

*„If you think this is weird, just
look at yourselves.“
- Charles Mingus*

9 Klassen

9.1 Konstruktoren und Destruktoren

Kurz zusammengefasst kann man folgende Eigenschaften von Konstruktoren feststellen:

- Der Name der Sub ist immer New.
- Der Konstruktor wird vom System automatisch aufgerufen, wenn eine Instanz der Klasse erstellt wird.
- Ein Konstruktor gibt nie einen Wert zurück.
- Es kann einen oder mehrere Konstruktoren geben. Sollte es mehrere geben, so müssen Sie sich in der Anzahl und dem Typ der Parameter eindeutig unterscheiden. Sollte kein Konstruktor definiert sein, so existiert ein Default-Konstruktor ohne Parameter.
- Jeder Konstruktor Aufruf impliziert einen Aufruf eines Basisklassenkonstruktors (z.B. den Konstruktor von object). Mit Hilfe des Aufrufs von base kann man wählen, welcher Konstruktor der Basisklasse aufgerufen werden soll.

Bei Destruktoren gibt es auch einige einfache Regeln:

- Der Name der Sub ist immer Finalize.
- Der Destruktor wird automatisch aufgerufen, wenn eine Instanz der Klasse zerstört wird. Er kann ausschließlich aus der Klasse selbst oder ihren Derivaten aufgerufen werden.
- Er besitzt keinerlei Parameter.
- Es gibt keine überladenen Destruktormethoden. Es muss immer eindeutig sein, welche Methode als Destruktor aufgerufen werden soll.
- Es gibt eine alternative Destruktormethode, die händisch aufgerufen werden kann, die Sub Dispose. Dieser Destruktor implementiert immer das Interface I-Disposable.Dispose.

159) Erstellen Sie eine Klasse mit dem Namen „Beispielklasse“. Im Konstruktor soll der Text „Konstruktor der Beispielklasse“ auf der Konsole ausgegeben werden. Erstellen Sie eine Instanz dieser Klasse in der Methode Main.

160) Erweitern Sie die Beispielklasse um einen Destruktor. Es soll dabei der Text „Destruktor der Beispielklasse“ auf der Konsole ausgegeben werden.

161) Schreiben Sie zum Testen der Beispielklasse folgende Main – Methode. Wie wird die Ausgabe aussehen, wenn man das Programm einmal startet?

```
Dim k1 As Beispielklasse = new Beispielklasse()
If True Then
    Dim k2 As Beispielklasse = new Beispielklasse()
End If
```

162) Ändern Sie das Beispiel so ab, dass die Beispielklasse in einer eigenen Datei aber noch im selben Namespace steht.

- 163) Ändern Sie das Beispiel so ab, dass die Beispielklasse im Namespace „Beispielnamespace“ steht.
- 164) Welche der folgenden Ausdrücke sind Modifikatoren?
- Friend
 - Public
 - Shadows
 - MustOverride
 - Foreign
 - Secure
 - Private

- 165) Warum ist der folgende Quelltext fehlerhaft?

```
Public Shadows Beispiel
    Private i As Integer = 10
    Public Sub New()
        i = 5;
    End Sub
Beispiel.i = 1
```

- 166) Warum ist der folgende Quelltext fehlerhaft?

```
Friend Static Class Beispiel
    Private i As Integer = 10
    Public Shared j As Integer = 1
    Public Sub New()
        i = 5
    End Sub
End Class
Dim bsp As Beispiel = new Beispiel()
bsp.i = 3
bsp.j = 2
Beispiel.j = -1
```

- 167) Erstellen Sie ein grafisches Diagramm von folgendem Quelltext:

```
Public MustInherit Class A
End Class
```

```
Friend Class B
    Inherits A
End Class
```

```
Public Interface C
End Interface
```

```
Public Interface D
End Interface
```

```
Public Interface E
    Inherits C, D
End Interface
```

```
Friend Class F
    Inherits B
End Class
```

```
Public Interface G
End Interface
```

```
Friend Class H
    Implements E
    Implements G
End Class
```

9.2 Felder

Da ja Felder prinzipiell nichts anderes sind als Variablen, müssen wir hier eigentlich nur noch die weiterführenden Konzepte wie etwa Modifikatoren oder die Gültigkeitsdauer von Feldern üben. Wichtig sind für uns in diesem Zusammenhang folgende Modifikatoren:

- Public
- Protected
- Friend
- Protected Friend
- Private
- Static
- Shadows
- ReadOnly
- WithEvents


- 168) Erklären Sie den Modifikator Shared. Begründen Sie dabei, warum die Main – Methode in unseren Beispielen Shared ist.
- 169) Schreiben Sie eine Klasse „KomplexeZahl“, die zwei Felder vom Typ double mit Namen „real“ und „imaginaer“ besitzt. Diese beiden Felder sollen nicht von ausserhalb der Klasse erreichbar sein.
- 170) Schreiben Sie eine Klasse „Vektor2D“ mit zwei Feldern vom Typ double mit Namen „x“ und „y“. Beide Felder sollen zwar nicht von ausserhalb der Klasse erreichbar sein, aber in eventuell davon vererbten Klassen müssen Sie editierbar sein.
- 171) Schreiben Sie eine Klasse „Farbpalette“ mit drei Feldern vom Typ Integer mit Namen „rot“, „gruen“ und „blau“. Diese sollen zwar von ausserhalb der Klasse gelesen aber nicht geschrieben werden.
- 172) Erstellen Sie eine Klasse „Kreis“ mit einem double Feld mit Namen „Radius“. Dieses Feld soll in dieser Klasse und in allen vererbten Klassen verwendbar sein. Schreiben Sie zusätzlich zwei Konstruktoren: einen Konstruktor, der den Radius mit dem Wert 0.0 initialisiert und einen Konstruktor, der den Radius mit einem Parameter vom Typ double initialisiert.
- 173) Nennen Sie ein sinnvolles Beispiel für den Modifikator Friend.
- 174) Erstellen Sie eine Klasse „Mathematik“ mit einer Konstante mit dem Namen „Pi“ und dem Wert 3,1415. Schreiben Sie in Main den nötigen Quelltext um den Wert von Pi auf der Konsole auszugeben.



9.3 Methoden

Methoden bilden einen elementaren Baustein der Programmierung. Sie ermöglichen uns die Kapselung und die Wiederverwendung von Quelltexten. Dank Parametern ist eine hohe Flexibilität beim Aufruf von Methoden möglich.

In diesem Kapitel wollen wir uns der Deklaration und Definition von Methoden widmen. Dies betrifft vor allem Modifikatoren und den Einsatz von Parametern. Zwei Spezialfälle einer Methode, nämlich den Konstruktor und den Destruktor haben wir ja schon kennengelernt.

- 175) Erstellen Sie die Klasse „Beispielklasse“ mit einer Methode „Beispielmethode“, welche den Text „Aufruf der Beispielmethode“ auf der Konsole ausgibt. Diese soll keinen Wert zurückgeben und öffentlich aufrufbar sein. Erstellen Sie in der Main – Methode eine Instanz dieser Klasse und rufen Sie die Funktion auf.
- 176) Erweitern Sie die Klasse „KomplexeZahl“ aus dem letzten Kapitel so, dass die beiden Felder über den Konstruktor mit zwei Werten initialisiert werden.
- 177) Schreiben Sie folgende drei Methoden für die Klasse „KomplexeZahl“: Addieren, Subtrahieren und Multiplizieren. Keine dieser Methoden hat einen Rückgabebetyp, jedoch bekommen sie einen Parameter vom Typ „KomplexeZahl“. Dieser Parameter soll zur Berechnung verwendet werden.
- 178) Schreiben Sie einen Konstruktor, der ein Objekt vom Typ „KomplexeZahl“ übergeben bekommt und mit dessen Werten die aktuelle Klasse initialisiert.
- 179) Schreiben Sie für die drei Methoden aus der vorletzten Übung eine überladene Version, die als Funktion implementiert ist und die zwei Parameter vom Typen „KomplexeZahl“ annimmt und den Rückgabewert „KomplexeZahl“ besitzt. Verwenden Sie zur Berechnung die bestehenden Methoden.
- 180) Schreiben Sie für die Klasse „KomplexeZahl“ eine Methode „Ausgabe“, die die komplexe Zahl in der Form „(real, imaginär i)“, also z.B.: „(1,2i)“ ausgibt.
- 181) Überschreiben Sie die Methode ToString() der Klasse „KomplexeZahl“ so, dass sie die Methode „Ausgabe“ aufruft.
- 182) Ändern Sie die Methoden aus der Übung 179 so, dass sie nur innerhalb der Klasse aufgerufen werden können.
- 183) Erstellen Sie eine Klasse „Beispielklasse“ mit einer Methode „swap“, die zwei ganze Zahlen als Referenzparameter übergeben bekommt. Diese Methode soll die Werte der beiden Parameter austauschen.
-  184) Erstellen Sie noch einmal die Methode „swap“, benutzen Sie diesmal eine andere Implementation als beim letzten mal.
- 185) Fügen Sie zur „Beispielklasse“ eine Methode „Zufallszahl“ hinzu, die einen Parameter vom Typ out double besitzt und in diesem Parameter Zufallszahlen zurückgibt. Schreiben Sie einen entsprechenden Quelltext um diese Methode zu testen.

9.4 Eigenschaften

Eigenschaften sind Sonderformen von Feldern. Für den Benutzer der Klasse sehen Sie aus wie gewöhnliche Felder mit einem public Modifikator. Tatsächlich verbirgt sich aber hinter einer Eigenschaft oft aufwendiger Quelltext um die Gültigkeit, Formatierung und

ähnliches zu überprüfen. In vielen Fällen kann es auch vorkommen, dass sich eine Eigenschaft im Hintergrund aus mehreren Feldern und/oder dem Ergebnis einer Berechnung zusammensetzt.

Beispielsweise könnte in der Klasse „Vektor2D“ eine Eigenschaft „Länge“ existieren, die immer aus den aktuellen Werten neu berechnet wird und daher kein Feld benötigt, in dem sie gespeichert wäre. Wie an diesem Beispiel gut zu erkennen ist, können Eigenschaften gelesen und geschrieben oder entweder gelesen oder geschrieben werden. Im allgemeinen findet man sehr häufig Eigenschaften die ausschließlich gelesen werden können.

- 186) Erstellen Sie eine Klasse „Beispielklasse“ und eine Eigenschaft „Beispieleigenschaft“ vom Typen string. Diese Eigenschaft soll geschrieben und gelesen werden. Schreiben Sie entsprechenden Quelltext um diese Eigenschaft zu testen.
- 187) Ändern Sie die Klasse „KomplexeZahl“ so ab, dass „real“ und „imaginaer“ Eigenschaften sind.
- 188) Fügen Sie zur Klasse „Vektor2D“ die Eigenschaft „Länge“ hinzu, welche die Länge des Vektors repräsentiert. Diese Eigenschaft soll nicht beschreibbar sein.
- 189) Erstellen Sie eine Klasse „Farben“, die die nicht beschreibbaren Eigenschaften „rot“, „gruen“ und „blau“ hat sowie die beschreibbare und lesbare Eigenschaft „aktuelleFarbe“. Die Eigenschaft „aktuelleFarbe“ soll vom Typ string sein, alle anderen sollen als uint definiert sein.



9.5 Arbeiten mit Objekten

Zum Abschluss des Kapitels „Klassen“ wollen wir uns noch etwas einigen Besonderheiten und Methoden mit „Nebenwirkungen“, wie etwa dem Kopieren von Objekten, der Instanzbildung und der Verwendung von this zuwenden.

- 190) Erstellen Sie ein Array mit zehn Elementen vom Typ „KomplexeZahl“.
- 191) Erstellen Sie ein Array mit drei Elementen vom Typ „KomplexeZahl“. Diese sollen mit den Werten (1, 2i), (4, 2i) und (9, 6i) initialisiert werden.
- 192) Erklären Sie die Ausgabe des folgenden Quelltextes:



```
Public Class A
    Public i As Integer = 0
    Public Sub New(ByVal a As Integer)
        i = a
    End Sub
    Public Shared Sub Main()
        Dim a1 As A = New A(1)
        Dim a2 As A = New A(2)
        a2 = a1
        a2.i = 10
        Console.WriteLine(a1.i)
        Console.WriteLine(a2.i)
    End Sub
End Class
```