Versuchen Sie immer, Ihren Code zu kommentieren!

Erstellen Sie zuerst ein Klassendiagramm für die Aufgaben!

Aufgabe 1

- Schreiben sie eine Klasse Person mit Attributen für den Nachnamen, Vornamen und das Alter. Schreiben Sie einen Konstruktor, welcher alle Eigenschaften beim Anlegen des Objektes initialisiert. Alle Eigenschaften der Klasse sollen nur durch Methoden lesbar sein. Lediglich der Nachname soll mittels einer Methode änderbar sein. Zudem soll die Klasse eine Adresse speichern können. Eine Adresse soll als eigene Klasse mit Straße und Hausnummer, PLZ und Ort gespeichert werden. Es soll eine Möglichkeit geben der Person eine Adresse hinzuzufügen bzw. durch Hinzufügen einer neuen Adresse die alte Adresse zu ändern. Person und Adresse sollen jeweils eine Methode zur Ausgabe aller Daten auf der Konsole implementieren.

Erstellen Sie im Hauptprogramm mindestens eine Person mit einer Adresse und testen Sie alle Methoden.

- Schreiben Sie eine Klasse Hund. Die Klasse Hund hat einen Namen. Zudem besitzt die Klasse zwei Methoden zum Füttern und Gassi gehen (machen jeweils eine Konsolenausgabe). Der Hund soll nur nach dem Füttern Gassi gehen. Eine Ausgabemethode soll den Namen ausgeben und auch Informationen darüber, ob der Hund schon gefüttert wurde. Eine Person muss mit dem Hund Gassi gehen. Nur Personen die älter als 16 Jahre sind, dürfen mit dem Hund Gassi gehen.

Erweitern Sie die Klasse Person um einen Hund.

Testen Sie alle Methoden der Klasse Hund mit einem Personen-Objekt aus Teil 1.

Aufgabe 2

Es soll ein Programm zur Verwaltung einer Arztpraxis entwickelt werden. Das vorgegebene Pflichtenheft sieht folgendermaßen aus:

- Über jeden Kassen-Patienten sind folgende Daten zu speichern: Patienten-Nr., Patientenname, Adresse, Geburtsdatum, Versicherten-Karte vorgelegt (ja, nein).

- Jeder Kassen-Patient gehört zu genau einer Krankenkasse.

- Jede Krankenkasse kann mehr als einen Kassen-Patienten haben (Die Krankenkasse hat in diesem Beispiel keinen Zugriff auf die Patienten).

- Über jede Krankenkasse sind folgende Daten zu speichern: Kassennummer, Kassenname.

- Wird ein neuer Kassen-Patient angelegt, dann sind die Patienten-Nr. und der Patientenname einzutragen. Gleichzeitig muss ein Kassen-Objekt für den Patienten

angegeben werden und vom Patienten zur Kasse eine Verbindung herstellen zu können.

Hinweis: Ggf. sind mehrere Patienten bei der gleichen Krankenkasse.

- Die Patienten-Nr. soll fortlaufend vergeben werden.

- Der Patientenname ist mit seinem Namen beim Erzeugen vorzubelegen. "Versicherten-Karte vorgelegt" ist mit "ja" zu initialisieren.

Implementieren Sie das erstellte Klassendiagramm in einem lauffähigen Programm. Legen Sie mindestens zwei Patienten und zwei Krankenkassen an. Geben Sie alle Daten eines Patienten auf der Konsole aus.

Aufgabe 3

Es liegt folgende Lagerdatei vor:

Artikel-nummer	Bezeichnung	Istbestand	Höchst-bestand	Preis	Verbrauch pro Tag	Bestelldauer in Tagen
100	Sofa	31	75	350,00	5	7
101	Stuhl	52	150	33,95	12	3
102	Schrank	77	100	160,50	3	21
103	Sessel	4	90	175,00	7	8
104	Tisch	28	120	99,95	5	12

Folgende Aufgaben sind zu lösen:

- Legen Sie obige Daten in einem geeigneten Format als Textdatei an.

- Identifizieren Sie notwendige Klassen mit entsprechenden Eigenschaften.

- Identifizieren Sie für die folgenden Punkte die Klasse, in welcher die einzelnen Methoden implementiert werden müssen.

- Lesen Sie die Daten aus der Textdatei und berechnen Sie den gesamten Lagerwert. Erstellen Sie eine Methode die den Lagerwert zurückliefert.

- Schreiben Sie eine Methode die einen Artikel auf dem Bildschirm ausgibt.

- Erweitern Sie die Ausgabe um zusätzliche Spalten für den Meldebestand und einen Bestellvorschlag.

- Der Meldebestand errechnet sich aus dem Verbrauch pro Tag und der Bestelldauer, wobei noch zusätzlich 2 Tage Reserve berechnet werden.

    - Ein Bestellvorschlag ergibt sich, wenn der Istbestand kleiner oder gleich dem Meldebestand ist.

    - Der Bestellvorschlag errechnet sich dann aus der Differenz zwischen Istbestand und Meldebestand.

- Schreiben Sie eine Methode die alle Artikel mit dem Meldebestand in einem Bestellvorschlag auf dem Bildschirm ausgibt. Die Ausgabe soll in Tabellenform (siehe oben) erfolgen.

Aufgabe 4

Schreiben Sie ein Programm, das folgende Klassen und Methoden für einen Koordinatensystem implementiert:

- Klasse Punkt
    - Attribute: X- und Y-Wert

- Klasse Kreis
    - Attribute: Radius und Mittelpunkt (im Koordinatensystem)
    - Methoden: Umfang und Inhalt errechnen
        - Umfang: $2 * PI * Radius$
        - Inhalt: $Pi * radius^2$

- Klasse Vektor
    - Attribute: Startpunkt und Endpunkt
    - Methode: Länge errechnen
        - Länge: $\sqrt{x^2 + y^2}$

Überlegen Sie zunächst, von welchem Typ jedes Attribut sein sollte. Legen Sie wo nötig entsprechende Get-Methoden an.

Legen Sie innerhalb der Main()-Methode mindestens ein Objekt jeder Klasse an und testen Sie alle Methoden.

Aufgabe 5

Erzeugen Sie eine Klasse TennisSpieler, die zur Speicherung von Daten über Tennisspieler (z. B. bei einem Turnier) verwendet werden könnte:

Die Klasse soll folgende Attribute erhalten: Name und Alter.

Erweitern Sie die Klasse anschließend um folgende Punkte:

- Geben Sie einen geeigneten Konstruktor für die Klasse TennisSpieler an.

- Legen Sie eine Methode BerechneAltersDifferenz() an, welcher Sie einen anderen Tennisspieler übergeben und Sie als Rückgabeparameter die Altersdifferenz zum aktuellen Tennisspieler erhalten.

- Erweitern Sie die Klasse TennisSpieler um ein Attribut namens Verfolger, welches eine Referenz auf einen weiteren Tennisspieler (den unmittelbaren Verfolger in der Weltrangliste) darstellt und passen Sie Ihren bisherigen Konstruktor zur Initialisierung dieses neuen Attributs sinnvoll an.

    - Hinweis: Tatsächliche Reihenfolgen und Punktestände spielen in diesem Beispiel keine Rolle!

- Erstellen Sie eine Methode Ausgabe(), welche den Namen des Spielers und den Namen seines Verfolgers ausgibt.

- Erweitern Sie die Klasse TennisSpieler um eine Methode IstLetzter(), die genau dann den Wert true liefert, wenn das Tennisspieler-Objekt keinen Verfolger in der Weltrangliste hat.

Aufgabe 6

Die Klasse MyLinkedQueue realisiert eine Warteschlange mit Hilfe einer einfach verketteten Liste. Eine Warteschlange besitzt die Eigenschaft, dass Elemente nur am Ende der Schlange hinzugefügt und am Anfang der Schlange entfernt werden können. Die einfach verkettete Liste besteht aus Objekten vom Typ Entry.

Legen Sie zunächst eine Klasse Entry an, welche ein Attribut vom Typ string besitzt (Name: data) und ein Attribut vom Typ Entry (Name: next). Implementieren Sie einen Konstruktor, welcher den String als Übergabeparameter entgegennimmt und zuweist. Legen Sie zudem die nötigen Get- & Set-Methoden für die Attribute der Klasse an.

Legen Sie dann die Klasse MyLinkedQueue an, welche zwei Attribute vom Typ Entry besitzt (Namen: head und tail).

Implementieren Sie für diese Klasse MyLinkedQueue die folgenden Methoden:

- IsEmpty(): Überprüft, ob die Schlange leer ist. Ist dies der Fall, wird true zurückgegeben, ansonsten false.

- Enqueue(): Fügt einen übergebenen String data in ein neues Element am Ende der Warteschlange ein.

- Print(): Gibt alle Elemente der Schlange beginnend beim Kopf aus.

- Dequeue(): Entfernt das Element am Anfang der Warteschlange und gibt den darin gespeicherten String als Ergebnis zurück. Ist die Schlange leer, wird null zurückgegeben.

Nach der Implementierung soll die Warteschlange z.B. wie folgt verwendet werden:

```
MyLinkedQueue queue = new MyLinkedQueue();

queue.Enqueue("1. Eintrag");
queue.Enqueue("2. Eintrag");

queue.Print();

…
```

Aufgabe 7

Teil 1

Erstellen Sie eine Klasse Rennschnecke. Rennschnecken sollen folgende Eigenschaften besitzen:

- Name
- Maximalgeschwindigkeit (Veränderung der Strecke pro Schritt)
- Die Schnecke soll wissen welchen Weg sie bereits zurückgelegt hat

Legen Sie in der Klasse Rennschnecke folgende Methoden an:

- Einen Konstruktor, der den Instanzvariablen beim Erstellen einer neuen Instanz Werte zuweist.

- Krieche()
    - Bewegt die Schnecke abhängig von ihrer Maximalgeschwindigkeit eine zufällige Strecke weiter. D.h. sie kriecht eine zufällige Strecke größer 0 und kleiner ihrer Maximalgeschwindigkeit.

- Ausgabe()
    - Gibt die Daten der Schnecke mit return als String zurück.

- Alle notwendigen Get-Methoden.

Teil 2

Erstellen Sie eine Klasse Rennen. Ein Rennen hat folgende Eigenschaften:

- Name
- Maximale Anzahl der teilnehmenden Schnecken
- Die teilnehmenden Schnecken selbst (als Array)
- Die Länge der zu kriechenden Strecke

Überlegen Sie, welche dieser Werte bereits im Konstruktor gesetzt werden sollten.

Legen Sie in der Klasse Rennen folgende Methoden an:

- AddRennschnecke(Rennschnecke neueSchnecke)
    - Fügt dem Rennen eine Schnecke hinzu.

- Ausgabe()
    - Gibt die Daten des Rennens als String zurück.

- ErmittleGewinner()
    - Liefert null zurück, wenn noch keine der teilnehmenden Schnecken das Ziel erreicht hat und anderenfalls die Gewinnerschnecke.

- LasseSchneckenKriechen()
    - Lässt alle teilnehmenden Schnecken einmal kriechen.

- Durchführen()
    - Es wird so lange LasseSchneckenKriechen() aufgerufen, bis eine der Schnecken das Ziel erreicht hat.

- IstRennteilnehmer(string schneckenName)
    - Liefert true wenn die angegebene Schnecke in der Liste der vorhandenen Schnecken enthalten ist.

Teil 3

Erstellen Sie eine Klasse Wettbüro.

Ein Wettbüro hat die folgenden Eigenschaften:
- Das Rennen, für welches es seine Wetten entgegennimmt
- Eine Liste von angenommenen Wetten
- Fester Faktor, mit welchem Wetteinsätze bei einem Gewinn multipliziert werden. Z.B. ganzzahliger Wert größer 1

Die Klasse Wettbüro hat folgende Methoden:
- Einen Konstruktor , der Rennen und Faktor entgegen nimmt.

- WetteAnnehmen(string schneckenName, int wettEinsatz, string spieler)
    - Nimmt eine Wette entgegen. Die Wette ist bezogen auf eine Schnecke für das Rennen, das von dem Büro bearbeitet wird.
    - Hinweis: Um die einzelnen Wetten speichern zu können, sollten die Wett-Daten in eigenen Objekten der Klasse Wette mit entsprechenden Get-Methoden gespeichert werden.

- RennAblauf()
    - Führt das betreute Rennen durch.

- Ausgabe()
    - Gibt die Daten des Wettbüros, die Daten des Rennens sowie sämtliche abgeschlossene Wetten als String zurück.

Zusätze Teil 3

- Dieselbe Schnecke kann in ein Rennen zweimal eingetragen werden.
- Es können negative Wetten abgeschlossen werden.
- Gehen zwei Schnecken gleichzeitig durchs Ziel, wird die Schnecke ausgegeben, auf die die Suche nach dem Gewinner zuerst stößt.