Wiederholungs-/Vertiefungsaufgaben Klassen

Dieses Aufgabenblatt hilft euch dabei, Klassen zu wiederholen und zu vertiefen.
Wir werden dieses Blatt im nächsten Tutorium besprechen. Es finden sich allerdings schon in unserem Moodle und auf GitHub (diese Lösungen werden jeweils in den einzelnen Aufgaben verlinkt).

Falls es zu Unstimmigkeiten in den Aufgaben oder Lösungen kommen sollte, gebt bitte kurz Bescheid.

Eine Vervielfältigung außerhalb des Tutoriums nur unter Absprache erlaubt!

Aufgabe 1

Schreibe eine Klasse Student, welche folgende Felder besitzt:
- **string matrikelnummer** (warum string und kein int oder uint?)
- **string name**, **string vorname**
- **int fachsemester**
- **double notendurchschnitt**

Diese Felder sollen nicht einfach so verändert werden können!
Das heißt, dass der Aufruf **student1.name = "Test"** nicht gültig sein darf ➔ welche Zugriffsmodifizierer müssen die Felder besitzen?

Auf der Klasse Student sollen mehrere Konstruktorenaufrufe möglich gemacht werden.
Zunächst kann ein neues Studenten-Objekt erstellt werden können, ohne Parameter in den Konstruktor mit zu übergeben.
Dem gegenüber steht ein Konstruktor, welche **alle** Felder mit übergeben bekommt. Hierbei soll allerdings die Validität der Parameter überprüft werden.
Die einzelnen Parameter sind dann valide, wenn:
- **matrikelnummer** muss eine positive, ganze, 7-stellige Zahl sein, welche zwischen 0000001 und 9999999 liegt.
- **name** und **vorname** haben keine weitere Begrenzungen, sie dürfen nur nicht leer sein (empty-string).
- **fachsemester** muss eine positive, ganze Zahl sein, welche zwischen 1 und 10 liegt.
- **notendurchschnitt** muss zwischen 1,00 und 4,00 liegen. Zudem soll der **notendurchschnitt** nur 2 Nachkommastellen haben. Falls eine genauere Zahl angegeben wird, wird auch hier ein Fehler geworfen (wir runden nicht für den Aufrufer!).

Falls eine der übergebenen Parameter diese Voraussetzungen zur Validität nicht erfüllt, wird ein Fehler geworfen und kein Objekt erstellt. Es soll ein entsprechender Fehler geworfen werden, der dem Aufrufer des Konstruktors den Fehler klar und deutlich beschreibt, damit Änderungen vorgenommen werden können.

Da ein Studenten-Objekt auch mit leeren Konstruktor erstellt werden kann, muss dem Benutzer die Möglichkeit geboten werden, die Felder im Nachhinein zu ändern.
Hierzu sollen, wie eingangs erwähnt, nicht direkt die Felder angesprochen werden können! (welcher Umweg fällt hier ein?). Auch erlaubt soll der lesende Aufruf dieser Felder sein.
Auch bei dem Setzen der Felder im Nachhinein soll die oben erwähnte Fehlererkennung greifen (wie kann redundanter [/doppelter] Code verhindert werden?).

In der Klasse Student wird zudem noch die ToString(…)-Methode **überschrieben**. Es soll eine Zusammenfassung des Studenten-Objektes ausgegeben werden. Wie die Formatierung genau aussieht, ist hierbei erstmal egal. Es ist nur darauf zu achten, dass bei mehreren ausgegebenen Studenten die Übersicht (relativ) einfach fällt.

Abschließend erhält die Klasse Student eine Methode SageHallo(), welche keinen Parameter enthält und auch keinen Rückgabetyp hat. Diese Methode soll auf der Konsole „Hallo, mein Name ist <name>" ausgeben. Ob der <name> aus Name und Vorname besteht ist egal.

Musterlösung.

Aufgabe 2

Nun wollen wir die Studentenklasse verwalten. Dazu erstellen wir eine neue Klasse Hochschule, welche mehrere Studenten in einer Liste verwalten kann. Eine Unterscheidung nach Fakultäten wird in dieser Aufgabe nicht berücksichtigt.

Die Klasse Hochschule hat zwei Felder, welche zunächst nur einen lesenden Zugriff erlauben.
- **private Student[] eingeschriebeneStudenten**
- **private int anzahlStudenten**
- **private string name** (der Name darf auch einen schreibenden Zugriff erlauben, mit entsprechender Fehlerbehandlung: **name** darf nicht leer sein und muss mindestens die Länge 5 haben)

Die Felder werden durch einen Konstruktoraufruf gesetzt. Die Klasse Hochschule soll nur einen Konstruktoraufruf **mit** den Parametern **eingeschriebenenStudenten** und **name** unterstützen.
Bei diesem Konstruktoraufruf soll nur darauf geachtet werden, dass das übergebene Array mindestens einen Studenten besitzt.

Auch in dieser Klasse, soll — analog zur Klasse Student — die Methode ToString(…) überschrieben werden. Bei einem (indirekten) Aufruf dieser Methode soll nur der Name und die Anzahl der Studenten ausgegeben werden (mit etwas Text drum herum: „Die Hochschule mit dem Namen…“)

Nun soll die Funktionalität der Klasse erweitert werden. Die Klasse Hochschule soll folgende Methoden mit folgenden Eigenschaften/Funktionalitäten bekommen:

- **SucheStudent(string matrikelNummer)**: soll einen Studenten anhand der Matrikelnummer suchen und diesen zurückgeben (➔ was ist der Rückgabetyp?). Falls kein Student gefunden wird, soll eine Fehlermeldung ausgegeben werden.

- **GebeStudentenAus(string matrikelNummer):** analog zur Methode SucheStudent(…), allerdings soll hier der Student nicht zurückgegeben werden, sondern lediglich auf der Konsole ausgegeben werden

- **Peek(int index)**: diese Methode soll einfach den Student im internen Array **eingeschriebeneStudenten** am spezifizierten Index zurückgeben.
  Falls hierbei ein Index, welche nicht in dem Array liegt, übergeben wird, soll eine entsprechende Fehlermeldung ausgegeben werden. Es soll nicht versucht werden, das Array an diesem Index anzusprechen.

Neben diesen Methoden, soll die Funktionalität der Methode **Peek(int index)** auch direkt über die Klasse Hochschule möglich gemacht werden. Das heißt, dass der folgende Aufruf zu keinem Fehler führt, und auch nach unseren Erwartungen funktioniert:

```
Hochschule thnbg = new Hochschule(…);
Student ersterStudent = thnbg[0];
Console.WriteLine(ersterStudent);
```

Analog zur „normalen“ Methode, soll auch hier bei einem nicht-validen Index ein entsprechender Fehler ausgegeben werden.

Zuletzt soll der Operator „+“ überladen werden. Die Definition der Addition zweier Hochschulen ist nicht trivial und schwer zu fassen. Wir definieren dies aber folgend (der Additionsoperator verhält sich folgend):

- es werden die Arrays **eingeschriebeneStudenten** der beiden Hochschulen zusammenaddiert.
- der Name der neuen Hochschule ist: „Kombi-HS. aus: <name1> und <name2>“.
- es wird ein neues Hochschul-Objekt mit diesen Feldern erzeugt und zurückgegeben.

Es soll zudem eine Meldung über das erfolgreiche Anlegen dieser Kombi-Hochschule auf der Konsole ausgegeben werden.

Musterlösung.

Aufgabe 3

Die Funktionalität der Klasse Hochschule soll nun erneut etwas erweitert werden.
Da es nun vorwiegend über Rechengesetze und -operationen geht kapseln wir dies in einer eigenen Aufgabe.

Die Klasse Hochschule erhält eine Methode mit dem Namen **durchschnittlicherNotendurchschnitt(…)**, welchen Rückgabewert wir wie folgt definieren:

$$\frac{\sum_{i=0}^{n-1}(student_i.notendurchschnitt)}{n}$$

Es sollen also alle **notendurchschnit**te aller Studenten zusammengezählt, und durch die Anzahl der Studenten geteilt werden, damit ein Durchschnitt aller Notendurchschnitte entsteht (**auch**: arithmetisches Mittel).
Diese Method erhält keine Parameter und gibt das Ergebnis dem Aufrufer zurück (gerundet auf zwei Stellen).

Da das arithmetische Mittel sehr empfindlich gegenüber Ausreißern an beiden Enden ist, soll eine weitere Methode implementiert werden: **medianNotendurchschnitt(…).** Definiert ist der Median wie folgt.
Zusammengefasst kann aber gesagt werden, dass der Median bei einer geraden Zahl von Studenten das mittlere Element (der mittlere Notendurchschnitt) des sortieren Arrays ist, bei einer ungeraden Anzahl das arithmetische Mittel der beiden mittleren Elemente.

Ein Algorithmus für die Methode lässt sich also wie folgt definieren:

- Sortieren des Arrays **eingeschriebeneStudenten** nach dem Notendurchschnitt
- Berechnen des (aufgerundeten) mittleren Index des Arrays: $\lceil \frac{1}{2} \times n \rceil = k$ (Gauß-Klammern)
- Bei einer ungeraden Anzahl Studenten wird das arithmetische Mittel des Elements an den Indizes k und $k + 1$ gebildet und zurückgegeben.
- Bei einer geraden Anzahl wird das Element am Index k zurückgegeben

Für das Sortieren des Arrays können interne Bibliotheken oder eigene Sortieralgorithmen implementiert werden. Falls eine eigene Implementation eines Algorithmus benutzt wird, lager diesen in eine eigene Methode aus.

Auch hier soll das Ergebnis vor dem Zurückgeben auf 2 Nachkommastellen gerundet werden.

Tipp: die Studenten an sich sind für uns, vor allem für das Sortieren das Arrays in Teilaufgabe 2, gar nicht mehr interessant. Nur die Notendurchschnitte der einzelnen Studenten einer Hochschule haben für den Median Aussagekraft.
Um das Implementieren dieser Methode also einfacher zu gestalten, könnte aus den Notendurchschnitten der Studenten ein eigenes Array erstellt werden und dieses dann (entweder durch eine eigene Methode oder) mithilfe einer Bibliotheksfunktion sortiert werden.

Musterlösung.

Aufgabe 4

Abschließend zu diesem Aufgabenblatt sollen Hochschulen nun miteinander verglichen werden können. Bei einem Vergleich zweier Hochschulen sollen intern das arithmetische Mittel der Notendurchschnitte der Studenten herangezogen werden. Zwei Hochschulen sind also gleich, wenn der Durchschnitt aller Notendurchschnitte aller Studenten gleich sind.

Hierzu soll der Operator „==“ überladen werden.

Folgender Aufruf soll nach der Implementierung richtig funktionieren:

```
Hochschule h1 = new Hochschule(…)   // arith. Mittel d. Studenten sei 2,6
Hochschule h2 = new Hochschule(…)   // auch hier: arith. Mittel: 2,6
Hochschule h3 = new Hochschule(…)   // arith. Mittel d. Studenten sei 3,1

if (h1 == h2)
{
    Console.WriteLine(“h1 und h2 sind gleich“);
}

if (h1 == h3)
{
    Console.WriteLine(“h1 und h3 sind gleich“);
}

if (h1 != h3)
{
    Console.WriteLine(“h1 und h3 sind nicht gleich“);
}
```

Ausgabe dieses Codes-Stückes:

```
h1 und h2 sind gleich
h1 und h3 sind nicht gleich
```

Musterlösung.

Bei weiteren Fragen zu diesen Aufgaben, welche nicht durch die Musterlösungen klar geworden sind, bitte einfach melden (oder im Forum auf Moodle nachfragen).

TECHNISCHE HOCHSCHULE NÜRNBERG
GEORG SIMON OHM