Ministerium für Bildung und Kultur
SAARLAND

Lösungshinweise: Nur für die Hand der Lehrperson

Schriftliche Abiturprüfung 2015

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)Assoziation	
	• Definition	**1**
	• Rollen- und Assoziationsname	**1**
	• Kardinalität	**1**
	Teil-Ganzes-Assoziation	
	• Beschreibung und Bezeichnung	**2**
	• UML-Beispiel	**2**
	b)generische Datentypen	**2**
	c)Sortieren	**7**
	d)funktionale Bindung	**2**
	Σ	**18**
2	**Objektorientierte Programmentwicklung**	
	• Klasse CVerkauf	
	• ToString()	**1**
	• Attribute und Methoden	**2**
	• Referenzierung	**3**
	• **Σ**	**6**
	• Klasse CMitarbeiter	
	• ToString()	**1**
	• Attribute und Methoden	**2**
	• Referenzierung	**1**
	Σ	**4**
3	**Objektorientierte Programmentwicklung**	
	▪ Klasse CVerwaltung	
	• container	**1**
	• Singleton	**4**
	• neuerMitarbeiter	**2**
	• addNeuerMitarbeiter	**2**
	• alleMitarbeiter	**4**

	• sucheMitarbeiter	**6**
	• showMitarbeiter	**3**
	• umsatzAlle	**6**
	• Select	**3**
	Σ	**31**
4	**Objektorientierte Programmierung (Persistenz)**	
	• Klasse CConnectionManager	
	Σ	**7**
	• Klasse CKFZVerkaufSQL	**2**
	• Selektiere Mitarbeiter	**10**
	• Selektiere Verkäufe SQL-Befehl	**5**
	Σ	**17**
5	**Testfälle**	
	• Instance	**1**
	• select	**1**
	• Ausgabe Mitarbeiter	**1**
	• Umsatz	**1**
	• Suchen	**1**
	Σ	**5**
6	**Datenbank: ER-Diagramm und Fragen**	**Punkte**
	• Koppel Verkauf	3
	• Koppel AnforderungDienst	3
	• KFZ	1
	• Kunde	1
	• Adresse	1
	• Firma	1
	• Dienstleister	1
	• Ort	1
	Σ	**12**
	Gesamtpunktzahl	**100**

Lösungvorschläge:

Aufgabe 1:

a Assoziationen

i. Was versteht man unter einer Assoziation zwischen Klassen in der Objektorientierten Programmierung und wie wird sie in UML dargestellt Erläutern Sie die Begriffe oder Bedeutung von Assoziationsname, Rollenname und der Kardinalität.

Lösung:

In der OOP wird die Abhängigkeit zweier Klassen voneinander als Assoziation bezeichnet und in UML als Verbindungslinie zwischen den beteiligten Klassen dargestellt. Die Beziehung der Klassen wird in der Regel durch einen *Assoziationsnamen* näher beschrieben. Der Rollenname spezifiziert die Beziehung etwas genauer. In einer Assoziation zwischen Konto und Kunde bei einer Bank kann der Zugriff auf das Konto durch den Kunden einmal in der Rolle des Kontoinhabers oder in der Rolle eines Kontoberechtigten erfolgen. Die Kardinalität einer Assoziation beschreibt die Anzahl der Objekte einer Klasse A, die zu einem Objekt der Klasse B inBeziehung stehen.

ii. Neben der Assoziation gleichberechtigter Klassen gibt es noch Teil-Ganzes – Beziehungen. Wie nennt man die beiden Sonderformen, wie werden Sie in UML dargestellt und welche Besonderheiten weisen sie auf? Geben Sie jeweils ein Beispiel an und erläutern Sie dies.

Lösung:

Als Teil-Ganzes Beziehungen gibt es die Aggregation und die Komposition.
Aggregation: Die Beziehung zwischen einem Berater und dem Projekt an dem er beteiligt ist ist ein Beispiel für eine Aggregation. Der Berater ist Teil des Projekts, kann aber andere Projekte betreuen. Er wird bei Beendigung (Löschen) des Projekts nicht gelöscht.

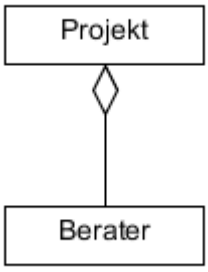

Abbildung 1: Aggregation

Komposition:

Beispiel für eine Komposition ist eine Rechnungsposition in einer Rechnung. Sie ist Teil der Rechnung, wird aber mit der Rechnung gelöscht. Rechnung ist das Aggragtionsobjekt, nachdem sich die Rechnungsposition als Teil bzgl. der Lebensdauer richtet.

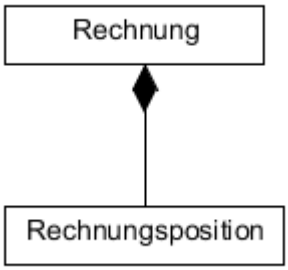

Abbildung 2: Komposition

b Datentypen

Was versteht man unter Generischen Datentypen? Welchen Vorteil bieten Sie und wozu werden sie verwendet?

Lösung:

Viele Programme, d.h. Klassen und Methoden sind weitgehend unabhängig von Datentypen. Algorithmen zum Suchen oder Sortieren funktionieren beispielsweise unabhängig davon ob Integer- Float-, String- oder sonstige Datentypen gesucht oder sortiert werden sollen. Durch generische Typen ist es möglich Klassen oder Methoden zu verallgemeinern., also typunabhängig zu implementieren. Als Typstellvertreter eingesetzt, der erst bei der Anwendung durch einen konkreten Typ ersetzt wird.

c Algorithmen

Implementieren Sie am Beispiel eines einfachen Ganzzahlfolge das Sortierverfahren Selektionsort (Sortieren durch Auswahl). Führen Sie das Verfahren am Beispiel der Zahlenfolge in Abbildung 2 durch. Geben Sie die Folge nach jedem Tausch aus.

0	1	2	3	4	5	6	7	8	9
49	40	39	6	83	9	47	87	21	66

Abbildung 3

Lösung:

```
package selectionsort;

public class SelectionSort {
public static int[] menge={49,40,39,6,83,9,47,87,21,66};

public static void ausgabe(int[] a){
    for(int i=0;i< a.length;i++){
     System.out.printf("%d ",a[i]);
    }
    System.out.println();
}
public static void sort(int[] a){
    int hilf, min, pos, i, j,n;//Deklarationen
    n=a.length;
    for(i=0;i<n-1;i++){
        min=a[i];//der i-te Wert wird als kleinster Wert gesetzt
        pos=i;//
        for(j=i+1;j<n;j++){//alle darauffolgenden Werte werden verglichen
            if(a[j]<min){
                min=a[j];
                pos=j;
            }
        }
            //Dreieckstausche Platz i mit Platz j
```

```
                hilf=a[i];
                a[i]=min;
                a[pos]=hilf;
                ausgabe(a);
            }

        }

        public static void main(String[] args) {
            // TODO code application logic here
            ausgabe(menge);
            sort(menge);
        }

    }
```

d Qualitätssicherung

Was versteht man unter funktionaler Bindung von Methoden und warum ist diese zur Qualitätssicherung anzustreben?

L**ösung:**

Eine gute Bindung liegt vor, wenn nur solche Elemente zu einer Methode zusammengefasst werden, die auch dazugehören. Ziel ist es eine gute funktionale Bindung zu erreichen. Diese liegt vor, wenn alle Elemente an der Verwirklichung einer einzigen abgeschlossenen Funktion beteiligt sind.

Aufgabe 2: Objektorientierte Programmentwicklung

Klasse CMitarbeiter

```
import java.util.ArrayList;

public class CMitarbeiter {
    private int mid;
    private String name;
    private ArrayList<CVerkauf>meineVerkaeufe;

    public CMitarbeiter(){
    }
    public CMitarbeiter(int mid, String name){
        this.mid=mid;
        this.name=name;
        this.meineVerkaeufe=new ArrayList<CVerkauf>();
    }
    @Override
    public String toString(){
        return String.format("\nMid: %d\tName: %s",mid, name);
    }
    public ArrayList<CVerkauf> getMeineVerkaeufe() {
        return meineVerkaeufe;
    }

    public int getMid() {
        return mid;
    }
}
```

Klasse CVerkauf

```
package kfzverkauf;
import java.util.ArrayList;
public class CVerkauf {
    private int reNr;
    private String kfz;
    private double preis;
    public CVerkauf(){
    }
    public CVerkauf(int reNr, String kfz, double preis, CMitarbeiter mit) {
        this.reNr = reNr;
        this.kfz = kfz;
        this.preis = preis;
        //Referenz zu Mitarbeiter
        ArrayList <CVerkauf>hilf;
        hilf=mit.getMeineVerkaeufe();
        hilf.add(this);
    }
    public String toString(){
        String aus= String.format("\nReNr.: %d\t%s,"
                + "\tPreis: %.2f €",reNr, kfz,preis );
        return aus;
    }
    public int getReNr() {
        return reNr;
    }
    public double getPreis() {
        return preis;
    }
}
```

Aufgabe 3: Objektorientierte Programmentwicklung (Containerklasse)

Lösung:

Klasse CVerwaltung

```
package kfzverkauf;
import java.io.IOException;
import java.sql.SQLException;
import java.util.ArrayList;

public class CVerwaltung {
    private static CVerwaltung instance=null;
    //Container
    private ArrayList<CMitarbeiter> meineMitarbeiter;
    //Singleton
    private CVerwaltung(){
     meineMitarbeiter=new ArrayList<CMitarbeiter>();
    }
```

```
    public static CVerwaltung getInstance(){
     if(instance==null){
         instance= new CVerwaltung();
     }else
     {
         System.out.println("Objekt existiert schon!!!!!");
     }
     return instance;
    }
   //Methoden
    public CMitarbeiter neuerMitarbeiter(int mit, String name){
        return new CMitarbeiter(mit, name);
    }
    public void addNeuerMitarbeiter(CMitarbeiter m){
        meineMitarbeiter.add(m);
    }

    public void alleMitarbeiter(){
        for(CMitarbeiter m:meineMitarbeiter){
            System.out.println(m.toString());

        }
    }

    public CMitarbeiter sucheMitarbeiter(int nr){
        boolean treffer=false;
        CMitarbeiter t=null;
        ArrayList<CVerkauf>hilf;
        for(CMitarbeiter m: meineMitarbeiter){
            hilf=m.getMeineVerkaeufe();
            for(CVerkauf v:hilf){
                if(v.getReNr()==nr){
                    t=m;
                }
            }
        }
        return t;
    }

     public void alleVerkaeufe()
     {
         for(CMitarbeiter m:meineMitarbeiter){
             ArrayList<CVerkauf>hilf=m.getMeineVerkaeufe();
             for(CVerkauf v:hilf){
                 System.out.println(v.toString());
             }
         }
     }
```

```
public void alleVerkaufeMitarbeiter(){
    int count;
    System.out.println("Mitarbeiter mit Verkäufen");
    System.out.println("==========================");
    for(CMitarbeiter m:meineMitarbeiter){
        System.out.println(m.toString());
        System.out.println("----------------------------------------");
        System.out.println("Verkäufe:");
        count=0;
        for(CVerkauf v: m.getMeineVerkaeufe()){
            System.out.print(v.toString());
            count++;
        }
        if(count==0){
            System.out.print("keine Verkäufe");
        }
        System.out.printf("\n_____");
    }
}
```

```
public void showMitarbeiter(CMitarbeiter m){
    System.out.printf("\n==========================");
    System.out.printf("\n\nDer gesuchte Mitarbeiter ist:\n%s ",m.toString());
    System.out.printf("\n==========================");
 System.out.printf("\nVerkäufe:");
 for(CVerkauf v:m.getMeineVerkaeufe()){
   System.out.print(v.toString());
 }
}
```

```
public void umsatzAlle(){
    double umsatz = 0;
    double max=0.0;
    String bestof=null;
System.out.printf("\n\nUmsätze aller Mitarbeiter");
    System.out.printf("\n==========================");
    for(CMitarbeiter m :meineMitarbeiter){
     System.out.print(m.toString());
     umsatz=0;
    for(CVerkauf v:m.getMeineVerkaeufe()){
     umsatz=umsatz+v.getPreis();
    }
    System.out.printf("\t\tUmsatz: %.2f €",umsatz);
    if(umsatz>max){
     bestof=m.toString();
    }

 }
     System.out.printf("\n%s ist Verkäufer des Monats",bestof);
    }
```

```
        public void selectMitarbeiter() throws ClassNotFoundException, SQLException, IOException{
          KFZVerkaufSQL  k= KFZVerkaufSQL.getInstance();
          meineMitarbeiter=k.selectMitarbeiter();
          CMitarbeiter aktuell = null;
          for(CMitarbeiter m:meineMitarbeiter){
              k.selectVerkäufe(m);

          }
        }


    }
```

Aufgabe 4 Objektorientierte Programmentwicklung (Persistenzschicht)

Klasse ConnectionManager

```
package kfzverkauf;
import java.io.*;
import java.util.*;
import java.net.*;
import java.sql.*;
public class CConnectionManager {
 private static String driver="com.mysql.jdbc.Driver";
    private static String url="jdbc:mysql://localhost/KFZVerkauf";
    private static String user="root";
    private static String password="";
    private static Connection con;
    public static Connection getConnection()
            throws IOException, ClassNotFoundException, SQLException {
        if (con == null) {

            con = DriverManager.getConnection(url, user, password);
        }
        return con;
    }
    // Verbindung zur DB schließen
    public static void closeConnection() throws SQLException {
        if (con != null) {
            con.close();
            con = null;
        }
    }
}
```

Klasse CKFZVerkauSQL

```
package kfzverkauf;
import java.io.IOException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
public class KFZVerkaufSQL {
 private static KFZVerkaufSQL verkauf = null;

    private KFZVerkaufSQL() {
    }
    public static KFZVerkaufSQL getInstance() {
        if (verkauf == null) {
            verkauf = new KFZVerkaufSQL();
        }
        return verkauf;
    }
       public ArrayList<CMitarbeiter> selectMitarbeiter()
        throws SQLException, ClassNotFoundException, IOException{
        ArrayList<CMitarbeiter> liste = new ArrayList<CMitarbeiter>();
        CMitarbeiter b = null;
        CVerkauf v=null;
        Connection con = null;
        Statement s = null;
        ResultSet rs1 = null;
        //ResultSet rs2= null;
        int mid=0;
        String name;
        String sql1 = "select mid, name from Mitarbeiter" ;
        try {
            con = CConnectionManager.getConnection();
            s = (Statement) con.createStatement();
            rs1= s.executeQuery(sql1);
            while (rs1.next()) {
                mid=rs1.getInt("mid");
                name=rs1.getString("name");
               b = new CMitarbeiter(mid,name);
                liste.add(b);
            }
            }
         finally {
            if (rs1 != null) {
                rs1.close();
            }
            if (s != null) {
                s.close();
```

```
            }
        }
        CConnectionManager.closeConnection();
        return liste;
    }
    public ArrayList<CVerkauf> selectVerkäufe(CMitarbeiter m)
        throws SQLException, ClassNotFoundException, IOException{
        ArrayList<CVerkauf> liste = new ArrayList<CVerkauf>();

        CVerkauf v=null;
        Connection con = null;
        Statement s = null;
        ResultSet rs = null;
        int mid=m.getMid();
        String name;
          String sql= "select reNr,kfz,preis from Verkaeufe where "
                + "Verkaeufe.mid="+mid;
        try {
            con = CConnectionManager.getConnection();
            s = (Statement) con.createStatement();
            rs= s.executeQuery(sql);
            while (rs.next()) {
                v=  new CVerkauf(rs.getInt("reNr"), rs.getString("kfz"),
                        rs.getDouble("preis"), m);
               }
               liste.add(v);
            }
         finally {
            if (rs!= null) {
                rs.close();
            }
            if (s != null) {
                s.close();
            }
        }
        CConnectionManager.closeConnection();
        return liste;
    }
}
```

Aufgabe 5 Programmieren der Testfälle

Klasse CKFZVerkaufUI

```
package kfzverkauf;

import java.io.IOException;
import java.sql.SQLException;

public class KFZVerkauf {

    public static void main(String[] args) throws ClassNotFoundException, SQLException, IOException {
        // TODO code application logic here
        CVerwaltung v=CVerwaltung.getInstance();
        v.selectMitarbeiter();
        v.alleVerkaufeMitarbeiter();
        v.umsatzAlle();
        v.showMitarbeiter(v.sucheMitarbeiter(14234));
    }

}
```

Aufgabe 6 *Datenbanken*

Erweitern Sie das Datenbanksystem aus Aufgabe 4 (Abbildung 3) um die Entitäten die sich aus dem folgenden erweiterten Anwendungsfall ergeben und zeichnen Sie dazu das vollständige **ER – Diagramm**. Geben Sie jeweils die **Multiplizitäten** und sinnvolle Attribute an. Kennzeichnen Sie auch die **Schlüssel.**

Die Fahrzeuge (kfz) sollen als Bestand gespeichert und um die Attribute Erstzulassung und Kilometerstand erweitert werden. Ihr Status (verkauft oder im Bestand soll auch erfasst werden.
Die Kunden (Käufer) sollen mit Kundennummer, Name, Telefon und Adresse gespeichert werden.
Die Firma unterhält ferner Geschäftsbeziehungen zu verschiedenen Dienstleistern (Autoreparatur, Aufbereitung, Leasingunternehmen, Banken….).

Lösung:

Verkauf ist Koppelentität zwischen Mitarbeiter und Kunde. Sie löst eine n:m Beziehung auf. Der Schlüssel ist ein Kombinationsschlüssel aus den Primärschlüssel von Mitarbeiter und Verkauf.
Gleiches gilt für Firma und Dienstleister. AnforderungDienst ist auch hier Koppelentität.

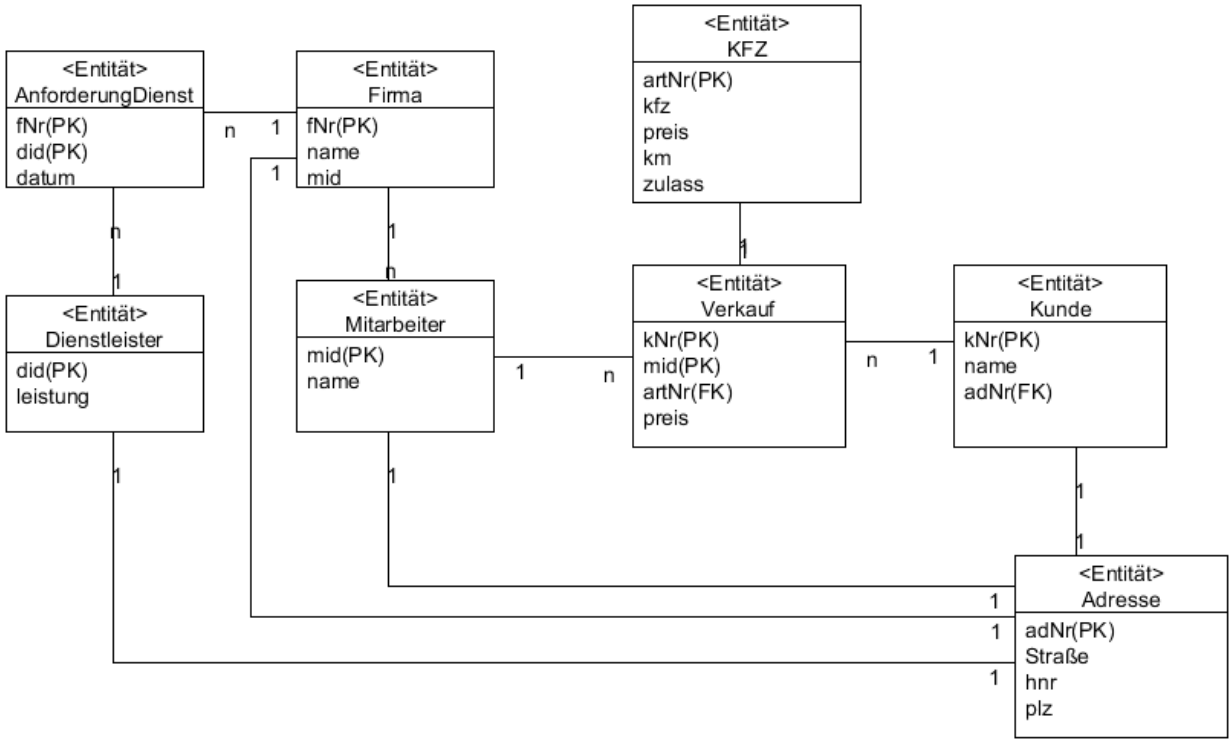

Abbildung 4: ER-Diagramm

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 -