



Besprechung Uebung 1

Software Engineering



Formelles

- Abgaben:
 - Bitte pünktlich abgeben!
 - Bitte von allen Gruppenmitgliedern die Namen und die Matrikelnummern angeben. In Mail und als Deckblatt der Lösungen. Bsp:
Silvio Meier, 00-713-129
Tobias Reinhard, 01-799-312
 - Bitte nur Zip-, PDF- und Textdateien abgeben!
 - Keine Umlaute, Leerzeichen oder Sonderzeichen in den Namen der abgegebenen Dateien!
 - In Zukunft behalten wir uns das recht vor Punkte abzuziehen, falls gegen eine dieser Anforderungen verstossen wird.
- Verschiedenes:
 - Gruppen für die Übungsbearbeitung nicht Einzelpersonen.
 - Eine Abgabe pro Gruppe genügt.
 - Eher schlecht z.T. schlecht gelöst.



Aufgabe A

- Prinzipiell gut gelöst.
- Original im Quelltext:

```
public void removeFormat() {  
    int i=0;  
    do {  
        EmailAddress e=getIndex(i);  
        if(!e.check()) {  
            this.removeAddress(e);  
        }  
        i++;  
    } while (i<addresses.size());  
}
```

- Problem: Exception `ArrayIndexOutOfBoundsException`, wenn die Liste leer ist.
- Problem annehmende vs. Ablehnde Schleife



Aufgabe A (2)

- Besser ist folgende Implementierung:

```
public void removeInvalidAddresses(){  
    for (int i=0; i<addresses.size(); i++) {  
        EmailAddress e=getIndex(i);  
        if (!e.checkEmailFormat()) {  
            this.removeAddress(e);  
        }  
    }  
}
```

- Lösung **for-** bzw. **while**-Schleife



Aufgabe B

- Diskussion über
- Gelöst eigentlich gut, aber....
- Diskutieren heisst verschiedene Argumente abwägen!
- Wie vorgehen:
 - Spezielles Problem diese Schleife
 - Generalisieren
 - Pro- und Contra-Punkte abwägen
 - Eine Schlussfolgerung ziehen



Aufgabe B (2)

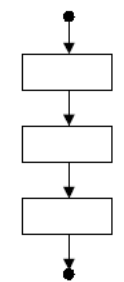
- Zwei Punkte beim Argumentieren:
 - Dynamische und statische Struktur von Programmen
 - Geschlossene Ablaufkonstrukte - wann werde diese gebrochen
- Beispiele dynamische vs. statische Programmstruktur:

```
5 print "ENTER U: "  
10 input U  
15 print "ENTER V: "  
20 input V  
30 if U<V then goto 100  
40 U=U-V  
50 if U=0 then goto 120  
60 goto 30  
100 T=U  
105 U=V  
110 V=T  
115 goto 40  
120 print "GCD: ";V
```

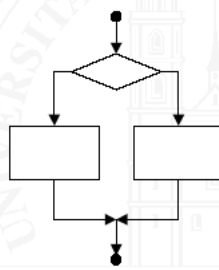
```
public int ggt(int u, int v) {  
    int t = 0; // Hilfsvariable  
    while (u > 0) {  
        if (u < v) {  
            t = u;  
            u = v;  
            v = t; //u und v getauscht  
        }  
        u = u - v;  
    }  
    return v;  
}
```

Aufgabe B (3)

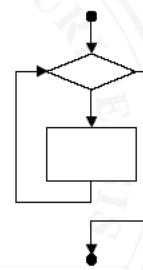
- Geschlossene Ablaufkonstrukte (genau ein Eintritts- und Austrittspunkt):



Sequenz



Alternative



Iteration

Aufgabe B (4)

- Daher kann man wie folgt argumentieren:
 - Das Break Konstrukt ist ein Sprungkonstrukt zum Ende der Schleife, somit führt break eigentlich einen weiteren Ausstiegspunkt im geschlossenen Ablaufkonstrukt ein, aber
 - Man kann auch argumentieren, dass das break lediglich eine Abkürzung zum Ausstiegspunkt ist.
 - Wesentlich ist, dass aber das break-Statement keinen Bruch des dynamischen Programmablaufs verursacht, wie einem direkt aus der Schleife springenden goto passiert.
 - Somit passt ein Break-Statement eigentlich irgendwie nicht in das Konzept geschlossener Ablaufkonstrukte. Aber weil es nicht einen Bruch zwischen dynamischen und statischen Programmablauf verursacht, bricht es das geschlossene Ablaufkonstrukt eigentlich nicht. Es kann sogar hilfreich komplizierte Schleifen zu vereinfachen.
 - Aber: Eine umsichtige Verwendung ist nötig.



Aufgabe C

- Frage war: Wann ist Rekursion besser als Iteration, nicht warum!
Konkreter Fall:

```
public String findEmail(String name) {  
    AddressList temp=new AddressList();  
    temp.setAddresses(addresses);  
    return findEmail(temp, name, name+" ");  
}  
  
public String findEmail(AddressList from,String name, String tofill) {  
    if (from.addresses.size()==0) {  
        return tofill;  
    } else {  
        String temp=from.getNameAtIndex(0);  
        if (temp.equals(name)) {  
            tofill=tofill.concat(from.getEmailAtIndex(0).concat(";"));  
        }  
        from.removeAddress(0);  
        return findEmail(from,name,tofill);  
    }  
}
```



Aufgabe C (2)

- Hier auch gefordert: Diskussion, d.h. Abwägen zwischen pro und Contra:
 - Rekursive Funktionen machen Programme schlanker...., aber nicht in jedem Fall.
 - Rekursive Funktionen sind nicht unbedingt verständlicher. Manchmal aber schon:
 - In bestimmten Fällen (z.B. Traversierung von Strukturen wie Bäumen) ist Rekursion wesentlich besser als Iteration.
 - In diesem Fall?
- Vermutlich finden die meisten, dass die Schleifenprogrammierung hier besser mittels einer Iteration erfolgt, als mit einer Rekursion.



Aufgabe C (3)

- Mögliche Iterative Lösung:

```
public String findEmail(String name) {  
    String emails=new String();  
    for (int i=0; i<addresses.size(); i++) {  
        EmailAddress e = getIndex(i);  
        if (this.getName().equals(name)) {  
            emails=emails.concat(";".concat(e.getEmail()));  
        }  
    }  
    return emails;  
}
```



Aufgabe D/E/F

- D/E eher schlecht gelöst, bzw. sehr durchzogen.
- F eigentlich gut gelöst.