# 2 Was ist VB.NET?

VB.NET ist eine Programmiersprache basierend auf dem Microsoft .NET-Framework. Das Framework verbindet verschiedene Programmiersprachen. Programme werden zwar in den jeweiligen Programmierspachen geschrieben, diese werden aber in ein und dieselbe *Common Language Runtime* kompiliert. Dies ist geradezu ein revolutionäres Konzept, denn bis dato waren wir (die Progrmmierer) unterschiedliche Kompiler mit unterschiedlichen Ergebnissen gewohnt. So konstantierten sich Stärken und Schwächen der einzelnen Programmiersprachen und daraus resultierent schrieb man einer Programmiersprache einem bestimmten Aufgabenbereich zu. Der Aufgabenbereich von Viausl Basic (der Vorgängerversion von VB.NET) war die Datenbankprogrammierung im Officeumfeld. Hier lagen die Stärken dieser Programmierspachen.
Mit den neuen Spachen von .NET sind die Grenzen gefallen. Ein Kompiler für alle Sprachen offenbart die Leistungsfähigkeit aller angeschlossenen Programmierspachen. Nun ist es egal ob der Sourccode in C# oder VB.NET erstellt wird. Das Ergebnis ist ein zum Framework kompatibeles Programm mit identischen Leistungsmerkmalen.
Für uns bedeutet dies: wir können mit einer relativ leicht verständlichen Syntax sowie einer übersichtlichen Programmierumgebung arbeiten und leistungsstarke Programme erstellen, welche einem Vergleich zu C# nicht scheuen müssen.

## 2.1 Unterschiede zu Visual Basic 6

Diese Erläuterungen werden nur für Leser mit VisualBasic6-Erfahrung von Interesse sein. Für Einsteiger werden die aufgelisteten Neuerungen noch keine Aussagekraft haben. Natürlich wird sich das mit wachsender Erfahrung ändern und dann werden Sie sicherlich einige der Neuerungen von VB.NET als selbstverständlich ansehen.

VB.NET ist der Nachfolger von Visual Basic 6. VB.NET ist aber auch Bestandteil von Visual Studio.NET. Hier haben wir bereits einen Unterschied zu VB 6: Visual Basic 6 war nicht in das Visual Studio-Paket integriert.

Zwar wurde VB gemeinsam mit dem Visual Studio-Paket ausgeliefert, aber sie verwendeten unterschiedliche IDEs. Integrated Development Environment (IDE) beschreibt die integrierte Entwicklungsumgebung von Visual Studio.NET. Mit Visual Studio.NET wird ein C++-, C#- oder VB.NET- Programmierer mit der gleichen Entwicklungsumgebung arbeiten. Dies ist ein bedeutender Vorteil. So rücken Programmierer der unterschiedlichen Programmiersprachen enger zusammen und die Wahl, in welcher Programmiersprache das nächste Projekt verwirklicht wird, ist leichter zu treffen.

### 2.1.1 Rapid Application Development

VB.NET unterstützt die schnelle Anwendungsentwicklung (`RAD`, Rapid Application Development) mit Projektvorlagen, Designern und anderen Features in der Visual Studio.NET-Entwicklungsumgebung.

### 2.1.2 Vererbung

Ein weiterer Fortschritt ist die Vererbung. Vererbung ist das Ableiten einer Klasse von einer anderen. Abgeleitete Klassen erben die Eigenschaften, Methoden und Ereignisse der Basisklasse und können diese erweitern. Abgeleitete Klassen können auch vererbte Methoden mit neuen Implementierungen überschreiben.

### 2.1.3 Ausnahmebehandlung

Außerdem wurde eine verbesserte Fehlerhandhabung (Ausnahmebehandlung) implementiert. Visual Basic verwendet eine erweiterte Version der `Try-Catch-Finally`-Syntax, die bereits von anderen Sprachen (z.B. von C++) unterstützt wurde. Die strukturierte Ausnahmebehandlung kombiniert eine moderne Kontrollstruktur (vergleichbar mit Select…Case oder While) mit Ausnahmen, geschützten Codeblöcken und Filtern.

### 2.1.4 Überladung

Überladung ist das Erstellen von mehreren Prozeduren, Eigenschaften oder Instanzkonstruktoren in einer Klasse, die zwar denselben Namen verwenden, aber über andere Argumenttypen verfügen. Overloads sind am besten an einem einfachen Beispiel zu erklären. Nehmen wir an, dass Sie eine Klasse nutzen, welche einmal einen String und einmal eine Zahl ausgeben

soll. Ohne `Overload` würden Sie zwei Prozeduren mit zwei unterschiedlichen Namen verwenden (`DisplayChar` und `DisplayInt`).

**Ohne Overload**

```
1  Sub DisplayChar(ByVal theChar As Char)
2  ' Add code that displays Char data.
3  End Sub

4  Sub DisplayInt(ByVal theInteger As Integer)
5  ' Add code that displays Integer data.
6  End Sub
```

Mit Overload benötigen wir ebenfalls zwei Prozeduren, jedoch teilen sich beide denselben Namen `Display`. Dies ist eine wesentliche Erleichterung. Schließlich wollen wir mit dem Aufruf der Prozedur das gleiche Ziel erreichen. Ob dieses Ziel mit `DisplayChar`, `DisplayInt` oder einfach von `Display` erreicht wird, ist dabei nebensächlich.

**Mit Overload**

```
1  Overloads Sub Display(ByVal theChar As Char)
2    ' Add code that displays Char data.
3  End Sub

4  Overloads Sub Display(ByVal theInteger As Integer)
5    ' Add code that displays Integer data.
6  End Sub
```

Jenach dem, welches Argument übergeben wird, wird die entsprechende Prozedur aufgerufen. Hierbei ist es nicht mehr von Bedeutung, ob das Argument einen `Integer`-Wert oder einen `Char`-Wert beinhaltet. Diese Entscheidung wird im Hintergrund für den Anwender unsichtbar getroffen.

### 2.1.5 Overrides: Überschreiben von Methoden und Eigenschaften

Abgeleitete Klassen erben Eigenschaften und Methoden, die in ihrer Basisklasse definiert sind. Dies ist praktisch, da Sie diese Elemente wiederverwenden können, sobald sie für die von Ihnen verwendete Klasse erforderlich sind. Wenn der vererbte Member nicht in seiner ursprünglichen Form verwendet werden kann, können Sie mit dem Overrides-

Schlüsselwort eine neue Implementierung definieren, vorausgesetzt, dass die Eigenschaft oder Methode in der Basisklasse durch das Overridable-Schlüsselwort gekennzeichnet ist.

### 2.1.6 Konstruktoren und Destruktoren

Mit Konstruktoren und Destruktoren wird das Erstellen oder Zerstören von Klassen gesteuert. Hierzu werden Prozeduren `Sub New` und `Sub Finalize` verwendet. Insbesondere mit Blick auf die Systemressourcen sollten nicht mehr verwendete Klassen freigegeben werden. Freie Objekte werden von VB.NET mit `Nothing` bezeichnet.

### 2.1.7 Datentypen

In VB.NET werden drei neue Datentypen eingeführt. Der Char-Datentyp ist eine vorzeichenlose 16-Bit-Größe, mit der Unicode-Zeichen gespeichert werden. Er entspricht dem System.Char-Datentyp von .NET-Framework. Der Short-Datentyp, eine 16-Bit-Ganzzahl mit Vorzeichen, wurde in früheren Versionen von Visual Basic als Integer bezeichnet. Der Decimal-Datentyp ist eine 96-Bit-Ganzzahl mit Vorzeichen, die mit einer variablen Potenz zur Basiszahl 10 skaliert wird. In früheren Versionen von Visual Basic war dieser Typ nur in einer Variante verfügbar.

Für alle Visual Basic 6-Umsteiger haben wir eine tabellarische Gegenüberstellung der Datentypen erstellt.

**Tabelle 1.** Datentypen

| Visual Basic 6 | VB.NET |
|---|---|
| Boolean | Boolean |
| Byte | Byte |
| Date | Date |
| Single | Single |
| Integer (2 Byte) | Short (2 Byte) |
| Long (4 Byte) | Integer (4 Byte) |
| | Long (8 Byte) |
| Currency (8 Byte) | Decimal (12 Byte) |

| | |
|---|---|
| Object | Object |
| Variant | Object |
| String | String |
| | Char (Unicode Character) |

### 2.1.8 Verweise

Mit Hilfe von Verweisen können Sie in anderen Assemblies definierte Objekte verwenden. In VB.NET zeigen Verweise auf Assemblies statt auf Typbibliotheken.

### 2.1.9 Namespaces

Namespaces verhindern Namenskonflikte, indem Klassen, Schnittstellen und Methoden in Hierarchien organisiert werden.

### 2.1.10 Assemblies

Assemblies ersetzen die Typbibliotheken und erweitern ihre Funktionalität, da alle erforderlichen Dateien für eine bestimmte Komponente oder Anwendung beschrieben werden. Eine Assembly kann einen oder mehrere Namespaces enthalten.

### 2.1.11 Multithreading

In VB.NET können Anwendungen programmiert werden, die mehrere Aufgaben unabhängig voneinander ausführen. Eine Aufgabe, durch die andere Aufgaben möglicherweise verzögert werden, kann in einem separaten Thread ausgeführt werden. Dieser Vorgang wird als Multithreading bezeichnet. Multithreading ermöglicht eine schnelle Reaktion von Anwendungen auf Benutzereingaben, da komplexe Aufgaben in von der Benutzeroberfläche getrennten Threads ausgeführt werden.

### 2.1.12 Common Language Runtime (CLR)

Die Laufzeitumgebung des .NET-Frameworks heißt Common Language Runtime oder kurz CLR. Sie ist auf den Betriebssystemdiensten aufgebaut und legt fest, wie .NET-Programme ausgeführt und unterstützt werden.

Zur Unterstützung gehören u.a. die Garbage Collection, Sicherheit und JIT-Compilation. Die CLR basiert auf einem eigenen Befehlssatz der CIL – Common Intermediate Language. Der Sorcecode einer .NET-Programmiersprache wird in die CIL übersetzt. Diese Übersetzung ist aber nicht lauffähig. Hierzu bedarf es einer weiteren Übersetzung (just-in-time) des Frameworks. Das Framework ist auf die jeweilige Plattform (System) angepasst. Hierdurch ist es möglich, dass ein und derselbe Sourcecode auf unterschiedlichen Systemen funktioniert. Diese Technik ist bereits aus anderen Programmiersprachen wie Java bekannt. Sie ermöglicht die Implementierung von Sourcecodes einer anderen .NET-Programmiersprache.

Ein weiteres Merkmal der CLR sind Sicherheitsmerkmale, wie der Garbage Collector. Er ist für die Speicherverwaltung eines Programms zuständig. Hierdurch sollen .NET-Programmiersprachen sicherer und robuster werden. Eine manuelle Speicherverwaltung wie bei C++ oder C entfällt weitgehend. So wird z. B. der Speicher für ungebundene Objekte autonom freigegeben.

Zusammenfassend können wir sagen, dass die CLR eine gemeinsame Plattform für alle .NET-Programme darstellt. Hierbei ist es nebensächlich, auf welcher Zielmaschine das Programm ausgeführt werden soll.

### 2.1.13 Garbage Collection

Der Garbage Collector (GC) ist für die Speicherverwaltung der Programme verantwortlich. Sie als Programmierer müssen sich nicht mehr um die Speicherbereinigung kümmern, sondern überlassen dies dem GC. Dynamisch erzeugte Objekte können in Lebenszyklen mit dem folgendem Ablauf eingeteilt werden:

1. Speicher reservieren (Allokation)
2. Initialisierung
3. Nutzungsdauer (mindestens eine Referenz existiert)
4. Ressource freigeben
5. Speicherplatz des Objektes kann neu belegt werden.

Wenn der Programmierer für das Speichermanagement die Verantwortung trägt, kann es vorkommen, dass reservierter Speicher nicht mehr freigegeben wird. Es entsteht ein Speicherloch. Das kommt häufiger vor, als man glaubt. Außerdem könnte der Programmierer ein Objekt löschen, obwohl es noch gebunden ist, d.h., es besteht noch eine Referenz. Unter .NET darf nur der GC den Speicher freigeben.

Speicher für Objekte von Klassen und Arrays werden auf dem Heap (managed heap) erzeugt. Der Heap ist ein zusammenhängender Speicherbereich. Er besitzt einen Zeiger, welcher auf die Stelle zeigt, an der ein neues Objekt angelegt werden kann. Wenn der Speicherbereich einmal nicht ausreicht, wird es schwierig. In diesem Fall müssen zuerst alle Threads gestoppt werden. Anschließend beginnt der GC mit der Arbeit. Er versucht, Speicherbereiche zu verschieben, zu löschen und zu optimieren, bis ausreichend Platz geschaffen wurde. Hierbei kann es zu längeren Bearbeitungszeiten kommen, welche besonders bei Echtzeitanwendungen zu spürbaren Unterbrechungen führen können.