Kapitel 3

# Die ersten C#-Programme erstellen



## In diesem Kapitel lernen Sie

- die Elemente der Benutzeroberfläche von Visual C# 2010 Express kennen
- welche Projektarten von Visual C# 2010 Express unterstützt werden
- wie Sie neue Projekte des Typs Konsolenanwendung erstellen
- den typischen Aufbau eines C#-Programms kennen
- wie Sie die .NET-Klassenbibliotheken verwenden
- was beim Erstellen eines Programms passiert

## 3.1 Die Oberfläche von Visual C# 2010 Express

Dieser Abschnitt macht Sie mit den verschiedenen Elementen bekannt, die Sie in der Benutzeroberfläche von Visual C# 2010 Express finden. Starten Sie das Programm (siehe auch Abschnitt 2.2), damit sich die sogenannte integrierte Entwicklungsumgebung (IDE, Integrated Development Environment) auf Ihrem Bildschirm präsentiert (siehe Abbildung 3.1 auf der folgenden Seite).

### Die Startseite

Den größten Teil des Fensters nimmt die Startseite in Beschlag, auf der Sie folgende Bereiche finden:

- **Zuletzt geöffnete Projekte** In diesem Bereich werden die zuletzt von Ihnen geöffneten Projekte und Projektmappen angezeigt. Mit den Links bei *Neues Projekt* bzw. *Projekt öffnen* können Sie andere Projekte öffnen bzw. ein neues Projekt erzeugen.
- **Erste Schritte** Im Bereich *Erste Schritte* finden Sie Links zu verschiedenen Hilfeinformationen. Gleichgültig, ob Sie sich Referenzinformationen zur Programmiersprache C# ansehen, in den Gewusst-wie-Artikeln stöbern oder weitere Inhalte herunterladen wollen, all diese Informationen sind nur einen Mausklick entfernt.
- **Aktuelle Nachrichten** Im Bereich *Aktuelle Nachrichten* erhalten Sie aktuelle Informationen zum Produkt Visual C# 2010 Express selbst. Wenn Updates, weitere Tools oder neue Versionen zur Verfügung stehen, werden Sie hier darüber informiert. Damit die aktuellen Nachrichten angezeigt werden, müssen Sie die Registerkarte *Aktuelle Nachrichten* öffnen und die Schaltfläche *RSS-Feed aktivieren* anklicken.

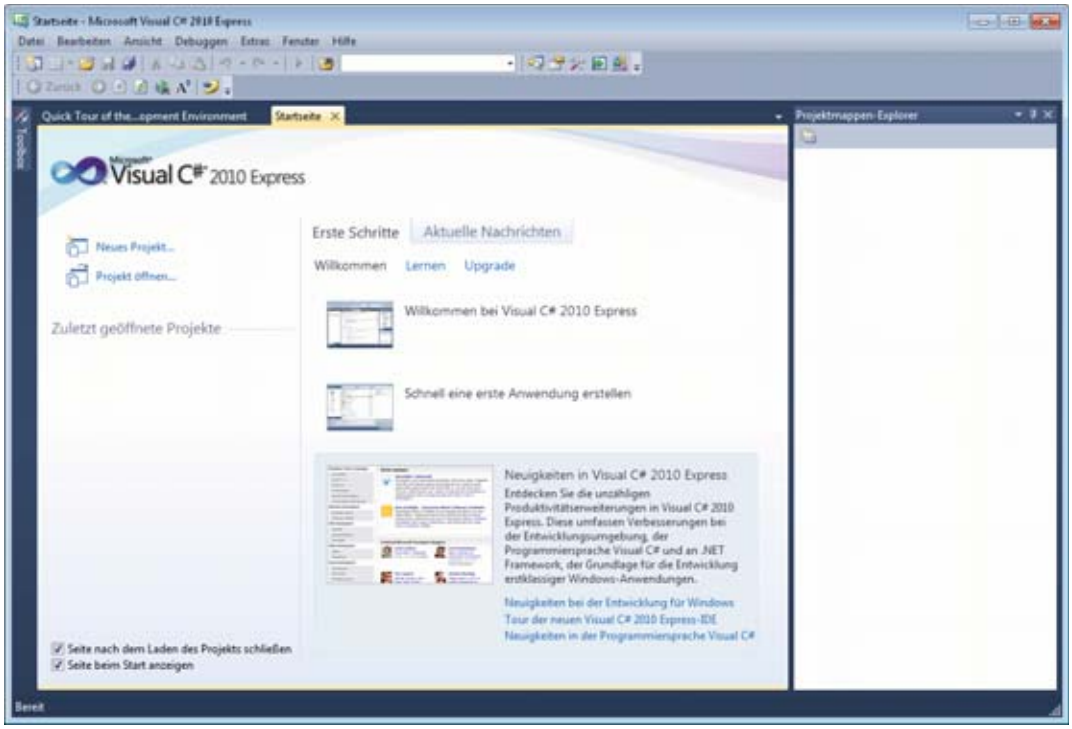



**Abbildung 3.1:** Die Benutzeroberfläche von Visual C# 2010 Express zeigt nach dem Start die Startseite an

**Tipps: Startseite anzeigen lassen, Startoption konfigurieren**

Falls Sie die Startseite einmal versehentlich geschlossen haben sollten, können Sie sie mit der Befehlsfolge *Ansicht//Startseite* wieder anzeigen lassen.

Falls Sie beim Starten von Visual C# 2010 Express die Startseite nicht sehen, sondern beispielsweise automatisch ein neues Projekt erstellen oder das zuletzt verwendete Projekt öffnen wollen, wählen Sie *Extras/Optionen* und schalten das Kontrollkästchen *Alle Einstellungen anzeigen* ein. Navigieren Sie in der Kategorienliste auf der linken Seite des Fensters zu *Umgebung/Start* und wählen Sie in der Liste *Beim Start* die Option aus, die Sie verwenden möchten.

## Wichtige Komponenten der IDE

Die IDE von Visual C# 2010 Express besteht teilweise aus Komponenten, die Sie von anderen Windows-Anwendungen bereits kennen:

- **Menüleiste** In der Menüleiste finden Sie alle Befehle, die Sie für das Erstellen, Ausführen und die Fehlersuche in Ihren Programmen benötigen und mit denen Sie weitere Fenster ein- und ausblenden können. Der Inhalt der Menüleiste und auch der Menüs selbst ist dynamisch und ändert sich, nachdem Sie beispielsweise ein Projekt geöffnet haben oder wenn Sie mit dem in der IDE integrierten Debugger (dies ist ein Werkzeug zur Fehlersuche, das Sie in Kapitel 10 ausführlich kennenlernen werden) die Haken und Ösen in Ihren Programmen erkennen und beheben können.

- **Symbolleiste** Unterhalb der Menüleiste sehen Sie nach dem ersten Starten des Programms die Standardsymbolleiste mit zahlreichen Schaltflächen (wie *Speichern, Rückgängig, Wiederholen* usw.), die Ihnen sicherlich von anderen Anwendungen her bekannt sind. Visual C# 2010 Express enthält weitere Symbolleisten, die sich automatisch einblenden, wenn Sie eine bestimmte Aufgabe erledigen (beispielsweise mit dem Tabellen-Designer eine Datenbanktabelle erstellen). Wie gewohnt können Sie andere Symbolleisten auch manuell sichtbar machen: Klicken Sie eine der sichtbaren Symbolleisten mit der rechten Maustaste an und wählen Sie den Namen der Leiste aus, die Sie ein- oder ausblenden wollen. Menüliebhaber verwenden das Untermenü *Ansicht/Symbolleisten.*
- **Statusleiste** Die Statusleiste befindet sich ganz unten am Fenster und zeigt Meldungen an, die sich auf den Vorgang beziehen, der gerade in der IDE durchgeführt wird.
- **Toolbox** An der linken Seite des Fensters sehen Sie eine Registerkarte mit der Beschriftung *Toolbox.* Die Toolbox enthält eine große Anzahl von Steuerelementen, die Sie in Ihren Windows-Anwendungen (also denjenigen, die ein Fenster besitzen) verwenden können. Hierzu gehören Textfelder, Schaltflächen, Menüs, Symbolleisten u.v.m. Wie Sie die Toolbox verwenden können und welche Schätze sich dort verbergen, erfahren Sie im vierten Teil des Buches (ab Kapitel 13).
- **Projektmappen-Explorer** Das Fenster ganz rechts in der IDE ist der Projektmappen-Explorer, der alle Dateien und Komponenten aufführt, die im aktuellen Projekt verwendet werden. Da derzeit noch kein Projekt geöffnet ist, ist der Projektmappen-Explorer noch leer. Dies wird sich im nächsten Abschnitt jedoch schlagartig ändern.



## Einfaches und erweitertes Menü

Visual C# 2010 Express enthält zwei verschiedene Versionen der Menüleiste. Nach der Installation ist standardmäßig das Menü in den Grundeinstellungen aktiviert. In dieser Variante wird nur eine Untermenge der Menübefehle und Symbolleisten angezeigt. In den Experteneinstellungen hingegen werden alle Befehle angezeigt, die Visual C# 2010 Express zur Verfügung stellt. Damit Sie die Schrittfolgen in diesem Buch nachvollziehen können, sollten Sie die Experteneinstellungen aktivieren. Öffnen Sie dazu das Menü *Optionen,* klicken Sie zuerst auf *Einstellungen* und dann auf *Experteneinstellungen.*

# 3.2 Das »klassische« erste Programm erstellen

Es ist gute Tradition, dass das erste Programm, das man in einer neuen Programmiersprache erstellt, einfach nur den Text »Hello World« auf den Bildschirm ausgibt. Wir wollen uns dieser Tradition anschließen, da der Quellcode dieses Programms sehr übersichtlich ist und er gleichzeitig sehr viele Standardelemente enthält, deren Verständnis ein gutes Fundament für das Erstellen weiterer Programme liefert.

1. Wählen Sie den Menübefehl *Datei/Neues Projekt.* Das Dialogfeld *Neues Projekt* wird angezeigt.
2. Markieren Sie in der Liste *Vorlagen* im mittleren Bereich des Dialogfeldes die Option *Konsolenanwendung.*

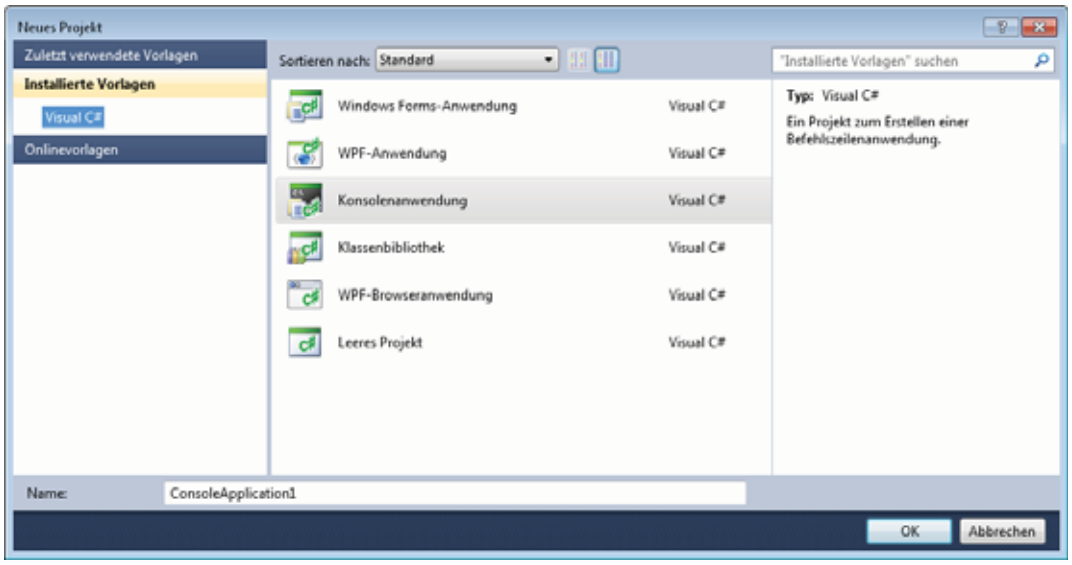



**Abbildung 3.2:** Im Dialogfeld *Neues Projekt* wählen Sie die Art des Projekts aus, das Sie erstellen wollen und legen dort auch den Namen des Projekts fest

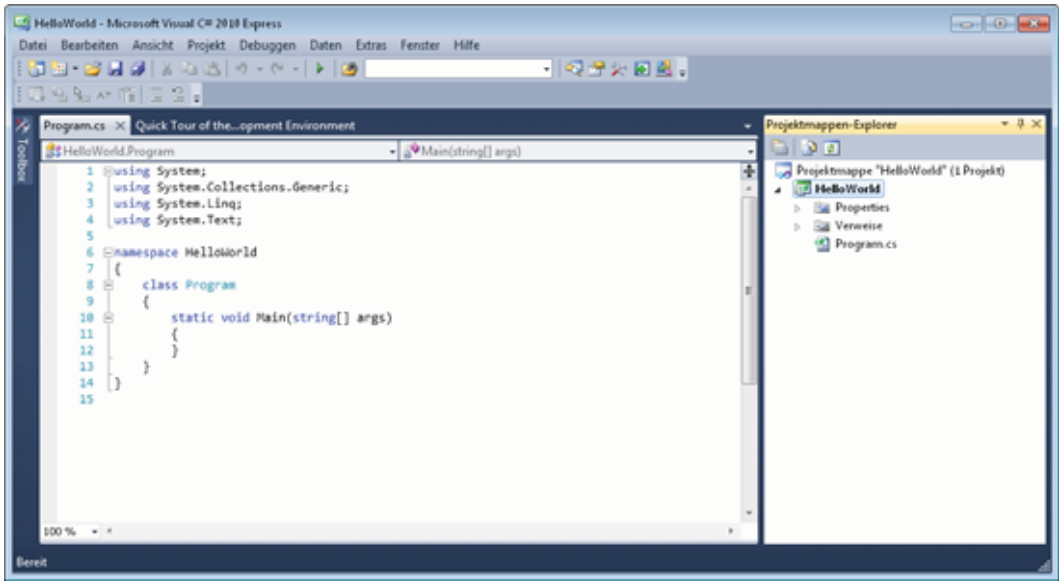
### Hinweis: Vorlagen und Projektarten

Im Dialogfeld *Neues Projekt* wählen Sie die Vorlage, und damit die Projektart, des neuen Projekts aus, das Sie erstellen wollen. Mit Visual C# 2010 Express können Sie die folgenden Projekte erstellen:

- **Windows Forms-Anwendung** Windows Forms-Anwendungen sind Programme, die in einem Fenster dargestellt ein Menü oder andere Steuerelemente besitzen, über die die Kommunikation mit den Anwendern vonstatten geht. Mit Visual C# 2010 Express lassen sich, wenn man erst einmal die Grundlagen der Programmiersprache beherrscht, auch Windows-Anwendungen schnell erstellen. Im vierten Teil dieses Buches (ab Kapitel 13) werden Sie anhand von zahlreichen Projekten lernen, wie das geht.
- **Konsolenanwendung** Konsolenanwendungen besitzen kein Fenster und keine grafische Oberfläche, sondern werden in einer Eingabeaufforderung dargestellt. Fast alle Beispiele im zweiten Teil des Buches, in dem es darum geht, Ihnen die Grundlagen der Programmiersprache C# zu vermitteln, sind Konsolenanwendungen, da diese weniger aufwendig, leichter zu verstehen und einfacher zu erstellen sind.
- **Klassenbibliothek** Bei einer Klassenbibliothek handelt es sich um eine Bibliotheksdatei, in der sich Programmcode befindet, der von mehreren Anwendungen verwendet werden kann. In Kapitel 11, in dem es vor allem um das Erstellen von Klassen in C# geht, werden Sie sehen, wie Sie eigene Klassenbibliotheken erstellen und einsetzen.
- **WPF-Anwendung** und **WPF-Browseranwendung** WPF ist die Abkürzung für Windows Presentation Foundation; hierbei handelt es sich um eine neue Präsentationsprogrammierschnittstelle für Microsoft Windows. WPF ist ein Framework zur Ausgabe von 2D-/3D-Grafiken, Video, Audio und Bildern. WPF setzt direkt auf DirectX auf und stellt so volle Hardwarebeschleunigung zur Verfügung. Das Erstellen von WPF-Anwendungen wird im Rahmen dieses Buches nicht behandelt.

3. Geben Sie in das Textfeld *Name* **HelloWorld** ein und klicken Sie auf *OK*.

   Visual C# 2010 Express erstellt das neue Projekt. Die zum Projekt gehörenden Dateien werden im Projektmappen-Explorer angezeigt. Den Quellcode des Programms sehen Sie im Text-Editor.




**Abbidung 3.3:** Die IDE hat das Gerüst der neuen Anwendung erstellt

**Hinweis: Zeilenummern im Editor anzeigen lassen**

Um wie in der obigen Abbildung zu sehen im Editor die Zeilennummern anzeigen zu lassen, wählen Sie den Befehl *Extras/Optionen*. Schalten Sie im Dialogfeld *Optionen* das Kontrollkästchen *Alle Einstellungen anzeigen* ein. Öffnen Sie dann in der Liste auf der linken Seite des Dialogfeldes den Knoten *Text-Editor/Alle Sprachen/Allgemein*. Schalten Sie schließlich das Kontrollkästchen *Zeilennummern* im Bereich *Anzeigen* ein und schließen Sie das Dialogfeld mit *OK*.

4. Setzen Sie die Einfügemarke hinter die öffnende geschweifte Klammer unterhalb von *static* (Zeile 11 in der obigen Abbildung) und drücken Sie die Eingabetaste, um eine neue Zeile zu erstellen.
5. Tippen Sie **conso** und wundern Sie sich nicht darüber, dass unter der Zeile, in der Sie tippen, ein kleines Fenster mit einer Auswahlliste aufgeht. Dies ist das IntelliSense-Fenster, das Ihre Tastatureingaben mithält und Ihnen eine Liste anbietet, aus der Sie das benötigte Schlüsselwort auswählen können. Die Markierung springt nach jedem Tastendruck immer zum ersten Eintrag, der mit den eingegebenen Buchstaben beginnt.
6. Nun sollte in der Liste *Console* markiert sein. Drücken Sie die Tabulatortaste, damit Ihre Eingabe conso im Editor durch `Console` ersetzt wird.
7. Geben Sie einen Punkt ein. Das IntelliSense-Fenster öffnet sich.
8. Geben Sie ein **w** ein, um in der IntelliSense-Liste zum ersten Eintrag zu gelangen, der mit dem Buchstaben w beginnt.

9. Drücken Sie die Taste Pfeil-Unten, bis der Eintrag `WriteLine` markiert ist. Beachten Sie, dass in einem Tooltipp-Fenster eine kurze Information zur Funktion von `WriteLine` angezeigt wird.

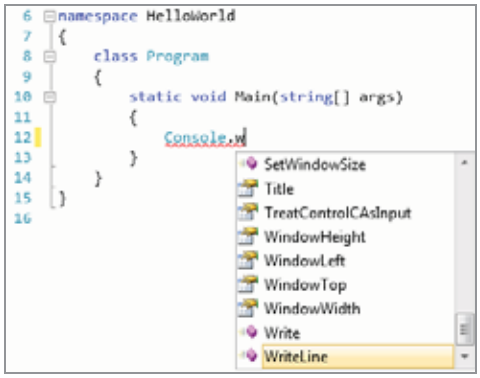



**Abbildung 3.4:** Das IntelliSense-Fenster macht Vorschläge, erspart Tipparbeit und vermeidet Tippfehler

10. Drücken Sie die Tabulatortaste, damit aus dem w im Text-Editor das *WriteLine* wird.
11. Vervollständigen Sie die angefangene Codezeile, bis diese so aussieht:

```
Console.WriteLine("Hello World");
```

Damit wäre Ihr erstes Programm fertig. Bevor wir den Quellcode genauer unter die Lupe nehmen, können Sie das Programm starten. Wählen Sie dazu *Debuggen/Starten ohne Debugging* oder drücken Sie Strg+F5. Das Programm wird erstellt (achten Sie auf die Meldung in der Statusleiste) und eine Eingabeaufforderung wird geöffnet, in der die Zeichenkette/der Text »Hello World« ausgegeben wird.


**Abbildung 3.5:** Die Konsolenanwendung »Hello World« in Aktion

Die Meldung »Drücken Sie eine beliebige Taste ...« zeigt an, dass die Ausführung der Konsolenanwendung beendet ist und das Fenster mit der Eingabeaufforderung durch das Drücken einer Taste vom Bildschirm verschwindet.

Möglicherweise ist Ihnen aufgefallen, dass das Menü *Debuggen* neben dem Befehl *Starten ohne Debugging* auch den Befehl *Debugging starten* enthält. Wenn Sie bei diesen einfachen Beispielen für Konsolenanwendungen den Befehl *Debugging starten* verwenden, wird das Programm ausgeführt, ein Konsolenfenster geöffnet und sofort wieder geschlossen. Der Befehl *Starten ohne Debuggen* hingegen sorgt dafür, dass die Meldung »Drücken Sie eine beliebige Taste« angezeigt wird und Sie die Ausgabe der Anwendung sehen können.

# 3.3 »Hello World« unter der Lupe

Auch wenn dieses kleine Programm nicht viel macht, so bildet es doch alle wichtigen Konzepte einer typischen Anwendung ab, die unter dem .NET Framework ausgeführt wird: Verwendung einer Klassenbibliothek, Erstellen eines eigenen Namensraums, Erstellen einer Klasse usw. Doch immer schön der Reihe nach. Damit Sie sich in den folgenden Erläuterungen besser zurechtfinden, hier noch einmal der komplette Quellcode des Programms, mit Zeilennummern versehen:

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5
6   namespace HelloWorld
7   {
8      class Program
9      {
10        static void Main(string[] args)
11        {
12           Console.WriteLine("Hello World");
13        }
14     }
15  }
```



## Die using-Direktiven

Die Zeilen 1 bis 4 im Quellcode werden automatisch in den Quellcode eingefügt, wenn Sie im Dialogfeld *Neues Projekt* die Vorlage *Konsolenanwendung* auswählen und Visual C# 2010 Express das Gerüst für die Quelldatei *Program.cs* erstellt. In diesen vier Zeilen wird das C#-Schlüsselwort `using` verwendet, gefolgt von den Namen der Klassenbibliotheken, die in die Quelldatei importiert werden sollen.

Bei Klassenbibliotheken handelt es sich um eine Sammlung von vielen nützlichen Funktionen, die Bestandteil von .NET Framework sind, die Sie in Ihren Programmen verwenden können, die Sie aber nicht selbst programmieren müssen. Das spart Zeit und Arbeit. Ein Beispiel für solch eine Funktion ist `WriteLine`, die im Beispiel verwendet wird, um einen Text auszugeben. Im .NET Framework sind die Klassenbibliotheken mit Namen versehen und hierarchisch organisiert. Eine Klassenbibliothek hat den Namen `System` (Zeile 1), sie enthält beispielsweise die Funktion `WriteLine`. Eine weitere hat den Namen `System.Text` (Zeile 4), dort finden Sie Funktionen, um Zeichenketten zu bearbeiten. Diese Bibliothek wird später im Buch in Kapitel 9 verwendet.

Wie oben erwähnt sind die Funktionen der Klassenbibliotheken hierarchisch organisiert und zu funktionalen Gruppen zusammengefasst. `System.Text` beispielsweise ist der Bibliothek `System` untergeordnet. Die Hierarchie wird abgebildet, indem die einzelnen Namen durch einen Punkt miteinander verbunden werden.

Diese Form der Namensgebung ist nicht nur auf die Bibliotheken selbst beschränkt; sie setzt sich bis zu den Funktionen fort, die in der Bibliothek enthalten sind. So ist `Console.WriteLine` nur eine abgekürzte Schreibweise der Funktion, deren vollständiger Name wie folgt lautet: `System.Console.WriteLine`.

Nun lässt sich erklären, welche Funktion die `using`-Anweisungen am Anfang der Quelldatei haben. In C# stehen Ihnen zwei Varianten zur Verfügung, um eine Bibliotheksfunktion zu verwenden:

- Entweder Sie verwenden den qualifizierten, vollständigen Namen der Funktion (dieser setzt sich aus dem Namen der Bibliothek, dem Namen der Klasse und dem Namen der Methode zusammen, in unserem Beispiel also `System.Console.WriteLine`)
- oder Sie verwenden am Anfang der Quelldatei die `using`-Direktive und können dann die Kurzschreibweise ohne den Namen der Bibliothek verwenden, wie es in Zeile 12 zu sehen ist.

Die `using`-Anweisung dient also dazu, den Quellcode übersichtlicher zu machen und Ihnen Tipparbeit zu ersparen, wenn Sie Funktionen der Klassenbibliothek von .NET Framework verwenden. Jede `using`-Direktive muss mit einem Semikolon abgeschlossen werden. (In der kleinen Übung im Kasten auf der folgenden Seite können Sie ausprobieren, was passiert, wenn Sie die erste `using`-Anweisung im Quellcode auskommentieren.)

Werfen Sie noch kurz einen Blick auf den Projektmappen-Explorer an der rechten Seite des Fensters von Visual C# 2010 Express. Dort sehen Sie den Ordner *Verweise,* davor ein Pluszeichen. Klicken Sie das Pluszeichen an, um den Ordner aufzuklappen. Dort sehen Sie ebenfalls den Namen der Bibliothek, nämlich *System.*

Klicken Sie nun *System* mit der rechten Maustaste an und wählen Sie im Kontextmenü den Befehl *Eigenschaften*. Im Eigenschaftenfenster, das standardmäßig unterhalb des Projektmappen-Explorers erscheint, gibt es einen Eintrag mit dem Namen *Pfad.* Wenn Sie das Eigenschaftenfenster etwas breiter ziehen, sehen Sie, dass neben *Pfad* der komplette Pfadname der Datei *system.dll* steht. Das ist die Datei, in der der Code für `Console.WriteLine` (und für viele andere Funktionen) enthalten ist. Im Projekt muss dieser Verweis enthalten sein (auch er wurde automatisch von der Vorlage *Konsolenanwendung* erstellt), damit Visual C# 2010 Express weiß, wo sich die Funktion befindet.

## Groß- und Kleinschreibung wird unterschieden

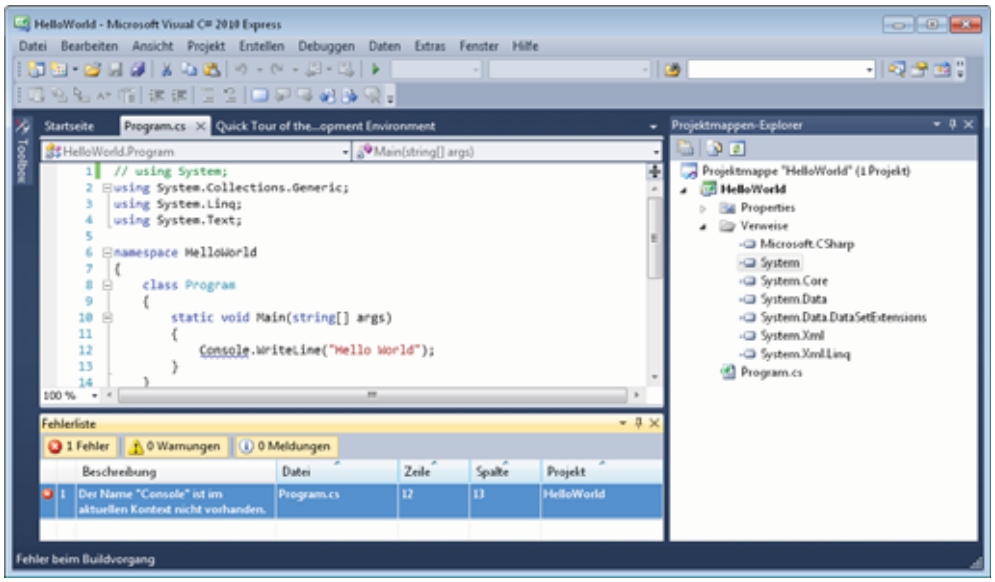
C# gehört zu den sogenannten case-sensitiven Sprachen, d.h., Groß- und Kleinschreibung werden unterschieden, und zwar immer. Die Direktive `using system` ist also eine andere Direktive als `using System`, auch wenn sich Letztere nur durch das große S von Ersterer unterscheidet. Eine falsche Groß-/Kleinschreibung gehört mit zu den häufigsten Fehlern, die Programmiereinsteiger in C# machen, besonders dann, wenn sie vorher in Visual Basic programmiert haben, in dem dies keine Rolle spielt. Lassen Sie sich einfach von IntelliSense helfen und wählen Sie die Klassenmethoden, die Sie verwenden wollen, in der Liste aus.

**Übung: Hilfe zu Syntaxfehlern**

Löschen Sie am Zeilenende von einer der `using`-Direktiven das Semikolon. An der Stelle, an der sich das Semikolon befand, wird eine rote Wellenlinie angezeigt, die Sie auf einen Syntaxfehler in Ihrem Quellcode aufmerksam macht. Zeigen Sie mit dem Mauszeiger auf die rote Wellenlinie, lesen Sie die Information zum Fehler in dem Tooltipp-Fenster, das erscheint, und korrigieren Sie den Fehler.

**Übung: Kommentar eingeben, Fehler produzieren, Klassenmethoden verwenden**

Setzen Sie die Einfügemarke an den Anfang der ersten Zeile des Quellcodes und geben Sie zwei Slashes ein //. Zeilen, die mit einem Doppelslash beginnen, sind in C# Kommentarzeilen und werden beim Erstellen des Programms ignoriert. Lassen Sie die Anwendung neu erstellen (Strg+F5) und im unteren Bereich des Fensters öffnet sich die Fehlerliste. Das Programm kann nicht mehr erstellt werden, da der Name `Console` nicht mehr bekannt ist. Compilerfehler werden im Quelltext mit einer blauen Wellenlinie markiert.




Ändern Sie Zeile 12 so ab, dass dort der qualifizierte, vollständige Name von `WriteLine` (also `Sytem.Console.WriteLine`) steht. Erstellen Sie das Programm erneut. Jetzt kann die Klassenmethode `WriteLine` wieder gefunden werden. Sie können in Ihren Programmen beide Varianten verwenden; da der Quellcode kürzer und übersichtlicher wird, ist die Verwendung der `using`-Direktive empfehlenswert.

## Die namespace-Direktive

Während Sie in C# eigene Programme erstellen, erstellen Sie permanent Namen: Namen für Klassen, Namen für Methoden, Namen für Variablen usw. Um mögliche Verwechslungsgefahren (vor allem bei größeren Projekten, an denen viele Programmierer mitarbeiten und in denen häufig Klassenbibliotheken verschiedener Hersteller eingesetzt werden) zu umgehen, werden in .NET Framework Namensräume (engl. namespaces) verwendet, um den Gültigkeitsbereich der Namen zu steuern und zu organisieren.

Implizit kennen Sie dies schon von der Verwendung der Klassenmethode `WriteLine` her und von den Informationen zur `using`-Anweisung auf Seite 33. Diese Methode ist Bestandteil der Klasse `Console`, diese Klasse wiederum befindet sich im Namensraum `System`.

Auch das kleine »Hello World«-Programm definiert seinen eigenen Namensraum, `HelloWorld`, und zwar in Zeile 6. Als Sie das neue Projekt erstellt und den Namen der Anwendung festgelegt haben, wurde dieser Name beim Erstellen des Codegerüstes von der IDE als Namensraum verwendet.

Die Definition eines Namensraums beginnt mit dem Schlüsselwort `namespace`, gefolgt vom Namen des Namensraums. Jeder Namensraum muss in geschweiften Klammern eingeschlossen sein. Die öffnende geschweifte Klammer sehen Sie in Zeile 7, die schließende geschweifte Klammer in Zeile 15. Ein Semikolon darf hier nicht verwendet werden.

```
namespace HelloWorld
{
   // die in diesem Namensraum gültigen Klassen und andere Typen werden hier definiert
}
```

Sie können den gleichen Namensraum auch in mehreren Quelldateien definieren. Wenn Sie beispielsweise eine Klassenbibliothek im Namensraum `MeineTools` erstellen und für jede Klassenmethode eine eigene Quelldatei erstellen, können Sie in jeder Quelldatei den Namensraum `MeineTools` definieren.

Im Namensraum selbst werden dann die Klassen und deren Methoden definiert, womit wir beim nächsten Thema wären.

## Die Klasse Program

In .NET sind alle Typen Klassen, was daher kommt, dass es sich beim .NET Framework um ein objektorientiertes, klassenbasiertes Framework handelt. Alle Methoden und andere Typen müssen innerhalb von Klassen definiert werden. Dies trifft auch auf das kleine Beispielprogramm zu, das die Klasse `Program` definiert. `Program` ist der Standardklassenname, den die IDE zuweist, wenn Sie eine Konsolenanwendung erstellen.

Die Klassendefinition wird mit dem Schlüsselwort `class` eingeleitet, dem der Name der Klasse folgt. Wie auch bei der `namespace`-Direktive muss die Klassendefinition von geschweiften Klammern eingeschlossen sein (Zeilen 9 und 14) und auch hier darf kein Semikolon verwendet werden.

```
namespace HelloWorld
{
   class Program
   {
      // Methoden und andere Typen der Klasse werden hier definiert
   }
}
```

## Die Klassenmethode Main

In der Klasse `Program` unseres Beispielprogramms gibt es genau eine Methode, und zwar die Methode `Main`. Die Methode `Main` ist der Einstiegspunkt für jede Konsolenanwendung und muss immer vorhanden sein.

Auch wenn wir Methoden genauer erst in einem späteren Kapitel besprechen, hier ein kurzer Überblick. An Methoden können Parameter übergeben werden, die hinter dem Namen der Methode in runden Klammern angegeben werden. Die Methode `Main` besitzt genau einen Parameter, in dem sich die Befehlszeilenargumente, die beim Starten des Programms angegeben werden, befinden. Es ist auch möglich, eine Main-Methode zu verwenden, die keine Parameter erhält, und dann so aussieht (die runden Klammern hinter dem Methodennamen bleiben einfach leer):

```
static void Main()
{
   // Anweisungen in der Methode Main
}
```

Vor dem Namen der Methode werden zwei Schlüsselwörter verwendet: `static` und `void`.

C# kennt zwei verschiedene Arten von Objekten: Objekte, von denen zuerst eine Instanz erstellt werden muss, bevor man sie verwenden kann, und statische Objekte. Von den meisten Klassen, die Ihnen in der .NET-Klassenbibliothek begegnen und die Sie selbst erstellen, erstellen Sie zuerst ein Objekt und greifen dann über das erzeugte Objekt bzw. den Namen, den Sie diesem Objekt zugewiesen haben, hierauf zu. Sie können mehrere Instanzen dieser Art von Objekten erstellen.

Bei statistischen Objekten hingegen können und dürfen Sie keine Instanzen erstellen. In einem Programm darf auch immer nur eine einzige Kopie eines statischen Objekts vorhanden sein. Zu dem Zeitpunkt, zu dem die Main-Methode als Einstiegspunkt der Anwendung aufgerufen wird, existiert noch keine Objektinstanz der Klasse `Program`. Dies ist der Grund, warum die Methode `Main` mit dem Schlüsselwort `static` versehen werden muss. (Auf das Thema statische Methoden und Objektinstanzen kommen wir ausführlicher zurück, wenn wir uns mit dem Erstellen von Klassen in C# beschäftigen.)



Das zweite Schlüsselwort, das Sie vor dem Methodennamen sehen, ist `void`. Methoden können Werte zurückgeben, müssen es aber nicht. Mit dem Schlüsselwort `void` wird angegeben, dass `Main` keinen Rückgabewert besitzt. Die Methode `Main` kann aber auch so definiert werden, dass sie eine Zahl an den Aufrufer zurückgibt. Sinnvoll ist dies beispielsweise, wenn Sie eine Konsolenanwendung aus einer Stapeldatei heraus aufrufen und das Resultat der Konsolenanwendung in der Stapeldatei mit IF ERRORLEVEL auswerten wollen. Einer der Datentypen in C#, mit dem Zahlen dargestellt werden, lautet `int`, und `int` ist als Rückgabewert von `Main` (neben `void`) erlaubt. Die Definition von `Main` mit Rückgabewert sieht so aus:

```
static int Main()
{
   // Anweisungen in der Methode Main
   return 0;
}
```

Um von einer Methode einen Wert zurückzugeben, wird in C# das Schlüsselwort `return` verwendet. Obiges Beispiel gibt also eine 0 an den Aufrufer zurück.

Der qualifizierte Name der Main-Methode setzt sich aus dem Namen des Namensraums, dem Namen der Klasse und dem Namen der Methode zusammen: `HelloWorld.Program.Main`. Die drei Elemente werden mit dem Punkt-Operator miteinander verbunden. Kommt Ihnen das bekannt vor? Richtig! Der Aufgabe des qualifizierten Namens einer von Ihnen selbst erstellten Methode folgt den gleichen Konventionen, wie Sie es bereits von der Verwendung der Klassenmethode `WriteLine` her kennen (siehe auch die Übung auf Seite 35).

# 3.4 Ein Programm mit Ein- und Ausgabe

Nach diesem kurzen theoretischen Ausflug hinter die Kulissen von .NET Framework wird es wieder Zeit, eine kleine Anwendung zu erstellen. Im Unterschied zum ersten Beispiel in diesem Kapitel soll die nächste Konsolenanwendung etwas interaktiver sein. Der Benutzer des Programms soll seinen Namen

eingeben können, woraufhin er dann persönlich begrüßt wird. In diesem Beispiel lernen Sie weitere Methoden aus dem Namespace `System` kennen.

1. Erstellen Sie ein neues Projekt. Verwenden Sie die Vorlage *Konsolenanwendung* und nennen Sie das Projekt **HelloUser.** (Die einzelnen Schritte können Sie ab Seite 29 nachlesen.)
2. Geben Sie in die Klassenmethode *Main* den folgenden Quellcode ein:

```
static void Main(string[] args)
{
   Console.Write("Bitte gib deinen Namen ein und drücke Enter: ");
   Console.WriteLine("Hallo {0}", Console.ReadLine());
   Console.WriteLine("Schön, dich zu sehen");
   Console.WriteLine();
}
```

3. Starten Sie das Programm durch Drücken von Strg+F5, geben Sie Ihren Namen ein und schauen Sie, was passiert.

Der Quellcode verwendet zwei neue Methoden der Klasse `Console`. In der ersten Anweisung in der `Main`-Methode wird zur Ausgabe des Aufforderungstextes `Console.Write` verwendet. Im Unterschied zu `WriteLine` wird nach der Ausgabe des Textes kein Zeilenwechsel durchgeführt. Der Cursor bleibt also in der gleichen Zeile stehen, in der auch der Text ausgegeben wurde.

In der zweiten Anweisung in *Main* werden zwei Methoden ineinander verschachtelt aufgerufen. `Console.WriteLine` kennen Sie bereits, jedoch wird hier eine Variante mit zwei Parametern verwendet. Der erste Parameter `Hallo {0}` besteht aus dem Text `Hallo` und der Angabe des Platzhalters `{0}`. Platzhalter bestehen aus einer Zahl, die in geschweifte Klammern eingefasst wird. Bei der eigentlichen Textausgabe wird der Platzhalter `{0}` durch den Text ersetzt, der mit `ReadLine` eingelesen wurde. `ReadLine` liest eine Zeile von der Tastatur ein, d.h., es werden so lange Zeichen eingelesen, bis der Anwender die Eingabetaste drückt. Diese eingelesene Textzeile wird dann anstelle des Platzhalters `{0}` ausgegeben. `ReadLine` wird vor `WriteLine` aufgerufen, da `WriteLine` die von `ReadLine` eingelesenen Informationen benötigt, um die Ausgabe auf dem Bildschirm machen zu können. (Im Abschnitt auf der folgenden Seite wird diese Verschachtelung genauer untersucht.)

**Tipp: Mehrere Platzhalter verwenden**

Im obigen Beispiel wurde im ersten Parameter für `Console.WriteLine` nur ein Platzhalter verwendet. Sie können jedoch auch mehrere Platzhalter verwenden. Der Platzhalter {0} entspricht dem ersten Argument nach der Zeichenkette, der Platzhalter {1} dem zweiten Argument nach der Zeichenkette usw. Die nummerierten Platzhalter können in beliebiger Reihenfolge in der Zeichenkette, dem ersten Argument, angegeben werden. Hierzu ein kleines Beispiel, in dem drei Variablen mit dem Vornamen, Nachnamen und dem Alter definiert und dann mit `WriteLine` angezeigt werden (Variablen sind das Thema des folgenden Kapitels, daher sei hier nur kurz erwähnt, dass Variablen Namen darstellen, über die in Programmen Daten gespeichert werden können):

```
string vorname = "Karl";
string nachname = "Müller";
string alter = "34";
Console.WriteLine("Sie heißen {1}, {0} und sind {2} Jahre alt", vorname, nachname, alter);
```

Dieses Programm erzeugt folgende Ausgabe: »Sie heißen Müller, Karl und sind 34 Jahre alt«.

Wenn Sie den gleichen Platzhalter an mehreren Stellen in der Zeichenkette verwenden, wird er mehrmals ausgegeben.

## Den Programmcode zeilenweise ausführen lassen

Eine weitere Besonderheit dieser Anweisungszeile besteht darin, dass hier zwei Methoden der .NET-Klassenbibliothek ineinander verschachtelt aufgerufen werden. Die Ausgabe mit `WriteLine` kann erst dann erfolgen, wenn die Eingabe durch `ReadLine` eingelesen ist. Das bestimmt daher auch die Reihenfolge, in der die Methoden aufgerufen werden: nämlich zuerst `ReadLine` und dann `WriteLine`.

Visual C# 2010 Express enthält einen integrierten Debugger, mit dem Sie Ihre Programme zeilenweise ausführen oder sich während der Ausführung der Programme den Inhalt von Variablen anzeigen lassen können u.v.m. Ausführlicher werden wir uns mit dem Debugger in einem eigenen Kapitel beschäftigen. In diesem Abschnitt werden Sie den Debugger verwenden, um *HelloUser* zeilenweise ausführen zu lassen. So können Sie auch gut erkennen, in welcher Reihenfolge die Bibliotheksfunktionen aufgerufen werden.

1. Wählen Sie den Befehl *Debuggen/Prozedurschritt* oder drücken Sie die Taste F10.

   An der linken Seite des Editor-Fensters sehen Sie einen gelben Pfeil, der die Programmzeile anzeigt, die als Nächstes ausgeführt wird. Außerdem ist diese Zeile gelb hervorgehoben. Nach dem Starten eines Programms steht der Zeiger auf der öffnenden geschweiften Klammer der Main-Methode.
2. Drücken Sie F10, um diese Zeile auszuführen. Nun wird die erste Zeile mit `Console.Write` markiert. Drücken Sie wiederum F10, um diese Zeile ausführen zu lassen.
3. Nun steht der Ausführungszeiger auf der Zeile mit dem verschachtelten Aufruf von `WriteLine` und `ReadLine`. Drücken Sie F10 und wechseln Sie ggf. zum Fenster mit der Eingabeaufforderung, in der das Programm ausgeführt wird.
4. Sie sehen den blinkenden Cursor hinter der Aufforderung *Bitte gib deinen Namen ein,* woran Sie erkennen, dass in der aktuellen Programmzeile zuerst `ReadLine` ausgeführt wird. Wenn Sie nun Ihren Namen eintippen und die Eingabetaste drücken, wird der Text *Hallo* gefolgt vom eingegebenen Namen angezeigt.
5. Wählen Sie *Debuggen/Weiter* oder drücken Sie F5, um das Programm bis zum Ende ausführen zu lassen.

# 3.5 Übungen zu diesem Kapitel

In diesem Abschnitt finden Sie einige Übungen zu diesem Kapitel. Die richtigen Antworten, Lösungen und den Programmcode finden Sie wie immer auf der Website *www.vsxpress.de.*

1. Mit welchem Befehl können Sie die Startseite von Visual C# 2010 Express anzeigen lassen?
2. Erstellen Sie ein neues Projekt, verwenden Sie als Projekttyp *Konsolenanwendung* und geben Sie mit der Methode `Console.WriteLine` einen beliebigen Text aus.
3. Lassen Sie das in Übung 2 erstellte Programm schrittweise ausführen.

# 3.6 Zusammenfassung

Dieses Kapitel hat Sie mit den wichtigsten Elementen der Benutzeroberfläche von Visual C# 2010 Express vertraut gemacht. Sie wissen nun, welche Bereiche sich auf der Startseite befinden und welche Komponenten. Beim Schreiben des ersten Programms haben Sie gesehen, dass Visual C# 2010 Express Vorlagen für verschiedene Projektarten enthält, die fertige Grundgerüste für die jeweilige Projektart erstellen. Ihre Aufgabe beim Schreiben eines Programms besteht dann darin, dieses Gerüst mit den Anweisungen zu füllen, damit das Programm den beabsichtigten Zweck erfüllt.

Konsolenanwendungen sind Programme, die keine grafische Benutzeroberfläche besitzen, sondern stattdessen in einem Eingabeaufforderungsfenster dargestellt werden. Jede Konsolenanwendung muss eine Methode mit dem Namen `Main` enthalten (das Gerüst hierfür wird von der Vorlage erstellt), die die Kontrolle erhält, wenn das Programm gestartet wird. Diese Methode befindet sich in einer Klasse, die von der Vorlage den Namen `Program` erhalten hat. Diesen Namen könnten Sie ändern, das ist aber in den Beispielen der nächsten Kapitel nicht erforderlich. (Das Thema Klassen wird uns ausführlich in Kapitel 11 dieses Buches beschäftigen.)

Damit Sie beim Programmieren auf nützliche Funktionen zugreifen und diese nicht selbst programmieren müssen, steht Ihnen die .NET-Klassenbibliothek zur Verfügung. In den Beispielen dieses Kapitels haben Sie einige der Funktionen (die im .NET Framework *Methoden* genannt werden) kennengelernt, mit denen fertige Texte auf dem Bildschirm ausgegeben und Eingaben vom Benutzer eingelesen werden können: `WriteLine`, `Write` und `ReadLine`.

Mit der `using`-Anweisung können Sie Bibliotheksfunktionen in den Quellcode Ihres Programms importieren, wodurch es möglich ist, statt des vollständigen, qualifizierten Namens über die Kurzschreibweise die Methoden der .NET-Klassenbibliothek zu verwenden. Visual C# 2010 Express fügt abhängig davon, welche Projektart Sie verwenden, in den Ordner *Verweise* des Projektmappen-Explorers die Verweise auf die Dateien ein, in denen sich der Code der Bibliotheksfunktionen befindet. Dies ist nötig, damit Sie die Methoden der Bibliotheken in Ihrem Programm verwenden können.

Methoden können Parameter besitzen, die in runden Klammern nach dem Namen der Methode angegeben werden und über die die Methode weitere Informationen dazu erhält, was sie machen soll. Bei `WriteLine` haben wir als Parameter die Texte angegeben, die angezeigt werden sollen. Methoden können außerdem einen Rückgabewert besitzen, über den die Codestelle, an der die Methode verwendet wird, Informationen darüber erhält, was in der Methode geschehen ist. So haben Sie den Rückgabewert von `ReadLine` (die den vom Benutzer eingegebenen Text zurückgibt) verwendet, um diesen dann wiederum als Parameter an `WriteLine` zu übergeben. Das Thema Methoden wird ausführlich in Kapitel 8 dieses Buches behandelt, in dem Sie lernen werden, wie Sie eigene Methoden erstellen. Bis dahin werden Sie viele weitere Methoden der Klassenbibliothek kennenlernen, die genauso einfach zu verwenden sind wie die Methoden, die Sie in diesem Kapitel eingesetzt haben.

Im nächsten Kapitel lernen Sie, wie Sie Variablen verwenden, um Informationen in einem Programm zwischenzuspeichern, und welche Datentypen Ihnen dazu zur Verfügung stehen.