

Klausur Nr. 1

Anmerkung:

Diese Aufgabe entstand in Anlehnung an die im Internet veröffentlichten Beispiel-Aufgaben der Landesregierung NRW zum Zentralabitur 2007 im Fach Informatik. Ich habe diese Aufgabe etwas umgewandelt und an den Unterricht in meinem Informatik-GK angepasst.

Bei dem Spiel Autoquartett geht es darum, möglichst viele Karten unter Beachtung der Spielregeln zu sammeln. Zum Spielbeginn werden die Karten gemischt und dann gleichmäßig an alle Mitspieler verteilt. Ein Spieler wird als Startspieler ausgelost. Jeder Spieler hält seine Karten in einem Kartenstapel so, dass er nur die obere Karte sieht (d. h. die anderen darf er sich nicht ansehen).

Autoquartett	
Name:	RoadStar
Hubraum:	3215 ccm
Leistung:	165 kW
Höchstgeschwindigkeit:	215 km/h

Der Startspieler nennt ein Merkmal seiner Karte, etwa „Hubraum 2954 ccm“. Alle Spieler vergleichen dieses Merkmal „Hubraum“ ihrer oberen Karte. Der Spieler, dessen obere Karte den größten Hubraum hat, erhält alle Karten und fügt sie unten an seinen Kartenstapel an. Bei gleichen größten Werten wird gelost.

Der Gewinner der ersten Runde ist nun an der Reihe und nennt ein Merkmal seiner jetzt oberliegenden Karte, usw. Falls ein Spieler keine Karten mehr besitzt, scheidet er aus. Sieger ist der Spieler, der zuletzt als einziger aktiver Spieler übrig bleibt.

Sie sollen in dieser Aufgabe Teile der Simulation des Spiels Autoquartett modellieren und programmieren.

```
public class Karte
{
    String name;
    int    hubraum;
    int    leistung;
    int    maxgesch;

    public Karte(String name,int hubraum,int leistung,int maxgesch)
    {
        this.name      = name;
        this.hubraum   = hubraum;
        this.leistung  = leistung;
        this.maxgesch  = maxgesch;
    }

    public void anzeigen()
    {
        System.out.println("Karte");
        System.out.println("=====");
        System.out.println("Name           : "+name);
        System.out.println("Hubraum        : "+hubraum);
        System.out.println("Leistung       : "+leistung);
        System.out.println("Höchstgeschw. : "+maxgesch);
        System.out.println("-----");
    }
}
```

Quelltext 1: Die komplette Klasse Karte

```
public class Tester
{
    Kartenstapel s,t;

    public Tester()
    {
        s = new Kartenstapel();
        t = new Kartenstapel();

        s.obenEinfuegen(new Karte("Karte s1", 120,120,120));
        s.obenEinfuegen(new Karte("Karte s2", 220,220,220));
        s.obenEinfuegen(new Karte("Karte s3", 320,320,320));
        s.obenEinfuegen(new Karte("Karte s4", 420,420,420));
        s.obenEinfuegen(new Karte("Karte s5", 500,500,500));
        s.obenEinfuegen(new Karte("Karte s6", 600,600,600));
        s.obenEinfuegen(new Karte("Karte s7", 700,700,700));
        s.mische();

        t.obenEinfuegen(new Karte("Karte t1", 100,100,100));
        t.obenEinfuegen(new Karte("Karte t2", 200,200,200));
        t.obenEinfuegen(new Karte("Karte t3", 300,300,300));
        t.obenEinfuegen(new Karte("Karte t4", 440,440,440));
        t.obenEinfuegen(new Karte("Karte t5", 540,540,540));
        t.obenEinfuegen(new Karte("Karte t6", 640,640,640));
        t.obenEinfuegen(new Karte("Karte t7", 740,740,740));
        t.mische();

        if (s.gibLeistung() > t.gibLeistung())
        {
            s.untenEinfuegen(t.obersteKarte());
            s.untenEinfuegen(s.obersteKarte());
            t.obenEntfernen();
            s.obenEntfernen();
        }
        else
        {
            t.untenEinfuegen(s.obersteKarte());
            t.untenEinfuegen(t.obersteKarte());
            s.obenEntfernen();
            t.obenEntfernen();
        }

        s.zeigeAlleKarten();
        t.zeigeAlleKarten();
    }
}
```

Quelltext 2: Die komplette Klasse Tester

```
// Importe
import java.util.ArrayList;
import java.util.Random;

// Quelltext der Klasse
public class Kartenstapel
{
    // Attribute
    ArrayList stapel;
    Random rand;

    // Konstruktor
    public Kartenstapel()
    {
        stapel = new ArrayList();
        rand = new Random();
    }

    // Einige interessante Methoden
    // nicht alle Methoden für Aufg. 2 sind hier aufgeführt!

    public Karte obersteKarte()
    {
        if (stapel.size() > 0)
            return (Karte) stapel.get(0);
        else return null;
    }

    public int gibLeistung()
    {
        if (stapel.size() > 0)
        {
            Karte k = (Karte) stapel.get(0);
            return k.leistung;
        }
        return 0;
    }

    public void mische()
    {
        // Ihre Aufgabe;
    }

    private void tausche()
    {
        int a = rand.nextInt(stapel.size());
        int b = rand.nextInt(stapel.size());

        if (a != b)
        {
            // ab hier muss in Aufg. 4 vervollständigt werden!
        }
    }
}
```

Quelltext 3: Auszüge aus der Klasse Kartenstapel

Aufgabenstellung

1. In Quelltext 1 sehen Sie die Klasse **Karte**. Die Implementation dieser Klasse entspricht aus bestimmten Gründen nicht dem Prinzip der **Datenkapselung**.
 - a) Erläutern Sie kurz, was man unter „Datenkapselung“ versteht und
 - b) führen Sie Gründe an, die den Autor dieser Klasse bewogen haben könnten, von diesem Prinzip abzuweichen.

2. In Quelltext 2 sehen Sie die Klasse **Tester**. Sie dient zum gründlichen Testen der Klasse **Kartenstapel**.
Analysieren Sie den Quelltext und erläutern Sie dann, wie der Autor der Klasse **Tester** die Funktionsweise der Klasse **Kartenstapel** testen will.

3. Suchen Sie sämtliche Methoden der Klasse **Kartenstapel**, die von der Klasse **Tester** benötigt werden und implementieren Sie diese Methoden.
Erläutern Sie Ihr Vorgehen.
Als Hilfestellung erhalten Sie den Quelltext 3, der einige interessante Programmzeilen aus der Klasse Kartenstapel enthält. Außerdem finden Sie im Anhang übersetzte Auszüge aus der Sun-Dokumentation der Klasse ArrayList.

4. a) Entwickeln und erläutern Sie eine Methode **mische()** und
b) vervollständigen und erläutern Sie die Methode **tausche()**.

Anhang: Die Klasse ArrayList (Auszüge)

void add(int index, Object element)

Inserts the specified element at the specified position in this list.
Fügt das angegebene Objekt an der angegebenen Position ein.

void add(Object o)

Appends the specified element to the end of this list.
Hängt das angegebene Objekt an das Ende der Liste an.

void clear()

Removes all of the elements from this list.
Löscht alle Elemente aus der Liste.

boolean contains(Object elem)

Returns true if this list contains the specified element.
Liefert true, falls die Liste das angegebene Element enthält.

Object get(int index)

Returns the element at the specified position in this list.
Liefert das Element an der angegebenen Position zurück.

Anmerkung:

Es wird nicht das Element selbst, sondern ein Zeiger auf dieses Element zurückgeliefert. Will man auf das zurückgelieferte Element zugreifen, so muss man den typecast-Operator anwenden.

int indexOf(Object elem)

Searches for the first occurrence of the given argument, testing for equality using the equals method.
Sucht nach dem ersten Vorkommen des gegebenen Objektes und liefert die Position zurück.

boolean isEmpty()

Tests if this list has no elements.
Liefert true zurück, falls die Liste keine Elemente enthält.

Object remove(int index)

Removes the element at the specified position in this list.
Entfernt das angegebene Element (wenn es vorhanden ist).

Anmerkung:

Diese Methode kann auch wie eine void-Methode benutzt werden, muss also nicht auf der rechten Seite einer Zuweisung stehen.

Beispiel:

```
remove(12);
```

Object set(int index, Object element)

Replaces the element at the specified position in this list with the specified element.
Ersetzt das Element an der angegebenen Position in der Liste durch das übergebene Objekt.

Anmerkung:

Auch diese Methode kann wie eine void-Methode benutzt werden.

Beispiel:

```
set(12, neueKarte);
```

int size()

Returns the number of elements in this list.
Liefert die Zahl der Elemente der Liste zurück.