Versuchen Sie immer, Ihren Code zu kommentieren!

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

Aufgabe 1

Entwickeln Sie ein Interface namens IFliegen, dass eine Methode Fliegen() deklariert. Diese Methode soll keinen Rückgabewert besitzen und keine Parameter erwarten. Entwickeln Sie dann eine Klasse namens Biene, die das Interface IFliegen implementiert. Geben Sie in der zu implementierenden Methode „Summsumm“ auf der Standardausgabe aus. Entwickeln Sie dann eine Klasse namens Fledermaus, die ebenfalls das Interface IFliegen implementiert. Geben Sie in der zu implementierenden Methode „Flatterflatter“ auf der Standardausgabe aus.

Entwickeln Sie ein Interface namens ISingen, dass eine Methode Singen() deklariert. Diese Methode soll keinen Rückgabewert besitzen und keine Parameter erwarten. Entwickeln Sie dann eine Klasse namens Singvogel , welche das Interface IFliegen und ISingen implementiert. Geben Sie in den zu implementierenden Methoden „Flapflap“ und „Zwitscherzwitscher“ auf der Standardausgabe aus.

Legen Sie jeweils eine Biene, eine Fledermaus und einen Singvogel an und testen Sie Ihre Methoden.

Aufgabe 2

Menschen sind an wichtigen Ereignissen interessiert. Um Objekten Nachrichten aus verschiedenen Quellen zukommen zu lassen, können diese das Interface `INachrichtenEmpfänger` verwenden:

```
interface INachrichtenEmpfänger
{
	// Übergabe einer neuen Nachricht
	void EmpfangeNachricht(string nachricht);
}
```

Eine empfangene Nachricht wird vom Nachrichten-Empfänger im einfachsten Fall auf dem Bildschirm ausgegeben.

Nachrichten können wiederum von verschiedenen Quellen erzeugt werden, z.B. Webseiten, TV, Radio oder Zeitungen. Die Fähigkeit, Nachrichten zu versenden, lässt sich somit auch in ein Interface `INachrichtenQuelle` abstrahieren:

```
interface INachrichtenQuelle
{
	// Interessierte können sich bei der Quelle anmelden
	// (falls sie noch nicht angemeldet sind)
	void Anmelden(INachrichtenEmpfänger empfänger);

	// Angemeldete können sich bei der Quelle wieder abmelden
	// (falls sie angemeldet sind)
	void Abmelden(INachrichtenEmpfänger empfänger);
```

```
        // neue Nachricht wird an alle angemeldeten Empfänger übergeben
        // (Aufruf deren Methode EmpfangeNachricht)
        void SendeNachricht(string nachricht);
}
```

Etwas umständlich an dieser Struktur ist, dass sich ein Nachrichten-Empfänger bei jeder Nachrichten-Quelle anmelden muss, von der er neue Nachrichten zugeschickt bekommen möchte.
Günstiger wäre ein Vermittler, der sich bei mehreren Nachrichten-Quellen anmeldet und alle Nachrichten, die er von den Quellen bekommt, direkt an Nachrichten-Empfänger weiterleitet, die sich bei ihm angemeldet haben. Ein Vermittler ist damit sowohl Nachrichten-Empfänger als auch Nachrichten-Quelle.

Aufgabe:
Legen Sie mindestens ein Objekt von einer Klasse Person an, welches das Interface `INachrichtenEmpfänger` implementiert.

Legen Sie verschiede Klassen für Nachrichten-Quellen an, die das Interface `INachrichtenQuelle` implementieren. Legen Sie von jeder Klasse mindestens ein Objekt an.

Definieren Sie anschließend eine Klasse `Vermittler`, die die beiden Interfaces `INachrichtenEmpfänger` und `INachrichtenQuelle` implementiert.

Testen Sie Ihren Code indem Sie mehrere Quellen bei dem Vermittler anmelden, eine oder mehrere Personen bei dem Vermittler anmelden und bei den Quellen Nachrichten erzeugen.

Aufgabe 3

Schreiben Sie drei Schnittstellen für die Verwendung von Temperaturwerten in Celsius, Fahrenheit und Kelvin. Jedes Interface besitzt ein Property für den Temperaturwert (z.B. `double Celsius {get; set;}`) und eine Methode `Ausgabe()` welche die Temperatur auf der Konsole (mit der richtigen Maßeinheit) ausgibt.

Die Getter sollen den Temperaturwert jeweils in Celsius, Fahrenheit und Kelvin zurückliefern (z.B. 1 Grad Fahrenheit = (Grad Celsius * 9) / 5 + 32).

Die Setter sollen den Temperaturwert immer in Grad Celsius speichern (z.B. 1 Grad Celsius = (Grad Fahrenheit - 32) * 5 / 9)

Hinweis Kelvin: 1 Kelvin = Grad Celsius + 273,15.

Schreiben Sie eine Klasse Temperatur, welche alle drei Schnittstellen implementiert und ein Attribut `temperatur` besitzt. In dem Attribut `temperatur` wird immer ein Celsiuswert gespeichert. Testen Sie Properties und die Ausgabe-Methoden. Testen Sie Ihre Implementierungen in der Main-Methode.

Zur Kontrolle: 10° Celsius sind 50° Fahrenheit und 283,15 Kelvin.

Aufgabe 4

Nicht weit von hier fließt der Fluss Ruhr, der einmal der Industrieregion Ruhrgebiet den Namen gegeben hat, durch eine idyllische Landschaft. In den Ruhrwiesen weiden heute Wasserbüffel.

Im Verlauf des Jahres steigt und fällt der Wasserstand des Flusses. Beim Normalstand haben die Büffel ideale Bedingungen, fällt der Wasserstand auf 0,3 m oder weniger, wird es für die Tiere zu trocken und sie müssen zusätzlich getränkt werden. Steigt der Wasserstand auf eine Höhe von 5 bis 7 m, werden die Ruhrwiesen überschwemmt und die Büffel sollten in den Stall gebracht werden. Bei einer Wasserhöhe ab 7 m müssten die Büffel im Fluss schwimmen. Dann wird es allerhöchste Zeit, sie vom Fluss zu entfernen.

Am Flussufer führt ein Rad- und Wanderweg vorbei. Besonders die Mitglieder des Wandervereins nutzen diesen Weg. Fällt der Wasserstand auf einen Meter oder weniger, ist es möglich, den Fluss zu durchqueren. Ab 6 m Wasserhöhe ist der Wanderweg überschwemmt.

Auf einem Damm durchquert eine Straße die Ruhrwiesen. Ab einem Wasserstand von 9 m ist die Straße überschwemmt und die Straßenwacht muss die Straße sperren. Für das Aufheben der Sperre ist dann eine andere Behörde zuständig, d.h. sie führt nichts durch wenn der Spiegel wieder sinkt.

Aufgabe:
Schreiben Sie ein Programm, das den Wasserstand zufällig innerhalb sinnvoller Grenzen steigen und fallen lässt. Drei Beobachter sollen den Wasserstand beobachten:

- Beobachter Nr.1 passt auf die Wasserbüffel auf
- Beobachter Nr.2 kontrolliert den Wanderweg für den Wanderverein
- Beobachter Nr.3 hat die Straße für die Straßenwacht im Auge

Bei Handlungsbedarf schreiben diese Beobachter eine passende Meldung auf die Konsole.

Aufgabe 5

Erstellen Sie die drei Klassen `Ninja`, `Schwert` und `Shuriken`. Die Klassen `Schwert` und `Shuriken` haben jeweils eine Methode `GegnerTreffen()`, welcher ein String (z.B. `gegner`) übergeben wird. Die Methode `GegnerTreffen()` gibt auf der Konsole z.B. den Text "Zertrennte <gegner> genau in der Mitte" oder "Traf <gegner> mit voller Wucht" mit dem übergebenem String aus.

Die Klasse `Ninja` hat als Attribut genau eine Waffe und eine Methode `Angreifen()`, welcher das Ziel als String (z.B. Gegner) übergeben bekommt. `Angreifen()`ruft die Methode `GegnerTreffen()` der jeweiligen Waffe auf.

Entwerfen Sie die drei Klassen so, dass mehrere Ninja Objekte mit verschiedenen Waffen, ohne Anpassung der Klassen, angelegt werden können.

Beispiel:

Aufruf:

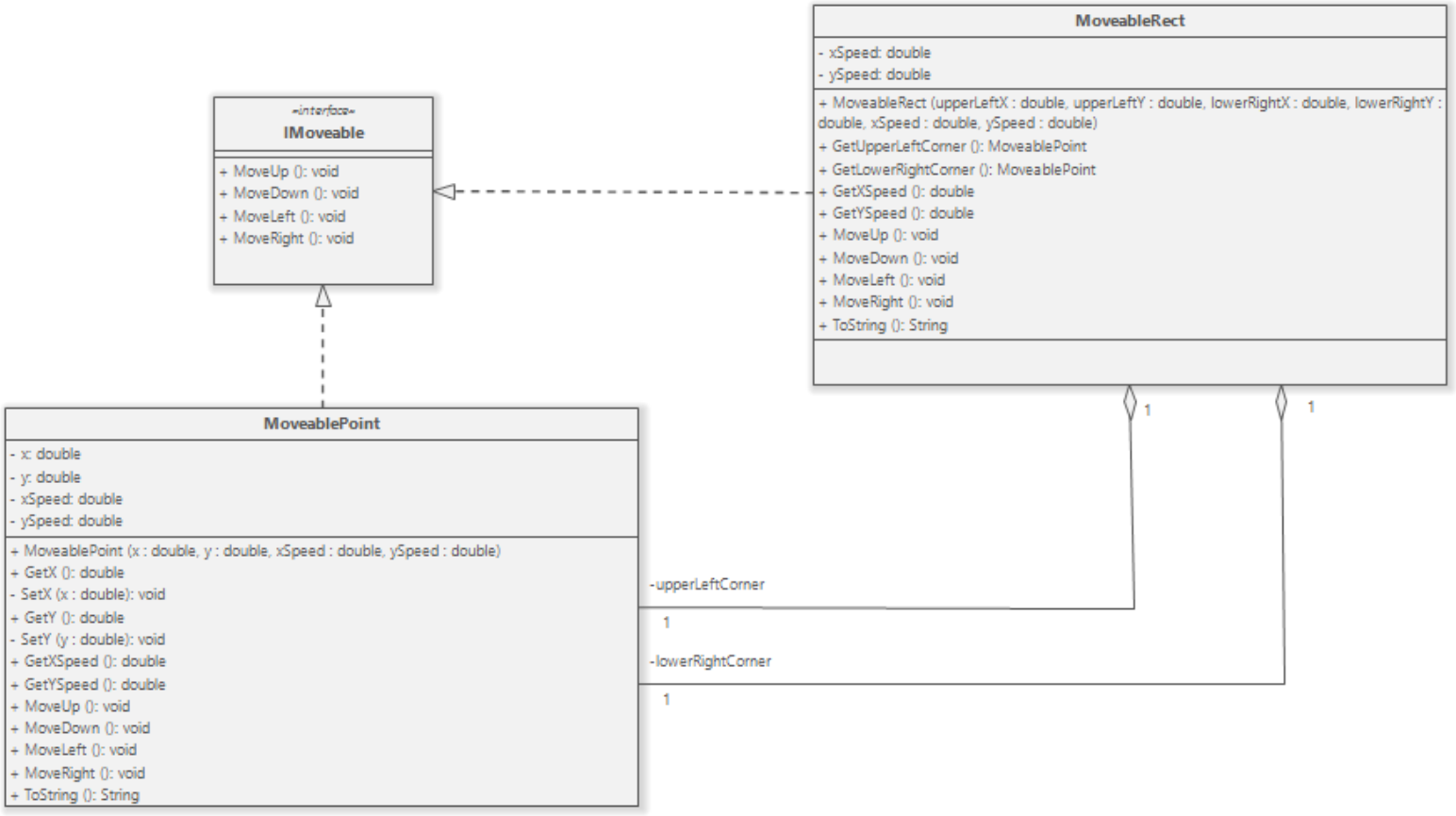
```
ninja1.Angreifen("den Bösewicht"); // Hat ein Schwert als Waffe
ninja2.Angreifen("den Gegner");    // Hat einen Shuriken als Waffe
```

Ausgabe:

```
Zertrennte den Bösewicht genau in der Mitte
Traf den Gegner mit voller Wucht
```

Aufgabe 6

Es ist das folgende UML-Klassendiagramm gegeben:



Sie sollen ein Programm schreiben, welches diesen Klassenentwurf umsetzt. In Ihrem Hauptprogramm (`Main()`-Methode) erstellen Sie dann ein Objekt des Typs `MovableRect` mit der linken oberen Ecke an den Koordinaten X = 1, Y = 10 und der rechten unteren Ecke an den Koordinaten X = 6, Y = 5. Das Rechteck soll sich mit einer Geschwindigkeit von drei Einheiten nach rechts bzw. links und zwei Einheiten nach oben bzw. unten bei jedem Schritt in die entsprechende Richtung bewegen.

An welchen Koordinaten befindet sich das Rechteck, wenn es folgende Bewegungen macht:

1. drei Schritten nach rechts
2. drei Schritten nach oben
3. zwei Schritte nach links

Geben Sie für jeden Zwischenschritt und die endgültige Position die aktuellen Koordinaten des Rechtecks aus!