TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Computer Vision Group
Einführung in die Informatik I für Ingenieure (MSE)
Prof. Dr. Daniel Cremers, Mohamed Souiai

WS 2012
Übungsblatt 6
30.11.2012

Aufgabe 6.1 (Ü) Objekte verstehen

Diskutieren Sie, was in dem folgenden Programm passiert.

```
1  class A {
2      private int i;
3      private int j;
4      private B b;
5      public A(int i, int j, B b){
6        setIandJandB(i,j,b);
7      }
8      public void setIandJandB(int i,int j, B neuesB){
9          i=i;
10         this.j=j;
11         b=neuesB;
12     }
13     public String toString(){
14         return "Toll, ich bin A und habe "+i+" / "+j+" / "+b.toString()
               ;
15     }
16 }
17 class B  {
18     private String name;
19     private int i;
20     public B(String name, int i){
21       this.i=i;
22       this.name=name;
23     }
24     public String toString(){
25       return "Wow, ich bin B "+name+" mit i="+i;
26     }
27     public void increase(){
28       i++;
29     }
30     public boolean isSameName(String s){
31       return (s==name);
32     }
33     public boolean isReallyNotSameName(String s){
34       return !(s.equals(name));
35     }
36 }
37 public class OO{
38     public static void main(String[] args){
39       int i=1,j=2;
40       B b = new B("TestB",0);
```

```
41          A a = new A(i,j,b);
42          System.out.println(a.toString());
43          i++;
44          b.increase();
45          System.out.println(a.toString());
46          b = new B("hollaDieWaldfee",0);
47          System.out.println(a.toString());
48          String[] s = {"holla","Die","Waldfee"};
49          System.out.println(b.isSameName(s[0]+s[1]+s[2]));
50          System.out.println(b.isReallyNotSameName(s[0]+s[1]+s[2]));
51      }
52  }
```

Lösungsvorschlag 6.1

Das Programm besteht aus drei Klassen `A`, `B` und `OO`, wobei letztere nur die `main`-Methode bereitstellt. Ein Objekt der Klasse `A` besteht aus zwei `int`-Werten und einer *Referenz* auf ein Objekt der Klasse `B`. Ein Objekt der Klasse `B` besteht aus einer Referenz auf einen `String` und einem `int`-Wert.
Führt man das Programm aus, passiert Folgendes:

Zeile

40: Ein neues Objekt `b` vom Typ `B` mit dem String "TestB" und dem `int`-Wert 0 wird angelegt.

41: Ein neues Objekt `a` vom Typ `A` mit dem Wert 2 für j und dem neuen Objekt `b` wird angelegt. Der Parameter i hat keinen Einfluss auf das Objekt. Betrachten Sie genauer die Zuweisung `i = i` in Zeile 9: i ist in beiden Fällen die Variable, die als Parameter der Funktion übergeben wird. Diese Zuweisung tut also nichts. Gewünscht ist, dass der Wert des Parameters der Variablen i des Objekts zugewiesen wird. Das würde erreicht durch: `this.i = i;` (NetBeans zeigt für diese Zeile möglicherweise die Warnung "Assignment To Itself" an.)

42: Es wird die `String`-Repräsentation des Objekts `a` ausgegeben. Das heißt, der Aufruf `a.toString()` ruft die Methode der Klasse `A` auf. Das wiederum bedeutet, dass der String "`Toll, ich bin A und habe "+i+"/ "+j+"/ "+b;` ausgegeben werden soll. Aber was wird für die Variablen ausgegeben?

- i Das Objekt-Attribut i hat keinen Wert bekommen (s. Erklärung zu Zeile 41), daher wird der Defaultwert für `int` ausgegeben: 0.
- j Das Objekt-Attribut j hat den Wert 2, dieser wird ausgegeben.
- b `b` ist ein Objekt vom Typ `B`. Hier wird automatisch die Methode `toString()` auf `b` angewendet. Dabei handelt es sich um die Methode der Klasse `B` (Zeilen 24-26), es wird also "`Wow, ich bin B"+name+"mit i= "+i;` angehängt. Dabei ist `name` "TestB" und i hat den Wert 0. Insgesamt ist die Ausgabe also:
  `Toll, ich bin A und habe 0 / 2 / Wow, ich bin B TestB mit i=0`

43: Der Wert der Variablen `i` die in Zeile 39 angelegt wurde, wird um 1 erhöht. Das hat keinen Einfluss auf die Werte der Attribute der Objekte `b` und `a`.

44: Der Wert des Attributs `i` von Objekt `b` wird mittels der Methode `increase()` um 1 erhöht. Dies hat keinen Einfluss auf die Variable `i` in der `main`-Methode oder das Attribut `i` des Objekts `a`. Allerdings handelt es sich bei `b` auch um das Attribut `b` des Objekts `a`.

45: Durch Zeile 44 hat sich der Wert des Attributs `i` des Attributs `b` in Objekt `a` verändert. Der erneute Aufruf von `a.toString()` führt zu dieser Ausgabe:
`Toll, ich bin A und habe 0 / 2 / Wow, ich bin B TestB mit i=1`

46: Die Variable `b` bekommt ein neues Objekt vom Typ `B` zugewiesen. Dies hat keinen

Einfluss auf das Attribut `b` des Objekts `a`.

47: Durch Zeile 46 hat sich nichts an Objekt `a` verändert. Der erneute Aufruf von `a.toString()` führt wieder zu dieser Ausgabe:
`Toll, ich bin A und habe 0 / 2 / Wow, ich bin B TestB mit i=1`

48: Ein String-Array `s` mit den Strings "holla", "Die" und "Waldfee" wird angelegt.

49: Der Name des neuen Objekts `b` wird mittels der Methode `isSameName()` aus der Klasse `B` mit dem String verglichen, der aus dem Teilstrings in `s` zusammengesetzt wird. `b.name` ist "hollaDieWaldfee" und auch `s[0]+s[1]+s[2]` ist "hollaDieWaldfee". Dennoch gibt die Methode `false` zurück, da es sich um zwei unterschiedliche Objekte vom Typ `String` handelt. Strings lassen sich nicht (mit dem gewünschten Effekt) mit == oder != vergleichen. Benutzt man diesen Vergleich, wird nur überprüft, ob es sich tatsächlich um das selbe Objekt handelt. Gewünscht ist der Vergleich der einzelnen Buchstaben. Dafür muss die Methode `equals()` benutzt werden: `return s.equals(name);` (NetBeans zeigt für diese Zeile möglicherweise die Warnung "Comparing Strings using == or !=" an.)

50: Der Name des neuen Objekts `b` wird mittels der Methode `isReallyNotSameName()` aus der Klasse `B` mit dem String verglichen, der aus dem Teilstrings in `s` zusammengesetzt wird. `b.name` ist "hollaDieWaldfee" und auch `s[0]+s[1]+s[2]` ist "hollaDieWaldfee". Diese Methode gibt korrekt `false` zurück, da beide Objekte den gleichen `String` repräsentieren.

Möglicherweise zeigt NetBeans auch noch in den Zeilen 6, 13 und 24 Warnungen an ("Overridable method call in constructor" und "Add @Override Annotation"). Diese resultieren aus den Eigenschaften Polymorphie und Vererbung von Java. Dies war aber noch nicht Thema, daher ignorieren wir diese Warnungen (vorerst).

Aufgabe 6.2 (Ü) Vorlesung

Gegeben sei die Klasse `Student` mit dem privaten Attribut `int id`.

```
class Student{
    private int id;
    public Student(int id){
        this.id = id;
    }
    public int getId(){
        return id;
    }

}
```

a) Implementieren Sie die Klasse `Vorlesung`, die maximal n `Student`-Objekte aufnehmen kann. Die maximale Studentenanzahl soll dabei im Konstruktor übergeben werden.

b) Erweitern Sie die Klasse `Vorlesung` um eine Methode

```
boolean anmelden(Student s)
```

die einen Studenten `s` zu dieser Vorlesung anmeldet. Dabei soll die Methode `anmelden` den Wert `true` zurückliefern, wenn ein Student für die Vorlesung erfolgreich angemeldet werden konnte. Die Anmeldung desselben Studenten soll dabei keinen Effekt haben. Ansonsten soll die Methode `false` zurückgeben.

c) Definieren Sie eine private Methode:

```
private void bereitsAngemeldet(int id)
```

die anzeigt, daß die ID des bereits angemeldeten Studenten registriert ist.

d) Modifizieren Sie die Methode `anmelden` so, daß diese die funktion `bereitsAngemeldet` aufruft, falls sich ein bereits für die Vorlesung angemeldeter Student nochmals anmeldet.

e) Implementieren Sie zudem in der Klasse `Vorlesung` eine Methode

```
public boolean abmelden(Student s)
```

die einen Studenten `s` von der Vorlesung wieder abmeldet, d.h. der Eintrag des Studenten wird aus der Vorlesung entfernt. Die Methode `abmelden` soll den Wert `true` zurückliefern, wenn ein Student für die Vorlesung erfolgreich abgemeldet werden konnte. Ansonsten soll die Methode `false` zurückgeben.

f) Implementieren Sie eine Methode

```
public void printTeilnehmer()
```

die sowohl die Anzahl als auch die IDs aller an der Vorlesung angemeldeten Teilnehmer ausgibt.

Lösungsvorschlag 6.2

```
class Vorlesung extends MiniJava {

    private Student[] teilnehmer;

    public Vorlesung(int capacity) {
        this.teilnehmer = new Student[capacity];
    }

    public boolean anmelden(Student s) {
        for (int i = 0; i < teilnehmer.length; i++) {
            if (teilnehmer[i] != null && teilnehmer[i].getId() == s.
               getId()) {
                bereitsAngemeldet(s.getId());
                return true;
            }

        }
        for (int i = 0; i < teilnehmer.length; i++) {
            if (teilnehmer[i] == null) {
                teilnehmer[i] = s;
                return true;
            }
        }
        return false;
    }

    private void bereitsAngemeldet(int id) {
        write("Student " + id + " ist bereits zur Vorlesung angemeldet"
           );
    }

    public boolean abmelden(Student s) {
        for (int i = 0; i < teilnehmer.length; i++) {
            if (teilnehmer[i] != null && teilnehmer[i].getId() == s.
               getId()) {
                teilnehmer[i] = null;
                return true;
            }
        }
        return false;
    }
```

```
    public void printTeilnehmer() {
        int max = 0;
        for (int i = 0; i < teilnehmer.length; i++) {
            if (teilnehmer[i] != null) {
                System.out.println("Teilnehmer mit id " + teilnehmer[i
                    ].getId());
                max++;
            }
        }
        System.out.println("Vorlesung hat " + max + " Teilnehmer");
    }
}

public class TestVorlesung extends MiniJava {

    public static void main(String[] args) {
        Student katia = new Student(0);
        Student ralph = new Student(1);
        Student sara = new Student(2);
        Vorlesung mse = new Vorlesung(200);
        if (!mse.anmelden(ralph)) {
            write("Student mit ID " + ralph.getId() + " konnte nicht
                angemeldet werden!!");
        }
        if (!mse.anmelden(katia)) {
            write("Student mit ID " + katia.getId() + " konnte nicht
                angemeldet werden!!");
        }
        if (!mse.anmelden(sara)) {
            write("Student mit ID " + sara.getId() + " konnte nicht
                angemeldet werden!");
        }
        // Ralph wird hier das zweite mal angemeldet!

        if (!mse.anmelden(ralph)) {
            write("Student mit ID " + sara.getId() + " konnte nicht
                angemeldet werden!");
        }
        // Teilnehmer mit ID's werden angezeit.
        mse.printTeilnehmer();

        if (!mse.abmelden(ralph)) {
            write("Student mit ID" + ralph.getId() + " konnte nicht
                gefunden werden!");
        }
        // Ralph wird hier das zweite mal abgemeldet!

        if (!mse.abmelden(ralph)) {
            write("Student mit ID" + ralph.getId() + " konnte nicht
                gefunden werden!");
        }
        mse.printTeilnehmer();
    }
}
```

Aufgabe 6.3 (H) Quersumme

Schreiben sie ein Programm `Quersumme.java`, das die Quersumme von einer ganzen Zahl

a) *iterativ* in der Funktion `static int quersummeIt(int zahl)`
b) *rekursiv* in der Funktion `static int quersummeRec(int zahl)`

berechnet und anschließend ausgibt.

Hinweis: Der Modulo-Operator ist bei dieser Aufgabe sehr hilfreich!

Lösungsvorschlag 6.3

```
public class Quersumme extends MiniJava{
        public static int quersummeIt(int zahl){
                int qs=0;
                while(zahl>0){
                        qs=qs+zahl%10;
                        zahl=zahl/10;
                }
                return qs;
        }
        public static int quersummeRec(int zahl){
                if(zahl>0){
                        return zahl%10 + quersummeRec(zahl/10);
                }else{
                        return zahl;
                }
        }
        public static void main(String[] args) {
                write("Iterativ: "+quersummeIt(312));
                write("Rekursiv: "+quersummeRec(312));
        }
}
```