Programmierkonventionen

Inhalt

[Einleitung 2](#_Toc4509471)

[Wozu Konventionen? 2](#_Toc4509472)

[Dokumentationskonventionen 2](#_Toc4509473)

[XML Dokumentationskommentaren 2](#_Toc4509474)

[Inline Dokumentation 3](#_Toc4509475)

[Konventionen für Kontrollstrukturen 3](#_Toc4509476)

[Benennungskonventionen 4](#_Toc4509477)

[Alle Namen 4](#_Toc4509478)

[Eigenständige Worte in Namen 4](#_Toc4509479)

[Namen von Klassen, Methoden und Eigenschaften 4](#_Toc4509480)

[Namen von Parametern/Argumenten und lokalen Variablen 4](#_Toc4509481)

[Namen von Feldern 4](#_Toc4509482)

[Konventionen zum Layout 4](#_Toc4509483)

[Zeilenumbrüche 4](#_Toc4509484)

[Einrückungen 4](#_Toc4509485)

[Leerzeilen 5](#_Toc4509486)

# Einleitung

Mit diesem, bei weitem nicht erstem Versuch, Konventionen für unsere Programmierarbeit zu beschreiben, nähere ich mich dem Thema nicht von der formalen, sondern von der praktischen Seite. Schließlich sollen sie uns die Arbeit beim Schreiben und, vielleicht wichtiger, beim Lesen und Verstehen von geschriebenem Code erleichtern.

Die Konventionen werden für die Programmiersprache C# beschrieben. In den meisten Fällen sollten sie jedoch auf andere Programmiersprachen anwendbar sein.

# Wozu Konventionen?

* Sie sorgen für ein einheitliches Erscheinungsbild des Codes. Leser können sich auf den Inhalt konzentrieren.
* Sie ermöglichen es den Lesern, den Code schneller zu verstehen, da einheitliches Layout den Lesern ermöglicht, Erfahrungen zu sammeln und anzuwenden.
* Sie erleichtern die Pflege und Wartung des Codes.

# Dokumentationskonventionen

Bei der Dokumentation unseres Codes unterscheiden wir die Dokumentation von Klassen-, Methoden- und Eigenschaftendeklarationen mit XML Dokumentationskommentaren und die Inline Dokumentation von Code(-blöcken).

## XML Dokumentationskommentaren

Klassen-, Methoden- und Eigenschaftendeklarationen werden mit XML Kommentaren dokumentiert. Verweise werden mit den eingebetteten XML-Tags (<see cref=""/>, <paramref name=""/>, usw.) dokumentiert.

XML-Kommentare werden, anders als Inline-Kommentare, mit „///“ eingeleitet.

Neben der eigentlichen Dokumentation ermöglichen sie die Navigation zu Verweisen und die Generierung einer XML-Dokumentationsdatei. Mit der XML-Dokumentationsdatei und einem geeigneten Tool kann eine Hilfedatei erzeugt werden.

Beispiel:  
 /// <summary>  
 /// Die Klasse Person repräsentiert die Eigenschaften, mit denen eine natürliche   
 /// Person beschrieben, gesucht und wiedererkannt werden kann.  
 /// </summary>  
 public class Person : ModelBase  
 {  
 /// <summary>  
 /// Gibt den Nachnamen einer <see cref="Person"/> zurück oder legt ihn fest.  
 /// </summary>  
 public string Nachname  
 {  
 get  
 {  
 return \_nachname;  
 }  
 set  
 {  
 if (\_nachname != value)  
 {  
 \_nachname = value;  
 NotifyPropertyChanged();  
 }  
 }  
 }  
 }

## Inline Dokumentation

Einzelne Codezeilen und Codeblöcke werden mit Inline-Kommentaren dokumentiert.

Diese Kommentare sollten sparsam verwendet werden, da sie den Code zerreißen und schwer lesbar machen. Vor der Verwendung von Inline-Kommentaren sollten wir Alternativen prüfen:

* Aussagekräftige Namen für Variablen, Parameter, Methoden und Eigenschaften.
* Auflösen verschachtelter Kontrollstrukturen.
* Extrahieren langer Codeblöcke in kurze Methoden oder Eigenschaften.
* Kapseln von Funktionalität. Eine Funktionalität je Methode oder Eigenschaft.

# Konventionen für Kontrollstrukturen

Code steht immer in geschweiften Klammern.

Beispiel:  
if (string.IsNullOrWhiteSpace(testString))  
{  
 Console.WriteLine(testString);  
}  
und NICHT:  
if (string.IsNullOrWhiteSpace(testString))  
 Console.WriteLine(testString);

Einzeiliger Code ist zulässig, wenn angebracht:

Beispiel:  
if (string.IsNullOrWhiteSpace(testString)) { throw new ArgumentNullException(nameof(testString)); }

# Benennungskonventionen

### Alle Namen

Für alle Namen gilt: keine Unterstriche (\_) zum Trennen von Worten.

* Ausnahme: Worte in Namen von Konstanten werden vollständig groß geschrieben und eigenständige Worte darin mit Unterstrich (\_) getrennt.

### Eigenständige Worte in Namen

Der Anfangsbuchstabe jedes eigenständigen Wortes wird groß geschrieben (z.B. Nachname, SonstigeNamen).

* Ausnahme: groß geschriebene Abkürzungen werden vollständig groß geschrieben (z.B. IOStream)

### Namen von Klassen, Methoden und Eigenschaften

Namen von Klassen, Methoden und Eigenschaften folgen dem PascalCasing und beginnen mit einem Großbuchstaben, gefolgt von Kleinbuchstaben.

Beispiele:  
public class Person : ModelBase { }  
public string Nachname { get; set; }  
public void TestString(string testString) { }

* Ausnahme: groß geschriebene Abkürzungen werden vollständig groß geschrieben (z.B. GetIOStream())

### Namen von Parametern/Argumenten und lokalen Variablen

Namen von Parametern/Argumenten und lokalen Variablen folgen dem camelCasing und beginnen mit dem klein geschriebenen ersten Wort (z.B. nachname, sonstigeNamen)

* Auch groß geschriebene Abkürzungen werden am Anfang eines Namens vollständig klein geschrieben (z.B. ioStream)

### Namen von Feldern

Namen von Feldern wird ein Unterstrich vorangestellt und folgen ansonsten der Schreibweise von Parametern/Argumenten und lokalen Variablen.

# Konventionen zum Layout

## Zeilenumbrüche

Code soll so mit Zeilenumbrüchen versehen werden, dass bei einer Auflösung von 1600 Bildpunkten in der Breite, kein horizontales Scrollen notwendig ist, um den Code zu lesen. Das sind zwischen 80‑120 Zeichen pro Zeile.

Text wird in lesbare Abschnitte aufgeteilt, die miteinander verkettet werden.

Beispiel:  
public readonly string Hinweis =

$"Erzeugen Sie eine neue {nameof(Person)} mit new {nameof(Person)}() und weisen Sie den Eigenschaften" +

$"{nameof(Nachname)}, {nameof(Vornamen)} und {nameof(Geburtsdatum)} gültige Werte zu";

Namespaces und voll qualifizierte Aufrufe werden, wenn notwendig, hinter den Punkten umgebrochen.

Beispiel:  
 PropertyChanged?.Invoke(this, new System.ComponentModel.  
 PropertyChangedEventArgs(propertyName));

LINQ-Ausdrücke werden wenn notwendig oder angebracht vor den Punkten umgebrochen.

Beispiel:  
List<Model.Person> filteredPersonsList = model.Personen

.Where(p => p.Nachname != null && p.Nachname.StartsWith("X"))

.Select(p => new Model.Person()

{

Nachname = p.Nachname,

Geburtsdatum = p.Geburtsdatum,

Vornamen = p.Vornamen

})

.ToList();

## Einrückungen

Die Standardeinstellungen der Entwicklungsumgebung werden verwendet.

Die Standardeinstellungen der Entwicklungsumgebung werden nur in begründeten Fällen geändert. Alle Teammitglieder müssen dann die gleichen Einstellungen verwenden.

## Leerzeilen

Zwischen zwei Codezeilen gibt es nie mehr als eine Leerzeile.

Unnötige Leerzeilen sind zu vermeiden.