Nils","Bohr"));
        Collections.sort(pl);
        for(i=0;i<pl.size();i++){
            System.out.print(pl.get(i).getVorname()+" ");
            System.out.println(pl.get(i).getNachname());
        }

        // Es können auch Felder sortiert werden:
        Person[] personen = new Person[2];
        personen[0]=new Person("Lilo","Hermann");
        personen[1]=new Person("Ernst","Busch");
        // Achtung Array muß komplett gefüllt sein, ohne null
        Arrays.sort(personen);
        for(i=0;i<personen.length;i++){
            System.out.print(personen[i].getVorname()+" ");
            System.out.println(personen[i].getNachname());
        }

        // Alternative Möglichkeit:
        MeinVergleicher mv = new MeinVergleicher();
        Arrays.sort(personen, mv);
        for(i=0;i<personen.length;i++){
            System.out.print(personen[i].getVorname()+" ");
            System.out.println(personen[i].getNachname());
        }
    }
}
```

```
class Person implements Comparable  {
    private String vorname;
    private String nachname;

    public Person(String vorname, String nachname) {
        this.vorname = vorname;
        this.nachname = nachname;
    }

    public String getVorname() {
        return vorname;
    }

    public void setVorname(String vorname) {
        this.vorname = vorname;
    }

    public String getNachname() {
        return nachname;
    }

    public void setNachname(String nachname) {
        this.nachname = nachname;
    }

    public int compareTo(Object obj) {
        int erg;
        int wert1, wert2;

        wert1=(int)(nachname.charAt(0));
        wert2=(int)((Person) obj).nachname.charAt(0);
        if(wert1 < wert2){
            erg=-1;
        }
        else if(wert1==wert2){
            erg=0;
        }
        else{
            erg=1;
        }
        return erg;
    }
}
```

```
// Alternative Möglichkeit zum Sortieren
class MeinVergleicher implements Comparator{
    public int compare(Object o1, Object o2){
        int erg;
        int wert1, wert2;
        wert1=(int)(((Person)o1).getNachname().charAt(0));
        wert2=(int)(((Person)o2).getNachname().charAt(0));

        if(wert1<wert2){
            erg=-1;
        }
        else if(wert1==wert2){
            erg=0;
        }
        else{
            erg=1;
        }
        return erg;
    }
}
```