Prof. Dr. V. Linnemann
Henrike Schuhart
Universität zu Lübeck
Institut für Informationssysteme

Lübeck, den 21. Januar 2005

Einführung in die Informatik I
Informatik I/A

Wintersemester 2004/05

Lösung: 12. Übungsblatt(Probeklausur)

Lösung 1: Kurzfragen zu Java und Objekt-Orientierter Programmierung

a) (3 Punkte)

```
new Double[20][20]
```

b) (3 Punkte)

- Modellierung der Instanzvariablen sind Implementierungsdetails. Falls sich deren Implementierung ändert müssen alle Programmstellen, die sich darauf beziehen verändert werden.
- Unkontrollierter Zugriff von ausserhalb der Klasse möglich.

Lösung 2: Programmverständnis

```
1  public void methode(char[] args){
2   boolean b=true;
3   Ausgabe ausgabe = Ausgabe.oeffnen("");
4   for(int i=0; i<args.length; i++){
5    if(args[i]==args[args.length-i-1])
6    {
7    }else{
8     b=false;
9     ausgabe.println("Funktioniert nicht!");
```

```
10          break;
11        }
12      }
13      if(b)ausgabe.println("Funktioniert!");
14    }
```

Diese Auflistung der Fehler muss nicht vorliegen, dient hier nur der Korrekturhilfe!

Zeile 1 Char[] -> char[]

Zeile 2 1 − > true

Zeile 3 Die Ausgabe muss erst initialisiert werden

Zeile 4 Schleifen-Variable `i` muss erst als `int` deklariert werden

Zeile 4 args.size() -> args.length

Zeile 5 = -> ==

Zeile 5 args.size() -> args.length

Zeile 10 Break -> break;

Die Methode prüft, ob die Zeichen im Zeichen-Array `args` vorwärts und rückswärts gelesen das gleiche Wort ergeben.

Lösung 3: Algorithmen - allgemein

a) (4 Punkte)

```
Stimmzettel stimmzettel = new Stimmzettel();
```

b) (16 Punkte)

```
public static void ausgeben(Stimmzettel[] zettel){
 int gueltig = 0;
 int ungueltig = 0;

 int cdu = 0;
 int spd = 0;
 int fdp = 0;
 int gruene = 0;

 //auszaehlen
 for(int i=0; i<zettel.length;i++){
  if(zettel[i].cdu && !zettel[i].spd && !zettel[i].fdp && !zettel[i].gruene){
   cdu++;
  }else
  if(!zettel[i].cdu && zettel[i].spd && !zettel[i].fdp && !zettel[i].gruene){
   spd++;
  }else
```

```
    if(!zettel[i].cdu && !zettel[i].spd && zettel[i].fdp && !zettel[i].gruene){
     fdp++;
    }else
    if(!zettel[i].cdu && !zettel[i].spd && !zettel[i].fdp && zettel[i].gruene){
     gruene++;
    }else{
     ungueltig++;
    }
   }

   //Gueltige Stimmzettel berechnen
   gueltig = zettel.length - ungueltig;

   //Anteile errechnen
   double anteilCDU = 100*((double)cdu) / gueltig;
   double anteilSPD = 100*((double)spd) / gueltig;
   double anteilGruene = 100*((double)gruene) / gueltig;
   double anteilFDP = 100*((double)fdp) / gueltig;

   // Ausgabe
   System.out.println("Die CDU hat "+anteilCDU+
                                          " % der Stimmen.");
   System.out.println("Die SPD hat "+anteilSPD+
                                          " % der Stimmen.");
   System.out.println("Die Gruenen haben "+anteilGruene+
                                          " % der Stimmen.");
   System.out.println("Die FDP hat "+anteilFDP+
                                          " % der Stimmen.");
   System.out.println("Es wurden "+ungueltig+
                        " ungueltige Stimmen abgegeben.");

  }
```

Lösung 4: Algorithmen - rekursiv

a) (6 Punkte)

```
/**
 * a>=0 und b>0
 */
public int div(int a, int b){
 if(a<b)return 0;
 return 1+div(a-b,b);

}
```

b) (2 Punkte) Aufrufbaum für div(9,3):

```
div(9,3)
   |
div(6,3)
   |
div(3,3)
   |
div(0,3)
```

c) (8 Punkte)

```
/**
 * a>=0
 */
public boolean istPrim(int a){
 if(a<=1)return false;
 if(a==2)return true;
 return istPrim(a,2);
}

public boolean istPrim(int a,int b){
 if(a==b)return true;
 if(a%b==0)return false;
 return istPrim(a,b+1);
}
```

d) (4 Punkte)

```
istPrim(5)
    |
istPrim(5,2)
    |
istPrim(5,3)
    |
istPrim(5,4)
    |
istPrim(5,5)
```

Lösung 5: Vererbung

a) (12 Punkte)

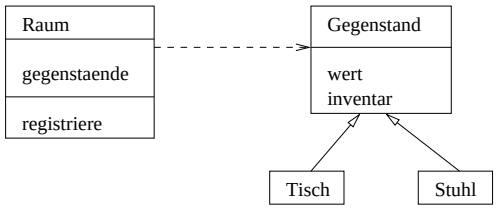

Abbildung 1: Vereinfachtes UML-Klassendiagramm zu a)

Listing 1: Die drei Klassen `Gegenstand,Tisch,Stuhl`

```
public class Gegenstand{
 private String inventar;
 private int wert;
```

```
 public Gegenstand(String inventar,int wert){
  this.inventar = inventar;
  this.wert = wert;
 }
}

public class Tisch extends Gegenstand{
 public Tisch(String inventar,int wert){
  super(inventar,wert);
 }
}

public class Stuhl extends Gegenstand{
 public Stuhl(String inventar,int wert){
  super(inventar,wert);
 }
}
```

Listing 2: Die Methode `registriere` der Klasse `Raum`

```
public void registriere(Gegenstand gegenstand){
 this.gegenstaende.add(gegenstand);
}
```

b) (5 Punkte)

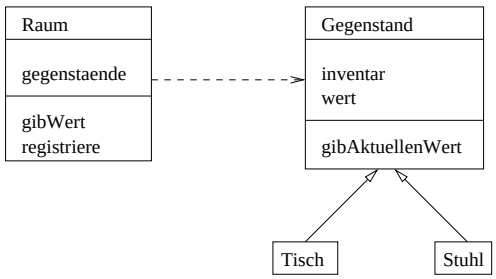

Abbildung 2: Vereinfachtes UML-Klassendiagramm zu b)

Listing 3: Die Methode `gibWert` der Klasse `Raum`

```
public int gibWert(){
 int wert = 0;
 for(int i=0; i<this.gegenstaende.size();i++){
  wert+=((Gegenstand)this.gegenstaende.get(i)).gibAktuellenWert();
 }
 return wert;
}
```

Listing 4: Die Methode `gibAktuellenWert` der Klasse `Gegenstand`

```
public int gibAktuellenWert(){
 return this.wert;
}
```

c) (9 Punkte)

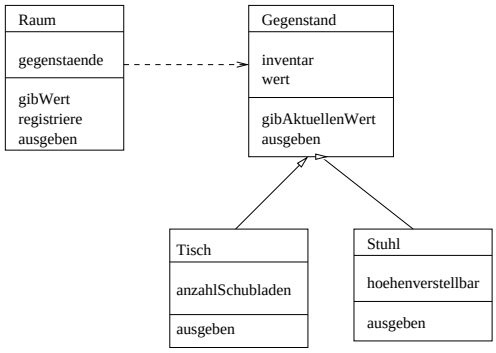

Abbildung 3: Vereinfachtes UML-Klassendiagramm zu c)

Listing 5: Die Methode `ausgeben` der Klasse `Raum`

```
public void ausgeben(){
  for(int i=0; i<this.gegenstaende.size();i++){
      ((Gegenstand)this.gegenstaende.get(i)).ausgeben();
      }
}
```

Listing 6: Die drei Klassen `Gegenstand`, `Tisch`, `Stuhl` mit jeweils eigener Version der Methode `ausgeben`

```
public class Gegenstand{
 private String inventar;
 private int wert;
 ...
 public void ausgeben(){
  System.out.println("Inventar: "+inventar);
  System.out.println("Wert: "+wert);
 }
}

public class Tisch extends Gegenstand{
 private int anzahlSchubladen;

 public void ausgeben(){
  super.ausgeben();
  System.out.println("Anzahl Schubladen: "+anzahlSchubladen);
 }
}

public class Stuhl extends Gegenstand{
 private boolean hoehenverstellbar;

 public void ausgeben(){
  super.ausgeben();
  System.out.println("Ist hoehenverstellbar: "+hoehenverstellbar);
 }
}
```