3 Objektorientiertes Programmieren

Aufgabe 1 Bestehendes erweitern

Schreibe eine Klasse *Zufall* die von der .NET Klasse *Random* erbt und „bessere“ (andere) Zufallszahlen erzeugt. Implementiere eine neue Funktion *Next()* mit Rückgabetyp **int** und eine Möglichkeit auf die ansonsten in der Oberklasse versteckte Methode *Sample()* zuzugreifen.

Aufgabe 2 Wer erbt hier von wem?

Mit dieser Aufgabe soll Grundsätzliches zur Vererbung eingeübt werden. Zunächst sollte eine abstrakte Klasse *Fahrzeug* erstellt werden, welche zwei (abstrakte) Funktionen *void Fahren()* und *int Raeder()* besitzen soll. Von dieser Klasse soll eine Klasse *Auto* erben, welche die beiden Funktionen implementiert. Die Klasse *Auto* soll außerdem noch die Eigenschaften *int Gaenge* und *bool Turbolader* besitzen. Nun soll eine dritte Klasse erstellt werden, welche von *Auto* erbt und *Porsche* heißt. Hier wird die Funktion *void Fahren()* überschrieben. Das Programm soll nun Instanzen von Porsche erstellen und die Funktion *Fahren()* ausführen.

Aufgabe 3 Wir legen eine Kopie an

Wir erweitern nun die vorherige Aufgabe indem wir in der Klasse *Auto* einen Kopierkonstruktor erstellen (alternativ eine *Clone()* Funktion). Dabei sollen die Eigenschaften der Klasse kopiert werden. Lege außerdem eine Funktion in der Klasse *Auto* an, welche zwei Instanzen der Klasse vergleicht. Versuche explizit eine *Auto*-Klasse in eine *Porsche*-Klasse zu casten. Wieso gibt es hier Probleme?

Aufgabe 4 Wie spät ist es?

Eine abstrakte Klasse *Zeit* soll eine Variable vom Typ *DateTime* (Struktur aus dem Namespace **System**, welche das aktuelle Datum und die aktuelle Zeit repräsentiert) und zwei Konstruktoren besitzen. Ein parameterloser Konstruktor dient dazu, die aktuelle Zeit des Computers zu ermitteln (nutze dazu die Eigenschaft *DateTime.Now*) und die Variable damit zu intialisieren. Dem zweiten Konstruktor kann eine beliebige Zeit übergeben werden. Die Klasse definiert außerdem die abstrakte Methode *ZeitAnzeige()*.

Leite von dieser Klasse zwei Klassen (*ZeitKurz* und *ZeitLang*) ab, welche die abstrakte Methode überschreiben. Die Methode der Klasse *ZeitKurz* soll die Zeit ohne Sekunden zurückgeben und

die der Klasse *ZeitLang* mit Angabe der Sekunden. Verwende dazu die Methoden *ToShortTimeString()* bzw. *ToLongTimeString()* des *DateTime*-Objekts. Teste beide Klassen!

Aufgabe 5 Vererbungshierarchie

In (Abb. 1) ist eine Vererbungshierarchie dargestellt, die zu implementieren ist. Die Methode *Vereinsdaten()* liefert die Werte aller Datenelemente als **string** zurück. Die Methode *Einnahmen()* liefert die Einnahmen des Vereins (Beitrag mal Anzahl der Mitglieder) als **double**-Wert zurück. Und die Methode *AnzahlAendern(int x)* ändert die Anzahl der Vereinsmitglieder (*anzahlMG*) auf den übergebenen Wert und liefert die neue Anzahl zurück.

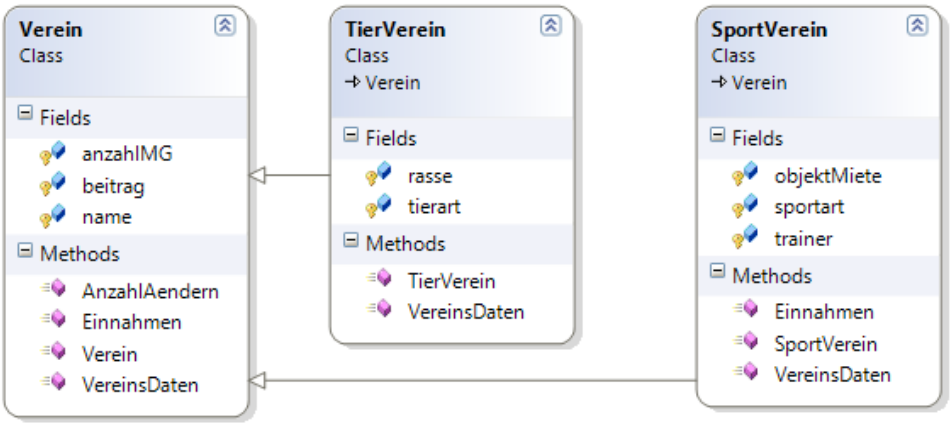

Abbildung 1: Vererbungshierarchie für Aufgabe 5

Die Methoden *VereinsDaten()* und *Einnahmen()* weichen in den abgeleiteten Klassen von den Methoden der Basisklassen ab. In der Methode *VereinsDaten()* sind zusätzlich die neuen Datenelemente der Klassen zurückzugeben. Von den Einnahmen des Sportvereins ist die Objektmiete abzuziehen. Die Methoden sind so zu überschreiben, dass sie ein polymorphes Verhalten aufweisen.

Der Quellcode soll so kurz wie möglich sein. Nutze (wenn möglich) die Methoden der Basisklasse! Erstelle anschließend eine Klasse mit einer *Main()*-Methode und einer Methode *Ausgaben()*. Erzeuge in der Einstiegsfunktion je ein Objekt der Klasse *SportVerein* und *TierVerein*. Rufe die Methode *Ausgaben()* zweimal auf und übergebe ihr jeweils eines der beiden *Verein*-Objekte. In der Methode *Ausgaben()* soll jede der drei Methoden für das übergebene Objekt einmal aufgerufen und die Rückgabewerte auf der Konsole ausgegeben werden.