Versuchen Sie immer, Ihren Code zu kommentieren!

Erstellen Sie für jede Aufgabe ein Klassendiagramm, bevor Sie mit der Aufgabe beginnen.

Aufgabe 1

Modellieren Sie die die Klassen Auto und Pickup nach den folgenden Vorgaben. Eigenschaften von Objekten dieser Klassen sollen ausschließlich beim Erzeugen gesetzt und ggf. durch Methoden verändert werden können.

Klasse Auto:

Ein Auto soll ein Kennzeichen, einen Kilometerstand und eine Anzahl Sitzplätze haben. Wenn nicht anders angegeben, ist das Kennzeichen eines Auto–Objekts standardmäßig »DO-OM 3«, es sollen aber auch Autos mit anderen Kennzeichen erzeugt werden können. Ein neu erzeugtes Auto hat grundsätzlich noch keine Kilometer gefahren. Wenn nicht anders angegeben wird hat ein Auto 5 Sitzplätze. Es sollen jedoch auch zweisitzige Autos erzeugt werden können. Zu guter Letzt hat ein Auto eine Antenne, die aus- oder eingefahren werden/sein kann. Die Klasse soll mindestens die folgenden Methoden besitzen:

- GetKennzeichen() : liefert das Kennzeichen als String zurück
- GetKilometerstand() : liefert den aktuellen Kilometerstand zurück
- Fahre(int kilometer) : schreibt eine Meldung an die Standardausgabe, dass das Auto x Kilometer fährt, und setzt dann den Kilometerstand entsprechend weiter
- AntenneEinfahren(): schreibt eine Meldung an die Standardausgabe, dass die Antenne eingefahren wird, und setzt den Status der Antenne
- AntenneAusfahren() : schreibt eine Meldung an die Standardausgabe, dass die Antenne ausgefahren wird, und setzt den Status der Antenne
- IstAntenneDraussen() : liefert true, wenn die Antenne ausgefahren ist ansonsten false
- VorDemWaschen() : bereitet das Auto auf den Aufenthalt in der Waschstraße vor – konkret bedeutet das bei einem Auto, dass die Antenne eingefahren wird
- Waschen() : bereitet das Auto für eine Autowäsche vor und schreibt dann eine Meldung an die Standardausgabe, dass der Wagen gewaschen wird
- GetSitzplätze() : liefert die Anzahl Sitzplätze zurück
- ToString() : gibt eine passende String–Repräsentation des Autos zurück

Klasse Pickup:

Ein Pickup ist eine spezielle Art von Auto. Es hat nur 2 Sitzplätze, dafür jedoch zusätzlich eine offene Ladefläche mit einem beim Erzeugen bestimmbaren Fassungsvermögen. Die Ladefläche ist beim Erzeugen des Pickups zunächst leer und kann maximal mit x Dingen beladen werden. Die Klasse muss mindestens folgende Methoden besitzen:

- Beladen(int ladungsMenge) : falls auf der Ladefläche noch genügend Platz ist, wird der Inhalt der Ladefläche um x Dinge erhöht, eine passende Meldung an die Standardausgabe geschrieben, und true zurückgegeben, andernfalls false

- Entladen(int ladungsMenge) : falls auf der Ladefläche mindestens x Dinge vorhanden sind, wird der Inhalt der Ladefläche um x Dinge verringert, eine passende Meldung geschrieben, und true zurückgegeben, andernfalls false

- VorDemWaschen() : bei einem Pickup soll vor dem Waschen nicht nur die Antenne eingefahren werden, sondern auch die Ladefläche geleert werden

- GetLadung() : liefert die Anzahl Dinge, die derzeit auf der Ladefläche liegen, zurück, ohne den Ladezustand zu verändern

- ToString() : gibt eine passende String–Repräsentation des Pickups zurück

Definieren Sie in der Main–Methode mehrere verschiedene Autos und Pickups. Dort ist unter anderem Folgendes zu implementieren:

- Lassen Sie sich die Kennzeichen und sonstigen Zustände der erzeugten Objekte ausgeben

- Lassen Sie die Autos und Pickups fahren und waschen

- Testen Sie die Beladelogik der Pickups

Aufgabe 2

1) Entwerfen Sie die Klasse Haustier wie folgt:

Attribute

- name
- steuerpflichtig
- jahreskostenTierarzt

Methoden

- Lesemethode für den Namen
- Lesemethode für die Steuerpflicht
- Lesemethode für die Kosten
- Konstruktor der Namen, Steuerpflicht und Jahreskosten für den Tierarzt erfasst
- Methode GetBeschreibung() vom Typ String. Gibt Text mit Namen und eventueller Steuerpflicht zurück

2) Entwerfen Sie die Klasse Hund, welche von Haustier abgeleitet wird:

Attribute
- Hunde sind steuerpflichtig
- Attribut: rasse

Methoden
- Konstruktor der den Namen, die Jahreskosten für den Tierarzt und die Rasse erfasst
- Lesemethode für die Rasse
- GetBeschreibung() die zusätzlich hundespezifische Daten zurückliefert

3) Entwerfen Sie die Klasse Katze, welche von Haustier abgeleitet wird:

Attribute
- Katzen sind nicht steuerpflichtig
- Attribut: lieblingsVogel (Ist eine Referenz auf ein Vogel-Objekt)

Methoden
- Konstruktor, der den Namen der Katze, die Jahreskosten für den Tierarzt und die Referenz des Lieblingsvogels erfasst
- GetVogelName() Name des Vogels als Zeichenkette
- GetBeschreibung() die zusätzliche katzenspezifische Daten zurückliefert
- Setzen des Lieblingsvogelattributes

4) Entwerfen Sie die Klasse Vogel, welche von Haustier abgeleitet wird:

Attribute
- Vögel sind nicht steuerpflichtig
- Attribut: singvogel (boolscher Wert)

Methoden
- Konstruktor, der den Namen des Vogels, die Jahreskosten für den Tierarzt und die Singvogeleigenschaft erfasst
- Lesemethode für Singvogeleigenschaft
- GetBeschreibung() die zusätzliche vogelspezifische Eigenschaften zurückliefert

5) Schreiben Sie ein Hauptprogramm das folgende Aufgaben ausführt:

- Verwaltung von mindestens 2 Haustieren pro Tierart einem gemeinsamen statischen Array vom Typ Haustier

Methoden
- AnlegenHaustiere(): Anlegen von mindestens 2 Tieren pro Tierart im statischen Haustierfeld

- SetNeuerLieblingsvogel(): Benutzen Sie einen Vogel, der aus dem Haustierfeld geholt wird und weisen Sie ihn einer Katze aus dem Haustierfeld zu
    - Prüfen Sie vor den Zuweisungen, dass Sie eine Katze, beziehungsweise einen Vogel aus dem Feld ausgelesen haben

- Ausgeben(): Iterieren über das Array hausTiere und geben Sie folgende Information aus:
    - Typ des Tieres
    - Inhalt von GetBeschreibung()
    - Summieren Sie die Tierarztkosten auf und geben Sie sie am Ende aus

Hinweis: Sie können mittels objektName.GetType().Name den Namen der Klasse ermitteln.

Aufgabe 3

In einem mittelalterlichen Königreich soll das Finanz- und Steuerwesen auf EDV umgestellt werden. Die verschiedenen Bevölkerungsgruppen sind König, Adel, Bürger und Leibeigener. Alle Bevölkerungsgruppen benötigen die folgenden Attribute und Methoden und sollen von der Klasse Einwohner abgeleitet werden.

Attribute:

- `einkommen`

Methoden:

- `SetEinkommen()`
- `GetZuVersteuerndesEinkommen()`
- `BerechneSteuern()`

Erstellen Sie eine entsprechende Vererbungshierarchie und überlegen Sie, welche Attribute und Methoden in welchen Klassen benötigt werden. Die Eigenschaft einkommen gibt das tatsächliche Jahreseinkommen eines Einwohners in Talern an.

Die Methoden GetZuVersteuerndesEinkommen() und BerechneSteuern() sollen die für jeden Einwohner des Königreiches korrekten Werte gemäß der folgenden königlichen Vorschriften liefern:

1. Sofern dieses Gesetz nichts Gegenteiliges aussagt, hat jeder Einwohner sein gesamtes Jahreseinkommen zu versteuern.

2. Jeder Einwohner hat 10% seines zu versteuernden Einkommens als Steuer zu entrichten. Der Steuerbetrag wird auf ganze Taler abgerundet, jedoch beträgt die Steuer immer mindestens einen Taler.

3. Der König zahlt auch für sein steuerpflichtiges Einkommen keine Steuern.

4. Für Angehörige des Adels beträgt die Steuer mindestens 20 Taler.

5. Bei Leibeigenen sind 12 Taler des Jahreseinkommens steuerfrei.

Da jährliche Änderungen bei der Steuerberechnung zu erwarten sind, darf die Grundregel (2.) änderungsfreundlich nur an einer Stelle der Klassenhierarchie implementiert werden!

- Implementieren Sie die Klassenhierarchie! Überlegen Sie sich zunächst, wie die Methoden in der Klasse Einwohner implementiert werden müssen. Welche Methoden müssen in den Unterklassen überschrieben werden?

- Testen Sie Ihre Klassen, indem Sie ein Objekt aus jeder Bevölkerungsgruppe anlegen und das Einkommen jeweils auf 100 Taler festlegen und anschließend die Steuern ausgeben.

Schreiben Sie anschließend eine Klasse SteuerEintreiber, welcher alle Bewohner des Königreiches kennt und daher die gesamten Steuereinnahmen des Königreichs ermitteln kann.

Aufgabe 4

Netze aus Widerständen bestehen aus einzelnen Widerständen. Diese sind entweder in Reihe oder parallel geschaltet.

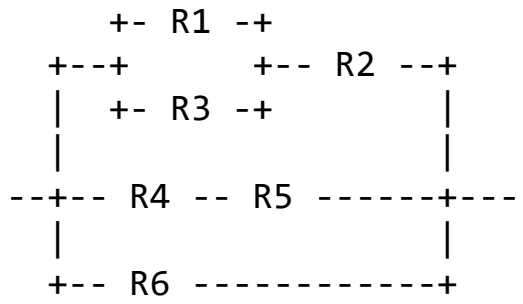
- Um einen einfachen Widerstand darzustellen, erstellen Sie die Klasse Widerstand. Diese Klasse soll eine double-Variable enthalten, um den Widerstandswert zu speichern. Zur Variable soll es eine Get-Methode geben. Zudem hat ein Widerstand einen Namen (z.B. R1) und ebenfalls eine Get-Methode für den Namen. Außerdem soll es einen Konstruktor mit einem string- und einem double-Parameter geben, der die Objektvariablen initialisiert. Es gibt in der Klasse eine Methode zur Ausgabe des Widerstandes mit Name und Widerstandswert des Widerstandes.

- Zur Darstellung eines Widerstandsnetzes legen Sie zuerst eine abstrakte Klasse Widerstandsnetz an, die von Widerstand erbt. In der Klasse gibt es eine Liste zur Speicherung aller Widerstände des Netzes. Die Klasse soll eine abstrakte Methode zur Widerstandsberechnung besitzen. Widerstandsnetz hat einen Konstruktor, dem eine beliebige Anzahl von Widerständen übergeben werden kann (mindestens jedoch zwei). Im Konstruktor wird die Methode zur Widerstandberechnung aufgerufen. Zudem wird der Name des Netzes erzeugt. Nutzen Sie hierfür eine virtuelle Methode ErzeugeName(). Bei einer Reihenschaltung, bestehend aus R3 und R5, lautet der Name Rr: R3,R5. Bei einer Parallelschaltung von R3 und R5 würde der Name Rp: R3,R5 lauten.

- Zur Berechnung eines in Reihe geschalteten Widerstandnetzes legen Sie die Klasse Reihenschaltung an, die von Widerstandsnetz erbt. Die geerbte abstrakte Methode berechnet den Gesamtwiderstand mit der Formel: $R_r = R1 + R2 + \cdots + Rn$

- Für parallel geschaltete Widerstände soll die Klasse Parallelschaltung erstellt werden. Die Formel für eine Parallelschaltung lautet: $Rp = 1/(1/R1+1/R2+\cdots+1/Rn)$

Erstellen Sie im Hauptprogramm ein Widerstandsnetzwerk nach folgendem Aufbau:



Die Widerstandswerte ergeben sich aus dem Namen des Widerstandes. R1 hat 100 Ohm, R2 hat 200 Ohm, usw. Berechnen Sie den Gesamtwiderstand des Netzes.

Der Gesamtwiderstand dieses Netzes beträgt 155,9055 Ohm.

Aufgabe 5

Ein Verein möchte sich ein klares Bild von der Einnahmen- und Ausgaben-Struktur seiner Mitglieder machen. Mitglieder können dabei entweder physische Personen oder wieder Vereine (also Zweigstellen des Vereins) sein. Alle Mitglieder (Klasse Mitglied) haben einen Namen und eine eindeutige Mitgliedsnummer. Die Mitgliedsnummer wird beim Anlegen eines neuen Mitgliedes vom System vergeben. Es existieren folgende Arten von physischen Personen als Mitgliedern:

- Unterstützende Mitglieder (Klasse UnterstützendesMitglied)
  Diese Mitglieder bezahlen einen Jahresbeitrag von €100,- und verursachen bei Vereinsfesten Ausgaben von durchschnittlich €15,- im Jahr.

- Aktive Mitglieder (Klasse AktivesMitglied)
  Diese Mitglieder besitzen einen ganzzahligen Aktivitätsgrad im Bereich von 0 bis 10. Sie gliedern sich in:

  - Spitzensportler (Klasse SpitzenSportler)
    Monatlicher Beitrag: €10,-
    Monatliche Ausgaben in €: Aktivitätsgrad * 5

  - Amateure (Klasse AmateurSportler)
    Monatlicher Beitrag: €25,-
    Monatliche Ausgaben in €: Aktivitätsgrad * 1,5

  - Trainer (Klasse Trainer)
    Monatlicher Beitrag: €10,-
    Monatliche Ausgaben in €: Aktivitätsgrad * 50

- Vorstandsmitglieder (Klasse Vorstandsmitglied)
  Vorstandsmitglieder haben einen ganzzahligen Kompetenzwert im Bereich 0 bis 10. Ein Vorstandsmitglied erzeugt durch das Gewinnen von Sponsorengeldern und

Förderungen Jahreseinnahmen von Kompetenz * 100 € und verursacht Ausgaben, indem es 20% Provision für erzeugte Einnahmen erhält.

Vereine können ebenfalls Mitglieder eines Vereins sein:

- Verein (Klasse Verein)
  Die Klasse Verein speichert eine Liste von Mitgliedern und kann die gesamten Einnahmen, Ausgaben und den Überschuss berechnen. Stellen Sie die Methode MitgliedHinzufügen(Mitglied mitglied) zur Verfügung. Die maximale Mitgliederanzahl eines Vereins wird beim Anlegen festgelegt und darf nicht überschritten werden.

Die Klasse Mitglied soll folgende Methoden besitzen:

- double GetEinnahmen()
  Berechnet die gesamten Einnahmen, die der Verein durch dieses Mitglied erzielt.

- double GetAusgaben()
  Berechnet die gesamten Ausgaben, die dieses Mitglied verursacht.

- double GetÜberschuss()
  Berechnet den finanziellen Überschuss, den das Mitglied dem Verein bringt.

- void Ausgabe()
  Gibt eine ordentlich formatierte und strukturierte Mitgliederliste (Name, Einnahmen, Ausgaben, Überschuss) aus.

Aufgabe 6

Teil 1

Ein Versandservice versendet Briefe und Pakete und identifiziert diese über eine ID. Zusätzlich werden der Adressat und der Absender gespeichert, sowie ein Flag, das anzeigt, ob das Versandstück bereits zugestellt wurde. Entwerfen Sie eine Klasse Postsendung, welche die gemeinsamen Eigenschaften eines Briefes und eines Paketes darstellt.

Jedes Versandstück hat zwei Adressen. Definieren Sie daher zunächst eine Klasse Adresse, die folgende Adressangaben verwaltet:

- nachname
- vorname
- strasse + hausnummer
- plz + ort
- land

Die Elemente der Klasse Adresse sind vom Typ `string`. Stellen Sie zur Initialisierung einen Konstruktor mit Parametern für alle Werte bereit. Der letzte Parameter für das Land kann ein

Standardwert, z.B. „Deutschland“, sein. Eine Adresse mit leeren Strings – mit Ausnahme des Landes – ist ungültig. Ob eine Adresse gültig oder ungültig ist, soll mit der Methode IstGültig() abgefragt werden können. Legen Sie zudem notwendige Getter und Setter, sowie eine ToString() Methode an.

Entwerfen Sie eine Klasse Postsendung mit den Elementen id für die ID, absender und empfänger zum Speichern der Adressen sowie einem Flag zugestellt, das den Wert true hat, falls die Post ausgeliefert wurde.

Definieren Sie einen Konstruktor, der als Argument die ID des neuen Postsendungs-Objektes erhält, sowie die Adresse des Empfängers und die des Absenders.

Stellen Sie Zugriffsmethoden zum Lesen und Schreiben des Senders und Empfängers bereit, wobei eine neue Adresse nur gesetzt werden kann, wenn sie gültig ist.

Zudem soll ein Versandstück nur ausgeliefert werden können, wenn die Empfängeradresse gültig ist. Definieren Sie die Methode ToString(), welche einen formatierten String der Postsendung zurückliefert.

Leiten Sie von der Klasse Postsendung die Klassen Brief und Paket zur Darstellung von Briefen und Paketen ab.

Ein Brief kann als Standardbrief, Eilbrief und Einschreiben verschickt werden. Definieren Sie einen entsprechenden Aufzählungstyp Briefkategorien und erweitern Sie die Klasse um ein privates Datenelement dieses Typs.

Die Klasse Brief stellt einen Konstruktor zur Verfügung, der ID, Absender, Empfänger und einen weiteren Parameter für die Kategorie des Briefes mit dem Default-Wert „Standard“ besitzt.

Jedes Paket hat ein bestimmtes Gewicht, das für den Transportpreis maßgeblich ist (der Preis wird nicht berechnet). Außerdem kann ein Paket versichert werden. Definieren Sie in der Klasse Paket ein Fließkommazahlen-Element für das Gewicht und einen booleschen Wert für die Versicherung.

Legen Sie notwendige Getter und Setter an und redefinieren Sie die Methode ToString()in Brief und Paket.

Teil 2

Entwickeln Sie die Klasse VersandService zur Verwaltung von Briefen und Paketen für einen Versandservice. Die Klasse besitzt eine Liste für alle Postsendungen (siehe auch Teil 3).

Die Klasse VersandService enthält die Methoden NeuerBrief() und NeuesPaket(), welche jeweils einen Brief bzw. ein Paket erzeugen. Den Methoden werden Absender und Empfänger übergeben. Zudem die für die jeweilige Postsendung notwendigen Attribute. Die Klasse VersandService soll die ID für eine neue Postsendung fortlaufend vergeben. Die erzeugten Postsendungen sollen in die Liste eingefügt werden.

Die Klasse VersandService hat eine Methode Ausliefern(), welche bei allen Postsendungen mit gültigem Absender und Empfänger das Attribut zugestellt auf true setzt.

Teil 3

Erweitern Sie die vorhandene Klassenhierarchie um eine Klasse für Pakete, welche eine Sendeverfolgung unterstützen. Für die Verfolgung werden Ort und Uhrzeit gespeichert. Leiten Sie von der Klasse Paket die Klasse VerfolgbaresPaket ab. Erweitern Sie die Klasse VerfolgbaresPaket um eine Liste mit Elementen vom Typ Station. Die Hilfsklasse Station enthält einen String zum Speichern des Ortes und einen Zeitstempel vom Typ DateTime. Die Klasse Station hat einen Konstruktor, welchem der Ort übergeben wird und welcher den Zeitstempel auf die aktuelle Uhrzeit und Datum setzt (mittels DateTime.Now). Die Methode ToString() von Station liefert Ort und Zeitstempel als String.

Die Klasse VerfolgbaresPaket hat einen Konstruktor mit allen notwendigen Attributen. Der Ort des Absenders soll automatisch als erste Station gesetzt werden.

VerfolgbaresPaket kann mittels der Methode SetzeStation() ein Ort übergeben werden. Die Methode sorgt dafür, dass eine neue Station in die Paketverfolgung hinzugefügt wird. Die Methode ToString() von VerfolgbaresPaket liefert alle Informationen zum Paket inklusive der Stationen als formatierten String.

Teil 4

Erstellen Sie eine Konsolen-Anwendung mit welcher Sie Adressen, Briefe, Pakete und verfolgbare Pakete anlegen, anzeigen und ausliefern können.