Hochschule
Albstadt-Sigmaringen
Albstadt-Sigmaringen University

Studiengänge ITS, TI, WIN
Prof. Dr. German Nemirovski

Programmieren 2
WS 17 / 18

Aufgabenblatt 3

Grundlagen

Anzahl der Aufgaben: **5**
Abgabefrist: **10.05.18**

Aufgabe 3.0:
Gegeben sei der Code der Klasse *Airplane* (Flugzeug):

```
class Airplane {
        private String manufacturer;        // Herstellername
        private int maxSpeed;               // Max. Geschwindigkeit
        private int wingsCount = 1;         // Anzahl Flügelpaare

        public Airplane(String manufacturer, int maxSpeed, int wingsCount) {
            this.manufacturer = manufacturer;
            this.wingsCount = wingsCount;
            this.maxSpeed = maxSpeed;
          }

        public String getManufacturer() {  return this.manufacturer; }
        public void setManufacturer(String manufacturer) { this.manufacturer =
                                                                manufacturer; }

        public int getMaxSpeed() { return this.maxSpeed; }
        public void setMaxSpeed(int maxSpeed) { this.maxSpeed = maxSpeed; }

        public int getWingsCount() { return this.wingsCount; }
        public void setWingsCount(int wingsCount) { this.wingsCount = wingsCount;
                                                    }
        public boolean getLooping() { return false; }      // per default soll ein Flugzeug keinen
                                                           // looping können

 }
```

Implementieren Sie eine Klasse *TransportAircraft* sowie eine Klasse *DoubleDecker*. Diese sollen folgende Spezifikation erfüllen:

Klasse *TransportAircraft*

- Ein *TransportAircraft* (Verkehrsflugzeug) ist ein *Airplane*, das genau ein Flügelpaar sowie drei zusätzliche Instanz-Variablen für die Anzahl der Passagiere, *cruiseSpeed* (in der Regel etwas geringer als die maximale Geschwindigkeit) und den Airline-Namen hat.

- Schreiben Sie einen Konstruktor, mit dem ein *TransportAircraft*-Objekt erzeugt werden kann. Dazu müssen der *manufacturer* (String), die *maxSpeed* (int), die *airline* (String), die *cruiseSpeed* (int) und die *passengerCount* (int) angegeben werden.

- Ein *TransportAircraft* soll **keinen** Looping können.

- Die Klasse soll die fehlenden *Setter* und *Getter* implementieren.

- Implementieren Sie eine *toString()*-Methode, welche alle Eigenschaften eines *TransportAircraft*-Objekts als ein String ausgibt.

Klasse *DoubleDecker*

- Ein *DoubleDecker* ist ein Flugzeug, das genau zwei Flügelpaare hat.

- Weiter ist ein *DoubleDecker* ein akrobatiktauglicheres Flugzeug, d.h. man kann damit *Loopings* fliegen. Für einen Looping muss der *DoubleDecker* eine Mindestgeschwindigkeit von 320 km/h erreichen. Definieren Sie dafür eine Konstante LOOPINGSPEED. Die Methode *getLooping*() soll true zurückgeben, falls die zulässige max. Geschwindigkeit (*maxSpeed*) grösser als *LOOPINGSPEED* ist.

- Die Klasse *DoubleDecker* hat eine Variable *openCockpit* vom Typ *boolean.* Sie gibt an, ob der *DoubleDecker* ein offenes oder geschlossenes Cockpit hat.

- Ein *DoubleDecker mit einem offenen Cockpit darf ebenfalls* **keine** *Looping Fliegen.*

- Schreiben Sie zwei Konstruktoren, mit denen ein *DoubleDecker*-Objekt initialisiert werden kann. Der erste Konstruktor hat folgende Parameter: *manufacturer* (String), *maxSpeed* (int) und einen *openCockpit* (boolean). Der zweite Konstruktor hat folgende Parameter: *manufacturer* (String) und *maxSpeed* (int). Der default-Wert für *openCockpit* ist *true*

- Die Klasse *DoubleDecker* soll nicht erweiterbar sein.

- Implementieren Sie eine *toString()*-Methode, die alle Eigenschaften eines *TransportAircraft*-Objekts als ein String ausgibt.

Schreiben Sie eine Tester-Klasse mit der *main()* Methode, die jeweils ein Objekt der Klassen *DoubleDecker* und *TransportAircraft* anlegt, und die Eigenschaften der Objekte mithilfe der *toString()* Methode ausgibt.

Aufgabe 3.1:
Ändern Sie die Tester-Klasse für die Aufgabe 3.0 indem Sie in der *main()* Methode ein Array vom Typ *Airplane* und jeweils drei Objekten der Klassen *DoubleDecker* und *TransportAircraft* anlegen, und die Eigenschaften der Objekte in einer Schleife jeweils mithilfe der *toString()* Methode ausgeben.

Aufgabe 3.2:
Implementieren Sie eine abstrakte Klasse *FlyingBody,* welche zwei Instanz-Variablen *manufacturer und maxSpeed* deklariert und die entsprechenden Setter und Getter implementiert.

Die Klasse *FlyingBody* implementiert außerdem eine abstrakte Methode

public LocalTime calcArrivalTime(LocalTime departure, int distance)

Die Parameter haben jeweils die folgende Bedeutung:

departure – ist die Abflugzeit,

distance – ist die Distanz zum Flugziel in km.

Verändern Sie die Klasse *Airplane* so, dass sie die zwei Eigenschaften von der Klasse *FlyingBody* erbt, ohne sie selbst zu deklarieren.

Die Klasse *Airplain* soll die Methode *calcArrivalTime* nicht implementieren. Dies soll erst in den Klassen *TransportAircraft* und *DoubleDecker* erfolgen.

In der Klasse *TransportAircraft* soll die Methode *calcArrivalTime außerdem überladen werden.* Ihre zweite Version soll einen dritten Parameter bekommen *flyingWithMaxSpeed (boolean).* Wenn der Parameter den Wert *true* hat, soll die maximale Geschwindigkeit zur Berechnung der Ankunftszeit (Arrival Time) verwendet werden. Sollte der Wert des Parameters *false* sein, so wird die *cruiseSpeed* zur Berechnung verwendet.

Das Ergebnis, das die Methode *calcArrivalTime* liefert ist die geschätzte Ankunft- bzw. Landung-Zeit, die durch die Addition der Abflugzeit (*departure*) mit der geschätzten Flugdauer, Division des Parameters *distance* durch die jeweilige Geschwindigkeit des Flugzeuges, berechnet wird.

In der Klasse *DoubleDecker* soll immer die maximale Geschwindigkeit für die Berechnung der Arrival Time verwendet werden.

Schreiben Sie eine Tester-Klasse mit der *main()* Methode, die den Einsatz der beiden Methoden

calcArrivalTime(LocalTime departure, int distance)

und

calcArrivalTime(LocalTime departure, int distance, boolean flyingWithMaxSpeed)

demonstriert.

Aufgabe 3.3:
Implementieren Sie eine Klasse *Runway* (Start- und Landebahn), die zwei Instanz-Variablen hat: *length* (int) und *width* (int), also die Länge und die Breite der Bahnen.

Implementieren Sie ein Interface *Landable*, welches die folgende Methode deklariert:

public boolean landingCheck(Runway r);

Verändern Sie die Klasse *Airplane* sodass diese das Interface und damit auch die Methode *landingCheck* implementiert. Möglicherweise sind in der die Klasse *Airplane* zwei weitere Eigenschaften (Instanz-Variablen) erforderlich: *minRunwayLength* (int) und *minRunwayWidth* (int).

Dabei soll die *landingCheck* – Methode so implementiert werden, das diese überprüft, ob die Möglichkeit der Landung auf einem als Parameter gegebenen *Runway* (Landebahn) möglich ist oder nicht, d.h. welche die Länge und die Breite der Landebahn mit den minimalen erforderlichen vergleicht und einen entsprechenden boolschen Rückgabewert liefert.

Schreiben Sie eine Tester-Klasse mit der *main()* Methode, die den Einsatz der Methode *landingCheck(Runway r) demonstriert.*

Aufgabe 3.4:
Erstellen Sie ein UML-Klassendiagramm für die als Lösung der Aufgabe 3.3 entstandenen Klassenhierarchie. Nutzen Sie dafür ein beliebiges Tool oder zeichnen Sie das Diagramm auf einem Papierblatt von Hand.