Saarland
Ministerium
für Bildung und Kultur

Lösungshinweise: Nur für die Hand der Lehrperson

Schriftliche Abiturprüfung 2014

Fach:	**Informatiksysteme (Fachrichtung Technik)**
Kurstyp:	E-Kurs
Bearbeitungszeit:	5 Zeitstunden
Hilfsmittel:	Java-IDE Netbeans, Umlet (Tool zum Zeichnen des Klassendiagramms), XAMPP (MySQL-DBMS, SQL-Client phpmyadmin)
Seitenzahl:	Die Lösungshinweise umfassen mit Deckblatt, Bewertungsrichtlinien, Lösungsvorschlägen und Bewertungstabelle 20 Seiten.

Bewertungsrichtlinien

Aufgabe	Beschreibung	Punkte
1	**Fragen zur Objektorientierten Programmentwicklung**	
	a)Multiplizitäten	
	• Definition	2
	• Abbildung 1 erläutern	2
	• 1:1 und n:m	2
	• Assoziationsklasse	2
	b)Schnittstellen(Interfaces)	
	• Definition und Bedeutung	3
	• Iterator	6
	c)Sortieren	
	• InsertionSort Pseudocode	4
	• Beispiel	3
	Σ	**24**
2	**Objektorientierte Programmentwicklung**	
	• Klassendiagramm	5
	• Klasse CGast	
	• ToString()	1
	• Attribute	1
	• Methoden	1
	Σ	**8**
	• Klasse CZimmer	
	• Art der Zimmer	1
	• Konstruktor	1
	• Attribute	1
	• Methoden	1
	• toString	1
	Σ	**5**
	• Klasse CBuchung	
	• Datum konvertieren	3
	• Konstruktor	1

	• Attribute	1
	• Methoden	1
	• Objektreferenzen	1
	• toString	1
	Σ	**8**
3	**Objektorientierte Programmierung**	
	• Klasse CVerwaltung	
	• Container	2
	• Singleton	2
	• NeuerGast()	2
	• neues Zimmer()	2
	• alleGaeste()	4
	• allefreienZimmer()	4
	• buchen()	4
	• ausbuchen()	4
	Σ	**24**
	• Klasse MyHotelUI	
	• Zimmer anlegen	2
	• MangerObjekt	1
	• Gäste anlegen	1
	• Ausgabe	1
	• Buchen	1
	• ausbuchen	1
	Σ	**7**
4	**Datenbank: ER-Diagramm und Fragen**	**Punkte**
	• Konzertagentur	15
	• Schlüssel	2
	• Löschanomalie	2
	• N:M Beziehung	2
	• Normalisierung	3

	Σ	**24**
	Gesamtpunktzahl	**100**

Lösungvorschläge:

Aufgabe 1

Multiplizitäten

i Ein Objekt kann zu genau einem, keinem oder zu vielen anderen Objekten eine Beziehung haben. Dieser Sachverhalt wird durch die Multiplizität einer Assoziation beschrieben. Die im UML-Klassediagramm dargestellten Assoziationen werden durch eine Linie beschieben an deren Ende diese Zahl (Wertigkeit) notiert wird. Sie sagt aus wie viel andere Objekte ein bestimmtes Objekt kennen kann.

ii Am Beispiel des Klassendiagramms in Abbildung 1 besteht zwischen CGast und Buchung eine 1:N Assoziation. Zu jedem Buchungsobjekt gehört ein Gastobjekt. Ein Gastobjekt kann aber mehrere Buchungsobjekte "kennen".

iii Man unterscheidet noch:

iv Die 1:1 Assoziation, bei der ein Objekt der Klasse A genau ein Objekt der Klasse B kennt und umgekehrt ein Objekt der Klasse B genau ein Objekt der Klasse A kennt. Beispiel ist ein Auto und sein Parkplatz.

v Die M:N Assoziation, bei der jedes Objekt der Klasse A viele der Klasse B kennt und umgekehrt. Beispiel : Ein Flug hat viele Passagiere, wobei ein Passagier mehrere Flüge absolvieren kann.

vi Eine Assoziationsklasse besitzt sowohl die Eigenschaft einer Assoziation, als auch die einer Klasse. Wie in dem Klassendiagramm in Abbildung 1 gezeigt, ist es sinnvoll die Information zur Buchung eines Zimmers im Hotel wie Datum und Dauer an die Assoziation zu hängen. Diese Attribute müssen aber in einer Klasse gepaselt werden, der sogenannten Assoziationsklasse. In der OOP wird die Assoziationsklasse wie gezeigt aufgelöst in zwei 1:n Assoziationen.

b Schnittstellen (Interfaces)

i Schnittstellen stellen in der OOP Methodensignaturen zur Verfügung, die nicht implementiert werden. Sie legen Funktionalitäten fest, aber nicht wie sie realisiert werden, also das "was" nicht das "wie". Sie werden häufig im Kontext der Vererbung verwendet und unterstützen den Polymorphismus.

ii Anwendung:

```
package schnittstelle;

import java.util.ArrayList;
import java.util.Iterator;


public class MeinContainer {
    private ArrayList<String> liste =new ArrayList <String>();
    MeinContainer(){
        liste.add("Meier");
        liste.add("Mueller");
        liste.add("Schulz");
        liste.add("Schneider");

    }

    public void Ausgabe(){
      Iterator it=liste.iterator();
      while(it.hasNext()){
       String aus = (String)it.next();
       System.out.println(aus);
      }
    }

}
```

```
package schnittstelle;

public class Schnittstelle {
    public static void main(String[] args) {
        // TODO code application logic here
        MeinContainer tonne=new MeinContainer();
        tonne.Ausgabe();
    }
}
```

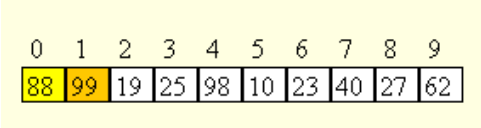
a Sortieren:

1. Gegeben ist eine Menge von n- Elementen;
2. Wähle ein Element aus und setze es auf Position eins;
3. Nehme das nächste Element;
4. Vergleiche das Element mit den sortierten Elementen, bis es kleiner als eines der einsortierten Elemente ist
5. Ordne es davor ein;
6. gehe wieder zu 3 und verfahre solange bis alle Elemente einsortiert sind

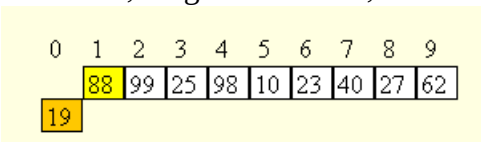
alt.:

Algorithmus:
für i=1 bis n-1:
wenn A[i].Schluessel < A[i-1].Schluessel dann:
(sonst ist A[i] bereits an der richtigen Stelle)
speichere A[i] in einer Hilfsvariablen v
j = i
(suche nach dem richtigen Platz)
solange j>0 und A[j-1].Schluessel > v.Schluessel:
verschiebe A[j-1] um einen Platz nach rechts
j = j-1
(jetzt ist erstmalig A[j-1].Schlüssel <= v.Schlüssel)
speichere die Hilfsvariable v in A[j] (freier Platz)
(A[0] ... A[i] ist sortiert)

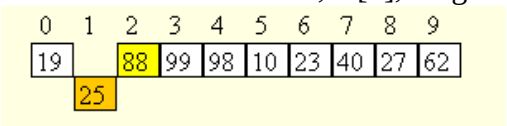
ai Beispiel:



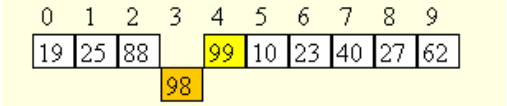
nehme 99, vergleiche mit 88, einordnen



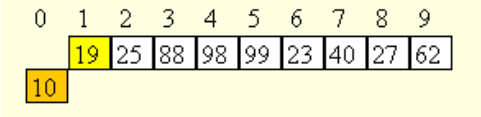
nehme nächstes Element, A[2], vergleiche mit den Elementen davor und ordne ein



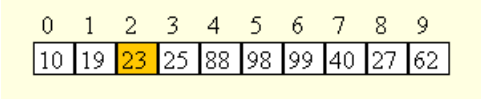
nehme nächstes Element, A[3], vergleiche mit den Elementen davor und ordne ein



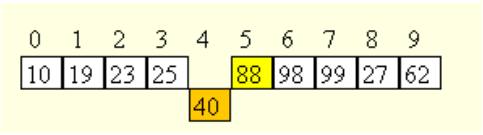
nehme nächstes Element, A[4], vergleiche mit den Elementen davor und ordne ein



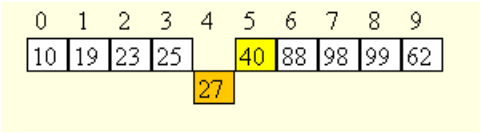
nehme nächstes Element, A[5], vergleiche mit den Elementen davor und ordne ein



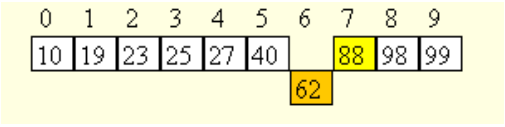
nehme nächstes Element, A[6], vergleiche mit den Elementen davor und ordne ein



nehme nächstes Element, A[7], vergleiche mit den Elementen davor und ordne ein



nehme nächstes Element, A[8], vergleiche mit den Elementen davor und ordne ein

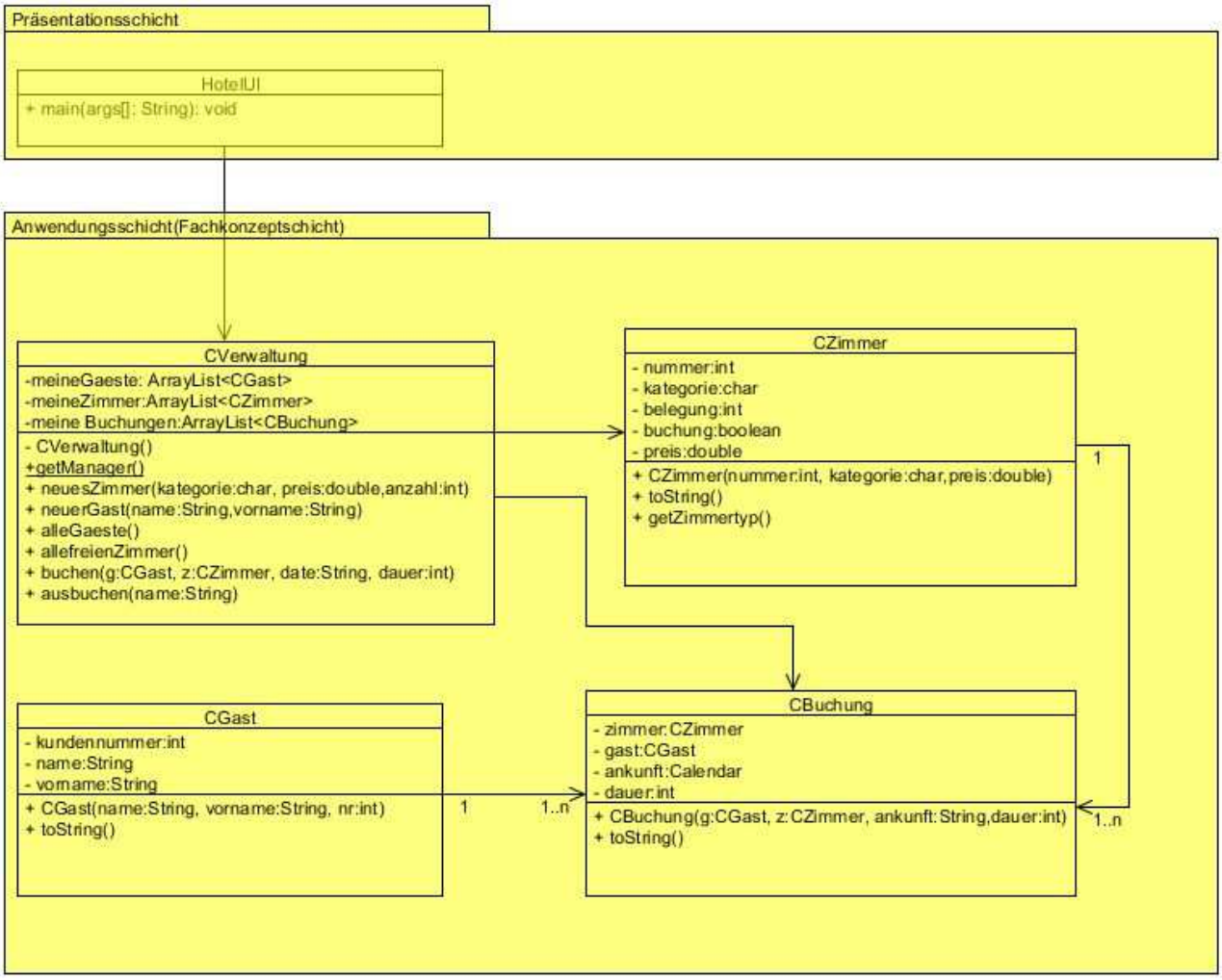


nehme nächstes Element, A[9], vergleiche mit den Elementen davor und ordne ein

Aufgabe2

Objektorientierte Programmentwicklung

a Klassendiagramm



b Anwendung:

Klasse CZimmer

```
package myhotel;


public class CZimmer {
    private  int nummer;
    private char kategorie;//A= Einzel, B=Doppel, C=Suite
    private double preis;
    private int belegung;//Anzahl der Gäste
    private boolean buchung;//true = gebucht
private static int anzahl;
    public CZimmer(){

    }
    public CZimmer(int nummer, char kategorie, double preis) {
        this.nummer = nummer;
        this.kategorie = kategorie;
        this.preis=preis;
        belegung = 0;
        buchung=false;
    }

    public static int getAnzahl() {
        return anzahl;
    }

    public static void setAnzahl(int anzahl) {
        CZimmer.anzahl = anzahl;
    }

    public int getBelegung() {
        return belegung;
    }

    public void setBelegung(int belegung) {
        this.belegung = belegung;
    }

    public boolean getBuchung() {
        return buchung;
    }

    public void setBuchung(boolean buchung) {
        this.buchung = buchung;
    }
```

```
    public char getKategorie() {
        return kategorie;
    }

    public void setKategorie(char kategorie) {
        this.kategorie = kategorie;
    }

    public int getNummer() {
        return nummer;
    }

    public void setNummer(int nummer) {
        this.nummer = nummer;
    }

    public double getPreis() {
        return preis;
    }

    public void setPreis(double preis) {
        this.preis = preis;
    }

    public String getZimmertyp(){
        String typ="";
        switch(kategorie)
        {
            case 'A':
                typ="Einzelzimmer";
                break;
            case 'B':
                typ="Doppelzimmer";
                break;
            case 'C':
                typ="Suite";
          }
        return typ;
    }
    @Override
    public String toString(){
        return (String.format("%nZimmernummer: %d%nKategorie: %s"
                , nummer, getZimmertyp()));
    }

}
```

Klasse CGast

```
package myhotel;

public class CGast {
    private String name;
    private String vorname;
    private int kundennummer;

    public CGast(String name, String vorname, int kundennummer) {
        this.name = name;
        this.vorname = vorname;
        this.kundennummer=kundennummer;
    }

    public int getKundennummer() {
        return kundennummer;
    }

    public void setKundennummer(int kundennummer) {
        this.kundennummer = kundennummer;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getVorname() {
        return vorname;
    }

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

   public String toString(){

        return (String.format("%nKundennummer: %d%nName: %s%n"
                + "Vorname: %s", kundennummer, name, vorname));
    }

}
```

Klasse CBuchung

```
package myhotel;

import java.text.DateFormat;
import java.text.ParseException;
import java.util.Calendar;
import java.util.Date;
import java.util.TimeZone;
import java.util.logging.Level;
import java.util.logging.Logger;

public class CBuchung {

    private Calendar ankunft;
    private int dauer;//in Tagen
    private CGast gast;
    private CZimmer zimmer;


    public CBuchung(String an, int dauer, CGast gast, CZimmer zimmer) {
        Calendar d=Calendar.getInstance();
        DateFormat myFormat=DateFormat.getDateInstance(DateFormat.SHORT);
        myFormat.setTimeZone(TimeZone.getDefault());
        Date date = null;
        try {
            date=myFormat.parse(an);
        } catch (ParseException ex) {
            Logger.getLogger(CBuchung.class.getName()).log(Level.SEVERE, null, ex);
        }
        d.setTime(date);
        this.ankunft=d;

        this.dauer = dauer;
        this.gast = gast;
        this.zimmer = zimmer;
        zimmer.setBuchung(true);
    }
```

```
    public String getAnkunft() {
        return ankunft.toString();
    }

    public void setAnkunft(Calendar ankunft) {
        this.ankunft = ankunft;
    }
     public void setAnkunft(String datum) {//tt.mm.jjjj
        Calendar d=Calendar.getInstance();
        DateFormat myFormat=DateFormat.getDateInstance(DateFormat.FULL);
        myFormat.setTimeZone(TimeZone.getDefault());
        Date date = null;
        try {
            date=myFormat.parse(datum);
        } catch (ParseException ex) {
            Logger.getLogger(CBuchung.class.getName()).log(Level.SEVERE, null, ex);
        }
        d.setTime(date);
        this.ankunft=d;
    }

    public int getDauer() {
        return dauer;
    }

    public void setDauer(int dauer) {
        this.dauer = dauer;
    }

    public String getAbreise(){
        long time;
        Calendar abreise=Calendar.getInstance();
        time=ankunft.getTimeInMillis()+24*3600*1000*dauer;
        abreise.setTimeInMillis(time);
        return abreise.toString();
    }

    public CGast getGast() {
        return gast;
    }

    public CZimmer getZimmer() {
        return zimmer;
    }

    public String toString(){
        return (String.format("%nZimmernummer: %d%n"
                + "Von: %s%nBis: %s",zimmer.getNummer(),ankunft,getAbreise() )).
    }

}
```

Aufgabe 3

Klasse CVerwaltung

```
package myhotel;

import java.util.ArrayList;

public class CVerwaltung {
    private ArrayList <CZimmer>meineZimmer=new ArrayList<CZimmer>();
    private ArrayList <CGast>meineGaeste=new ArrayList<CGast>();
    private ArrayList <CBuchung>meineBuchungen=new ArrayList<CBuchung>();

    private static CVerwaltung manager=null;
    //Wegen Singlton-Muster Konstruktor private
    private CVerwaltung() {
        CZimmer.setAnzahl(0);
    }
    public static CVerwaltung getManager() {

        if (manager == null) {
            manager = new CVerwaltung();
        }else{
        System.out.println("%nVerwaltungs-Objekt wurde bereits erzeugt!");

        }
        return manager;
    }
    ////////////////////////Methoden////////////////////////////
```

```
    public ArrayList<CBuchung> getMeineBuchungen() {
        return meineBuchungen;
    }

    public void setMeineBuchungen(ArrayList<CBuchung> meineBuchungen) {
        this.meineBuchungen = meineBuchungen;
    }

    public ArrayList<CGast> getMeineGaeste() {
        return meineGaeste;
    }

    public void setMeineGaeste(ArrayList<CGast> meineGaeste) {
        this.meineGaeste = meineGaeste;
    }

    public ArrayList<CZimmer> getMeineZimmer() {
        return meineZimmer;
    }

    public void setMeineZimmer(ArrayList<CZimmer> meineZimmer) {
        this.meineZimmer = meineZimmer;
    }

    public void neuerGast(String name, String vorname){
        int kdnr=meineGaeste.size()+1300;
        meineGaeste.add(new CGast(name,vorname,kdnr));
    }
```

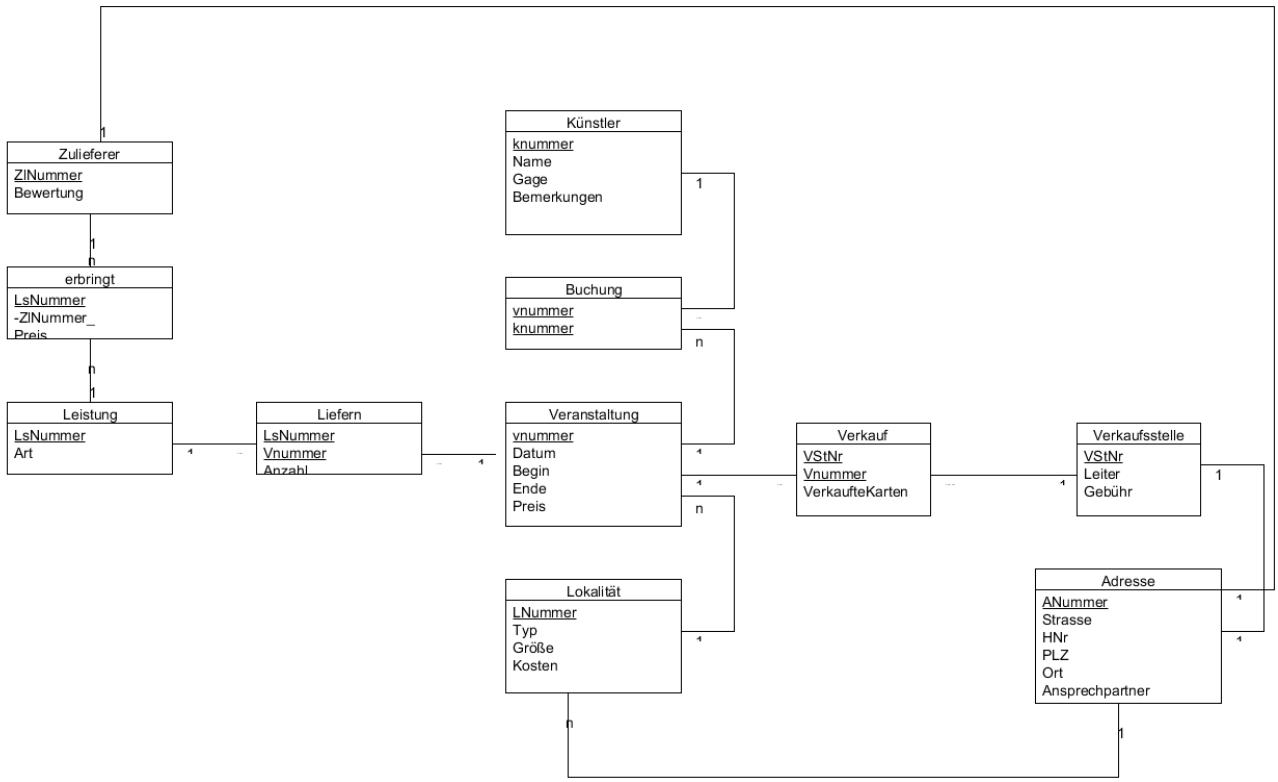
```
    public void neueZimmer( char kategorie, double preis, int anzahl){
        int count;
        count=CZimmer.getAnzahl();
        for(int i=1;i<=anzahl;i++){
            CZimmer z=new CZimmer(i+count, kategorie,preis);
            meineZimmer.add(z);
        }
        count=count+anzahl;
        CZimmer.setAnzahl(count);
    }
    public void alleGaeste(){
        System.out.println();
        System.out.println("------------------------");
        System.out.printf("%nFolgende Gäste sind gespeichert:");
        for(CGast g: meineGaeste){
            System.out.printf("%n%s", g.toString());
        }
    }
    public void gebuchteZimmer(){
     System.out.println();
     System.out.println("------------------------");
        System.out.printf("%nFolgende Zimmer sind belegt:");
      for(CZimmer z: meineZimmer){
          if(z.getBuchung()==true)
            System.out.printf("%n%s", z.toString());
    }
    }
    public void allefreienZimmer(){
        System.out.println();
        System.out.println("------------------------");
        System.out.printf("Folgende Zimmer sind noch frei:");
      for(CZimmer z: meineZimmer){
          if(z.getBuchung()==false)
            System.out.printf("%n%s", z.toString());
        }
    }
     public void buchen(CGast g, CZimmer z, String date, int dauer) {
        meineBuchungen.add(new CBuchung(date,dauer,g,z));
    }

    public void ausbuchen(String name){
        double rechnung=0.0;
        int nr=0;
     for(CBuchung b:meineBuchungen)  {
          if(b.getGast().getName().equals(name)){
              b.getZimmer().setBuchung(false);
              rechnung= b.getZimmer().getPreis()*b.getDauer();
              nr=b.getZimmer().getNummer();
              break;
          }
     }
     System.out.printf("%n%nDie Rechnung für Zimmer %d beträgt %.2f €",nr,rechnung );
     }
}
```

Die Klasse MyHotelUI

```
package myhotel;


public class MyHotel {

    
    public static void main(String[] args) {
        // TODO code application logic here
        CVerwaltung hotel=CVerwaltung.getManager();
        //Zimmer

        hotel.neueZimmer('B', 75.0, 3);
        hotel.neueZimmer('C', 125.0, 2);
        hotel.allefreienZimmer();
        hotel.neuerGast("Muster","Max");
        hotel.neuerGast("Musterfrau", "Ilse");
        hotel.alleGaeste();
        hotel.buchen(hotel.getMeineGaeste().get(0),hotel.getMeineZimmer().get(0),"11.08.2013",5);
        hotel.buchen(hotel.getMeineGaeste().get(0),hotel.getMeineZimmer().get(3),"12.08.2013",5);
        hotel.buchen(hotel.getMeineGaeste().get(1),hotel.getMeineZimmer().get(4),"13.08.2013",14);
        hotel.allefreienZimmer();
        //hotel.gebuchteZimmer();
        hotel.ausbuchen("Musterfrau");

    }
}
```

Aufgabe 4

a-c)



Schlüssel unterstrichen

a. *Der Schlüssel muss eindeutig sein und den ganzen Datensatz eindeutig beschreiben. Gelingt dies mit einem Schlüsselattribut nicht, werden Kombinationsschlüssel aus mehreren Attributen gebildet.*

b. *N:M Beziehungen werden in zwei N:1/M:1 Beziehungen aufgelöst indem eine Koppelentität eingefügt wird. Der Primärschlüssel dieser Relation setzt sich aus den Schlüsseln der gekoppelten Relationen zusammen.*

c. *Eine Löschanomalie entsteht, wenn beim Löschen eines Datensatzes mehr Information als gewünscht verloren gehen oder der Zugang zu bestimmten Daten nicht mehr möglich ist.*

d. *Normalisierung*

   Erste Normalform: Daten müssen Atomar vorliegen und durch einen Schlüssel eindeutig identifiziert werden können
   Zweite Normalform: Die Relation befindet sich in der ersten NF und jedes Nichtschlüsselattribut ist vom Primärschlüssel voll funktional abhängig.

Dritte Normalform: Die Relation befindet sich in der zweiten Normalform und es gibt keine transitiven Abhängigkeiten d.h. keine Abhängigkeiten zwischen Nichtschlüsselattributen

Bewertungstabelle

Prozent der maximal erreichbaren Roh-punktzahl	Note	Punktzahl
ab 97% bis 100% der max. Punktzahl	sehr gut	15 P
ab 93% bis weniger als 97%		14 P
ab 90% bis weniger als 93%		13 P
ab 85% bis weniger als 90%	gut	12 P
ab 80% bis weniger als 85%		11 P
ab 75% bis weniger als 80%		10 P
ab 70% bis weniger als 75%	befriedigend	09 P
ab 65% bis weniger als 70%		08 P
ab 60% bis weniger als 65%		07 P
ab 55% bis weniger als 60%	ausreichend	06 P
ab 50% bis weniger als 55%		05 P
ab 45% bis weniger als 50%		04 P
ab 38% bis weniger als 45%	mangelhaft	03 P
ab 32% bis weniger als 38%		02 P
ab 25% bis weniger als 32%		01 P
weniger als 25% der max. Punktzahl	ungenügend	00 P

- Ende der Lösungshinweise -