Namensräume von C#

Was ist ein Namensraum

- Ein Namensraum kann als eine Art Zusammenfassung von Elementen verstanden werden, die einen inneren Zusammenhang haben, vorstellbar als eine Art Container, in dem gleichartige Elemente zusammengefaßt sind
- Der bisher bekannte Namensraum System enthält verschiedene Klassen wie
  - Math
  - Array
  - Convert
- Durch die using – Anweisung erleichtert sich die Schreibweise, anstatt von System.Console.WriteLine kann System. weggelassen werden
- Auch eigene Projekte erhalten einen Namensraum und können damit ähnlich gruppiert werden.
- Wesentlicher Punkt von Namensräumen ist das Vermeiden von doppelten Namen – die bei Verwendung von mehreren using-Statements auftreten kann, weil dadurch der Namensraum als bekannt angesehen wird – die Namensraum-Adresse entfällt und es kann zur Namensdopplung kommen
- Ein Schreibfehler in einer Namensraumdefinition wird z.T. nicht vom Compiler bemängelt, ein using beinhaltet keinen Verweis auf die entsprechende Bibliothek

Wichtige weitere .NET Namensräume

System.Collections	enthält Schnittstellen und Klassen für nichttypisierte Auflistungen und ihre Handhabung (.Generic für typisierte)
System.Data.Odbc	Klassen für den Zugriff auf ODBC-Datenquellen
System.IO	synchrones und asynchrones Lesen und Schreiben von Datenströmen, u.a. auch Dateien
System.Text	enthält Klassen zur Konvertierung von Zeichenketten in verschiedenen Kodierungen wie ASCII, Unicode, UTF-8
System.Windows.Forms	Klassen für Windows-Anwendungen, Bereitstellung von Funktionalität zum Nutzen der Benutzeroberfläche von MS Windows
System.XML	Unterstützung des XML-Formats
…	

- Ein Namensraum bedeutet nicht, daß die Funktionalität damit genutzt werden kann – die Bibliothek mit dem Namensraum muß über einen Verweis eingebunden werden
- Für eigene Projekte sollte, insbesondere wenn Bibliotheken verwendet werden, ein unternehmensspezifische Verwendung von Namensräumen die Organisation unterstützen

Vordefinierte Hilfsklassen und Funktionen

Die Klasse Math im Namespace System

- Um mathematische Funktionen zu nutzen, steht analog zur Klasse Console für Konsolenoperationen die Klasse Math im Namensraum System zur Verfügung. Die Klasse enthält statische Methoden und Eigenschaften, auf die mit dem Punktoperator zugegriffen werden kann. Im Beispiel liefern Floor und Ceiling den ganzzahligen Teil einer Fließkommazahl, entweder nach oben (Ceiling) auf- oder nach unten (Floor) abgerundet

```
//Nutzung von statischen Methoden der Klasse Math
double mm_floor = Math.Floor(3.44);   //liefert 3.0
double mm_ceil = Math.Ceiling(1.14);  //liefert 2.0
double mm_sin = Math.Sin(Math.PI);    //liefert sin(pi) = ca. 0
```

- Wenn using System; nicht im Quelltext verwendet würde, wäre System.Math.Ceiling zu schreiben
- In der Klasse Math sind die beiden Konstanten Pi und E verfügbar

Weitere Elemente der Klasse Math

Acos	Gibt einen Winkel zurück, dessen Kosinus die angegebene Zahl ist
Asin	Gibt einen Winkel zurück, dessen Sinus die angegebene Zahl ist
Atan	Gibt einen Winkel zurück, dessen Tangens die angegebene Zahl ist
Atan2	Gibt einen Winkel zurück, dessen Tangens der Quotient zweier angegebener Zahlen ist
Ceiling	Gibt die kleinste Ganzzahl zurück, die größer oder gleich der angegebenen Zahl ist.
Cos	Gibt den Kosinus des angegebenen Winkels zurück.
Cosh	Gibt den Hyperbelkosinus des angegebenen Winkels zurück.
Equals	Stellt fest, ob zwei Instanzen von Object gleich sind. (Von Object geerbt.)
Exp	Gibt die angegebene Potenz von e zurück.
Floor	Gibt die größte Ganzzahl zurück, die kleiner oder gleich der angegebenen Zahl ist.
GetHashCode	Fungiert als Hashfunktion für einen bestimmten Typ, GetHashCode eignet sich für die Verwendung in Hashalgorithmen und Hashdatenstrukturen, z.B. in einer Hashtabelle (hash = zerhacken)
GetType	Ruft den Type der aktuellen Instanz ab. (Von Object geerbt.)

Weitere Elemente der Klasse Math

Log	Gibt den Logarithmus der angegebenen Zahl zurück
Log10	Gibt den Logarithmus einer angegebenen Zahl zur Basis 10 zurück
Max	Gibt die größere von zwei angegebenen Zahlen zurück
Min	Gibt die kleinere von zwei Zahlen zurück.
Pow	Potenziert eine angegebene Zahl mit dem angegebenen Exponenten
Round	Rundet einen Wert auf die nächste Ganzzahl oder auf die angegebene Anzahl von Dezimalstellen
Sign	Gibt einen Wert zurück, der das Vorzeichen einer Zahl angibt
Sin	Gibt den Sinus des angegebenen Winkels zurück
Sinh	Gibt den Hyperbelsinus des angegebenen Winkels zurück
Sqrt	Gibt die Quadratwurzel einer angegebenen Zahl zurück
Tan	Gibt den Tangens des angegebenen Winkels zurück
Tanh	Gibt den Hyperbeltangens des angegebenen Winkels zurück
Truncate	Berechnet den den ganzzahligen Teil einer Zahl.

Kombinierte Formatierung

- Wenn Texte mit Zahlen kombiniert oder in besonderer Weise formatiert werden sollen, stellt das in manchen Programmiersprachen ein Problem dar.
- Für einfache Kombinationen bietet C# den +-Operator an, der wegen der ToString() Methode, die alle Datentypen kennen, beliebige Typen und Texte verketten kann, was in vielen Fällen ausreichend ist. Alternativ bietet C# hier auch die kombinierte Formatierung an, bei der im Text Platzhalter mit einer 0-basierten Nummer eingefügt werden, die mit den folgenden (Zahlen-)Parametern befüllt (und entsprechend formatiert) werden, Textparameter werden ohne Zahlenformatierung dargestellt

```
string mm_txt;
//Der string soll sich aus mehreren Teilen zusammen setzen
mm_txt = "Das Jahr hat " + 365 + " Tage";
//Dies kann auch durch kombinierte Ausgabe erfolgen
mm_txt = string.Format("Das Jahr hat {0} Tage und {1} Monate", 365, 12);
```

- Die kombinierte Formatierung wird in diesem Beispiel durch die (statische) Methode Format der Klasse string angeboten, sie steht aber auch für wie Write- oder WriteLine-Methoden der Klasse Console und verschiedene andere zur Verfügung
- In den geschweiften Klammern steht die Referenz zum folgenden Parameter und eine Formatierung für die Darstellung des Parameters

Kombinierte Formatierung

- Im einfachsten Fall referenziert {n} den n+1-ten Parameter der folgenden Parameter-Liste – n ist also 0-basiert
- Durch die Angabe von {n,m} erfolgt eine rechtsbündige Ausgabe mit m Stellen – hat die Zahl mehr Stellen, wird über den Rand hinaus geschrieben, also z.B. bei {0,6} die Angabe des ersten Parameters mit 6 Stellen (ggfs. aufgefüllt mit Leerzeichen oder mehr als 6 Stellen, wenn länger)
- Durch die Angabe von {n:X} kann ein vordefiniertes Zahlenformat für die Ausgabe gewählt werden (C für Währung, D für Dezimalzahl, E für exponentielle Schreibweise, F für Festkomma, G für möglichst kompakte Datenform, N für Nummern mit Tausendertrennzeichen, P für Prozentangaben oder X für eine Darstellung als Hexadezimalzahl – alle Buchstaben können auch als Kleinbuchstabe angegeben werden), dem Zahlenformatbuchstaben kann eine Zahl folgen, die bspw. die Anzahl der Nachkommastellen bestimmt (gerundet), ohne Angabe wird die Genauigkeit durch NumberFormatInfo festgelegt

- Hinweis: Die kombinierte Formatierung steht nicht nur über die Funktion Format am String zur Verfügung, sondern wird auch von vielen Ausgabefunktionen unterstützt (wie WriteLine)

Beispiele zur Standardformatierung

```
Console.WriteLine("Pi auf 4 Stellen: {0:F4}\n"
    + "Pi mit Systemgenauigkeit: {0:F}\n"
    + "Pi in Kompaktschreibweise mit drei Stellen: {0:G3}\n"
    + "Pi in Exponentialschreibweise mit drei Stellen: {0:E3}"
    , Math.PI);
Console.WriteLine("Standardgenauigkeit ist: {0} Nachkommastellen",
    NumberFormatInfo.CurrentInfo.NumberDecimalDigits );
```

Die Ausgabe ist:

```
Pi auf 4 Stellen: 3,1416
Pi mit Systemgenauigkeit: 3,14
Pi in Kompaktschreibweise mit drei Stellen: 3,14
Pi in Exponentialschreibweise mit drei Stellen: 3,142E+000
Standardgenauigkeit ist: 2 Nachkommastellen
```

Benutzerdefinierte Zahlenformate

- Zusätzlich zu den Standardformatierungen gibt es benutzerdefinierte Angaben, die weitere Möglichkeiten anbieten:
  - # als Platzhalter für eine führende oder nachfolgende Leerstelle
  - 0 als Platzhalter für eine führende oder nachfolgende 0
  - . Position des Dezimalpunktes
  - , Position des Tausendertennzeichen
  - % Ausgabe als Prozentzahl (Wert wird mit 100 multipliziert)
  - ; Ermöglicht Angabe von drei Formatierungen für Zahlen kleiner, gleich und größer

```
Console.WriteLine("Pi auf 4 Nachkommastellen mit 3 Vorkommastellen: {0:000.0000}\n"
    + "3/8 auf bei Bedarf 6 Nachkommastellen: {1:#.######}\n"
    + "3/8 wie zuvor aber mit Vorkommanull: {1:0.######}\n"
    + "{2,15:N2} Euro ist eine Menge Geld\n"
    + "{3,15:N2} Euro ist aber mehr\n"
    , Math.PI, 3.0 / 8.0, 512456.28432, 1234567.543);
```

Ausgabe:

```
Pi auf 4 Nachkommastellen mit 3 Vorkommastellen: 003,1416
3/8 auf bei Bedarf 6 Nachkommastellen: ,375
3/8 wie zuvor aber mit Vorkommanull: 0,375
     512.456,28 Euro ist eine Menge Geld
   1.234.567,54 Euro ist aber mehr
```

Der DateTime-Typ und verwandte Typen

- Wird i.d.R durch eine als Datum interpretierte Gleitkommazahl abgebildet, der Vorkommateil gibt die Anzahl der Tage seit einem definierten Startdatum an, der Nachkommateil die Uhrzeit als Bruchteil von 24h
- Für verschiedene Funktionsbereiche gibt es verschiedene Datumsformate, z.B. das Datumsformat für Dateien oder wenn ein Datum über COM mit anderen Applikationen ausgetauscht werden soll. Auch Datenbanken haben Datumsformate, für die andere u.U. andere Darstellungen gelten
- In C# ist DateTime eine Klasse, die viele nützliche Funktionen beinhaltet, unter anderem zum Messen von Zeitspannen und zur Konvertierung der Datumsformate

```
Console.Write("Ihr Geburtsdatum ist: ");
string mm_geburtsDatum = Console.ReadLine();

DateTime mm_heute = DateTime.Now;
DateTime mm_geburtstag = DateTime.Parse(mm_geburtsDatum);
TimeSpan mm_aufDerWelt = mm_heute - mm_geburtstag;
int mm_tage = mm_aufDerWelt.Days;

Console.WriteLine("Sie sind heute {0} Tage alt", mm_tage);
```

Datum und Zeit

```
// Datum festlegen (heute)
DateTime date = DateTime.Now;
CultureInfo deutsch = new CultureInfo("de-DE");
//Gibt den Deutschen Wochentagsnamen
string day = deutsch.DateTimeFormat.DayNames[(int)date.DayOfWeek];
//Gibt den englischen Wochentagsnamen
string eday = date.DayOfWeek.ToString();
```

Hinweis:
ToLongDateString() liefert bspw.:
Montag, 01. April 2013
berücksichtigt also wie ToShortDateString() usw. die aktuellen CultureInfo-Einstellungen

```
Console.Title = "Aktuelles Datum und aktuelle Uhrzeit ermitteln";

// Aktuelles Datum ermitteln
DateTime now = DateTime.Now;
string currentDate = now.ToShortDateString();
string currentTime = now.ToShortTimeString();

// Datum und Zeit ausgeben
Console.WriteLine("Datum: {0}", currentDate);
Console.WriteLine("Zeit: {0}", currentTime);

Console.WriteLine();
Console.WriteLine();
Console.WriteLine("Beenden mit Return");
Console.ReadLine();
```

Die Klasse Convert

- Diese Klasse bietet einige hilfreiche Konvertierungsroutinen, die Typecasting zweckmäßig ergänzen, einige interessante Methoden, die alle mit verschiedenen Argumenttypen umgehen können (d.h. die Methoden sind überladen):
  - ToBoolean Konvertiert einen angegebenen Wert in einen entsprechenden booleschen Wert.
  - ToByte Konvertiert einen angegebenen Wert in eine 8-Bit-Ganzzahl ohne Vorzeichen.
  - ToChar Konvertiert einen angegebenen Wert in ein Unicode-Zeichen.
  - ToDateTime Konvertiert den angegebenen Wert in ein DateTime.
  - ToDecimal Konvertiert den angegebenen Wert in eine Decimal-Zahl.
  - ToDouble Konvertiert einen angegebenen Wert in eine Gleitkommazahl mit doppelter Genauigkeit.
  - ToInt16 Konvertiert einen angegebenen Wert in eine 16-Bit-Ganzzahl mit Vorzeichen.
  - ToInt32 Konvertiert einen angegebenen Wert in eine 32-Bit-Ganzzahl mit Vorzeichen.
  - ToInt64 Konvertiert einen angegebenen Wert in eine 64-Bit-Ganzzahl mit Vorzeichen.
  - ToSByte Konvertiert einen angegebenen Wert in eine 8-Bit-Ganzzahl mit Vorzeichen.
  - ToSingle Konvertiert einen angegebenen Wert in eine Gleitkommazahl mit einfacher Genauigkeit.
  - ToString Konvertiert den angegebenen Wert in die entsprechende String-Darstellung.
  - ToUInt16 Konvertiert einen angegebenen Wert in eine 16-Bit-Ganzzahl ohne Vorzeichen.
  - ToUInt32 Konvertiert einen angegebenen Wert in eine 32-Bit-Ganzzahl ohne Vorzeichen.
  - ToUInt64 Konvertiert einen angegebenen Wert in eine 64-Bit-Ganzzahl ohne Vorzeichen.

Die Klasse Convert

- Allein für die Methode ToString() existiert eine große Anzahl verschiedener Varianten:
  - ToString() Gibt einen String zurück, der den aktuellen Object darstellt. (Von Object geerbt – also der Standard-Fall)
  - ToString(Datentypvariante (z.B. bool, int, double usw.)) statische Methode, konvertiert den Wert des Übergabeparameters in die entsprechende String-Darstellung.
  - ToString(Datentypvariante (z.B. bool, int, double usw.), IFormatProvider) Konvertiert den Wert des ersten Arguments in die entsprechende String-Darstellung.
  - ToString(Ganzzahl-Typ, Int32) Konvertiert den Wert des ersten Parameters in die entsprechende Zeichenfolgendarstellung bezüglich einer angegebenen Basis (2. Parameter), z.B. 23 in 10111 für den Aufruf ToString(23, 2)

Richtlinien im Softwareentwicklungsprozess

Sinn und Zweck von Richtlinien

- Steigerung der Lesbarkeit
  - ein Großteil des Aufwandes während eines "Softwarelebens" entfällt auf das Lesen und Verstehen von Quelltext
  - Übersichtlichkeit (Formatierung und Einrückung, aber auch Länge von Methoden und Klassen)
  - Kommentare und Benennung von Variablen, Klassen, Methoden, …
- Steigerung der Wiederverwendbarkeit
  - Modularität von Software
  - objektorientierter Ansatz
  - Bibliotheken
  - Quelltextverwaltung (wie z.B. svn)
- Steigerung der Teamfähigkeit
  - Verteilung von Aufgaben gemäß vorhandener Kompetenzen
  - Einfache Nutzung komplexer Funktionalität
  - Paralleles Arbeiten
- Steigerung der Softwarewarequalität
  - Anforderungsmanagement und Testszenarien
  - Dokumentation von Abläufen
  - Fehlermanagement

Richtlinien für die Erstellung von Quelltext

- Programmiertechniken und -richtlinien
  - Festlegen verbindlicher Regeln für die Erstellung von Quellcode
  - siehe auch nächster Abschnitt
- Modularität
  - Strukturieren von Klassen, so daß sie möglichst eigenständig verwendbar sind
  - Verwendungsnachweis (ähnlich wie in Baugruppen), welche Abhängigkeiten bestehen zwischen den einzelnen Klassen und Bibliotheken, s.a. UML-Diagramm
  - Zusammenfassung von wiederzuverwendender Funktionalität in Bibliotheken
- Softwareentwicklungsprozess
  - Projektplanung (mit Design, Anforderungsmanagement, Ressourcenmanagement, …)
  - Zwingender Einsatz eines Quelltextverwaltungssystems
  - Einsatz eines Fehlermanagementsystems
  - Erarbeitung von QS-Richtlinien
  - Festlegung der Modalitäten zum Softwaretest

Quellcodeverwaltung

- lückenlose Dokumentation der Quelltexte zu einem Projekt
  - Zugriff auf alle Änderungsstände und die verantwortlichen Personen
  - Sichtbarkeit der Strukturierung eines Projektes
- Abbilden des "Buildprozesses" in der Quellcodeverwaltung
  - Erstellung des kompilierten Programms durch festgeschriebene Automatismen
  - Ausschluss von zufälligen Fehlern im Build
- Paralleles Arbeiten am Projekt
  - exklusives oder paralleles Auschecken von Dateien und deren lokale Bearbeitung
  - Zusammenführen verschiedener Quelltexte
  - Verfolgen und visuelle Kontrolle von Änderungen
- Zugriff auf definierte Entwicklungsstände
  - Labeln und Wiederherstellen von definierten Versionen
- Internetfähige Client-Server-Architektur mit guter Anbindung an die lokale Entwicklungsumgebung
- Rechtesystem für das Management der Zugriffs-, Freigabe- und anderer Rechte basierend auf User-Gruppen-Rechten

Softwaretest

- Auswahl einer Teststrategie:
  - funktionsorientiert (Macht das Programm, was gefordert ist - Benutzersicht)
    - Anhand der Softwareanforderungen werden die einzelnen Anwendungsfälle getestet.
    - Vorteile: Überprüfung der bestehenden Anforderungen und deren Evaluation
    - Nachteile: Bei Änderungen des Codes muss aufgrund von Wechselwirkungen vollständig neu getestet werden
  - codeorientiert (Arbeiten die Methoden so, wie dies erwartet wird - Implementierungssicht)
    - Alle Funktionen eines Moduls, einer Klasse … werden getestet. Grundlage des Tests ist das Design, in dem die einzelnen Verwendungen dokumentiert sind
    - Vorteile: ein sehr gründlicher Test der verwendeten Funktionalität, bei Änderung eines Moduls ist jedoch nur dieses Modul neu zu testen
    - Nachteil: durch die kombinatorischen Möglichkeiten ein sehr aufwendiger Test (i.d.R nur für kleine Module verwendbar)
  - Mischen der Teststrategie
- Auswahl der Testfälle (sowohl code- wie auch funktionsorientiert)
  - repräsentatives Testen
    - welche Funktionen werden in welcher Häufigkeit verwendet
  - risikoorientiertes Testen
    - welche Funktionen verursachen bei Fehlern hohe Kosten (kostenintensive Datendefekte, hohe Benutzerunzufriedenheit, Prozessstabilität, …)
  - schwachstellenorientiertes Testen
    - welche Funktionen sind hinsichtlich ihrer Funktionsfähigkeit und Fehleranfälligkeit besonders gefährdet (Zugriffe, Datenkonvertierung, fehlerhafte Daten, Komplexität, …)

Softwaretest

- Auswahl der Testdaten
  - Standardwerte
    - gebräuchliche Werte, die den Normalfall repräsentieren
  - Extremwerte
    - Werte am äußersten Rande der noch sinnvollen Wertebereiche
  - falsche Werte
    - fehlerhafte Dateien oder Dateiformate, leere oder falsch initialisierte Werte, unsinnige Werte wie ein negativer Abstand
- Auswahl der Testumgebung
  - automatischer, automatisierter oder manueller Test
  - Auswahl des Testpersonals
  - Hard- und Software
- Auswahl der Testhierarchie
  - Intern: Modultest – Komponententest - Integrationstest
  - Systemtest
    - z.B. als Pilotanwendung oder Auslieferung beim Kunden, aber auch intern möglich
  - Abnahmetest
    - Inhaltlich wie Systemtest, hier aber rechtsverbindliche Abnahme der vereinbarten Leistung beim Kunden, z.B. hinsichtlich Antwortzeiten, Genauigkeit, Verfügbarkeit, …

Fehlermanagement

- Aufnehmen eines Fehlers
  - Fehlerklassen und –handling durch Log-Files, Datenbanken, Messaging, …
  - Fehlerklassifizierung (schwerwiegend bis unkritisch, individuell gestaltbar)
  - Fehlerart (logischer Fehler oder Laufzeitfehler, eine weitere Detaillierung ist möglich, erschwert aber die Zuordnung zu diesem Zeitpunkt)
  - Kontext und Reproduzierbarkeit
    - Wie kann der Fehler ausgelöst werden, Verfügbarkeit von Testdaten, Ablaufbeschreibung
    - Umgebung (Programm und dessen Version, BS-Version, …)
  - Angaben zur Entdeckung (Software, Kunde, Tester, Entwickler)
- Management
  - Definition eines Workflows zur Fehlerbeseitigung und dessen Unterstützung
  - Nachvollziehbarkeit des Fehlers
    - Zuweisung der Verantwortlichkeit und eines Termins
    - Ist der Fehler nachvollziehbar/reproduzierbar anhand der vorliegenden Informationen
  - Behebung des Fehlers
    - Zuweisung der Verantwortlichkeit und eines Termins
    - Dokumentation des behobenen Fehlers im Quelltext und/oder Quellcodeverwaltung – Link auf Fehler im Fehlermanagementsystem
  - Test und Freigabe des revisionierten Quellcodes
    - ggfs. Softwarepatch, Verteilung,
  - Steuern und Entscheiden der Fehlerbehebungsstrategien und -umsetzung
    - aktuelle oder zukünftige Release, Zeitpunkte, ...

Programmiertechniken und -richtlinien

Programmiertechniken - Forderungen

- Der vollständige Quellcode sollte ein durchgängig einheitliches Format aufweisen und so harmonisiert sein, als ob er von einem einzigen Entwickler in einer einzigen Sitzung erstellt wurde
- Dazu ist Codierungsstandard festzulegen, um die Arbeit aller Entwickler am Projekt aufeinander abzustimmen
- Wenn vorhandener Quellcode in das Softwareprojekt eingebunden oder ein vorhandenes Softwaresystem gewartet wird, muss durch den Codierungsstandard festgelegt werden, wie mit der vorhandenen Codebasis umgegangen wird
- Die Lesbarkeit des Quellcodes hat direkten Einfluss darauf, wie gut ein Entwickler das Softwaresystem erfassen kann
- Mit dem Begriff der Codeverwaltbarkeit wird beschrieben, wie einfach das betreffende Softwaresystem geändert werden kann, um neue Features hinzuzufügen, vorhandene Features zu ändern, Programmfehler zu beheben oder die Leistung zu verbessern
- Lesbarkeit und Verwaltbarkeit ergeben sich zwar aus einer Vielzahl von Faktoren, das Codierungsverfahren ist jedoch ein spezieller Aspekt der Softwareentwicklung, auf den alle Entwickler einen Einfluss haben

Quelle: Microsoft

Programmiertechniken - Forderungen

- Dass ein Team von Entwicklern qualitativ hochwertigen Code erstellt kann durch einen Codierungsstandard unterstützt werden. Anschließend sind Routinecodeüberprüfungen durchzusetzen
- Durch geeignete Codierungsverfahren und empfohlene Programmiertechniken Code soll ein wesentlicher Beitrag zur Softwarequalität geleistet werden
- Darüber hinaus führt das konsistente Anwenden eindeutig definierter Codierungsstandards und geeigneter Codierungstechniken sowie das anschließende Durchführen von Routinecodeüberprüfungen mit hoher Wahrscheinlichkeit dazu, dass als Ergebnis des Softwareprojekts ein leicht verständliches und verwaltbares Softwaresystem entsteht
- Obwohl Codeüberprüfungen während des gesamten Entwicklungszyklus hauptsächlich ausgeführt werden, um Fehler im Code zu erkennen, können diese Überprüfungen auch zur einheitlichen Durchsetzung von Codierungsstandards beitragen. Das Durchsetzen eines Codierungsstandards ist nur dann sinnvoll, wenn er vom Beginn bis zum Abschluss des gesamten Softwareprojekts eingehalten wird. Es ist weder praktikabel noch klug, einen Codierungsstandard durchzusetzen, nachdem bereits Fakten geschaffen wurden.

Quelle: Microsoft

Programmiertechniken – Namen

- Das Benennungsschema ist eines der wichtigsten Hilfsmittel für das Verständnis des logischen Flusses einer Anwendung. Ein Name sollte eher Aufschluss über das "Was" als über das "Wie" geben. Indem Sie Namen vermeiden, die die zugrunde liegende, möglicherweise veränderliche Implementierung darlegen, können Sie eine Abstraktionsebene beibehalten, die komplexe Zusammenhänge vereinfacht. Sie können z.B. GetNextStudent() anstelle von GetNextArrayElement() verwenden.
- Als Grundsatz für die Benennung gilt, dass Schwierigkeiten bei der Auswahl eines treffenden Namens möglicherweise auf eine unzureichende Analyse oder Definition des Zwecks eines Elements hinweisen.
- Die Länge der Namen sollte so gewählt werden, dass sie sinnvoll, jedoch nicht zu ausführlich sind. Für das Programm dient ein eindeutiger Name nur zum Unterscheiden der einzelnen Elemente.
- Für den Leser sind aussagekräftige Namen jedoch sehr hilfreich. Wählen Sie deshalb Namen, die für eine Person verständlich sind. Vergewissern Sie sich aber, dass die ausgewählten Namen den Regeln und Standards der jeweiligen Sprache entsprechen.

Quelle: Microsoft

Programmiertechniken – Namen

- Vermeiden Sie missverständliche Namen, die Raum für subjektive Interpretationen lassen, z.B. AnalyzeThis() für eine Routine oder xxK8 für eine Variable. Solche Namen tragen eher zu Missverständnissen als zur Abstraktion bei
- In objektorientierten Sprachen brauchen Sie keine Klassennamen in die Namen von Klasseneigenschaften aufzunehmen, z.B. Book.BookTitle. Verwenden Sie stattdessen Book.Title
- Verwenden Sie die Verb-Substantiv-Methode für die Benennung von Routinen, die eine Operation für ein angegebenes Objekt ausführen, z.B. CalculateInvoiceTotal()
- In Sprachen, die ein Überladen von Funktionen zulassen, müssen alle Überladungen eine ähnliche Funktion ausführen. Richten Sie für Sprachen, die keine Überladung von Funktionen zulassen, einen Benennungsstandard ein, mit dem ähnliche Funktionen zueinander in Beziehung gesetzt werden können
- Fügen Sie ggf. Berechnungsqualifizierer (Avg, Sum, Min, Max, Index) an das Ende eines Variablennamens an
- Verwenden Sie in Variablennamen komplementäre Paare, z.B. Min/Max, Anfang/Ende und Oeffnen/Schliessen
- Verwenden Sie Variablennamen in einem Projekt mit immer der gleichen Bedeutung, ergänzen Sie notfalls die Bedeutung mit zusätzlichen Angaben (anzahlLeute)

Quelle: Microsoft

Programmiertechniken – Namen

- Da die meisten Namen durch Verketten mehrerer Wörter gebildet werden, verbessern Sie die Lesbarkeit durch eine gemischte Groß- und Kleinschreibung. Verwenden Sie außerdem für Routinennamen die Pascal-Schreibweise (CalculateInvoiceTotal), bei der der Anfangsbuchstabe jedes Wortes großgeschrieben wird. So kann besser zwischen Variablen und Routinen unterschieden werden
- Verwenden Sie für Variablennamen eine Höckerschreibweise (camelCase), z.B. documentFormatType, bei der mit Ausnahme des ersten Wortes der Anfangsbuchstabe jedes Wortes großgeschrieben wird
- Boolesche Variablennamen sollten z.B. Is enthalten, was die Werte Yes/No oder True/False impliziert, z.B. fileIsFound
- Vermeiden Sie Begriffe wie Flag beim Benennen von Statusvariablen. Diese unterscheiden sich von booleschen Variablen darin, dass sie mehr als zwei mögliche Werte aufweisen können. Wählen Sie statt documentFlag einen aussagekräftigeren Namen wie documentFormatType
- Verwenden Sie selbst für kurzlebige Variablen, die nur in wenigen Codezeilen enthalten sind, einen sinnvollen Namen. Verwenden Sie Variablennamen aus einem Buchstaben wie i oder j ausschließlich für Indizes kurzer Schleifen

Quelle: Microsoft

Programmiertechniken – Namen

- Verwenden Sie keine Literalzahlen oder Literalzeichenfolgen wie
  `for(int i = 1;i <= 7; ++i)`
  Verwenden Sie stattdessen benannte Konstanten wie NUM_DAYS_IN_WEEK, um Verständnis und Wartung zu erleichtern. Dies gilt auch im Besonderen für die Handhabung von Einstellungen, die auf unterschiedlichen Rechnern (z.B. bei Dateinamen) unterschiedlich sein können. Hier sollte immer auf Konstanten oder sonstige Variablen zugegriffen werden
- Drücken Sie beim Benennen von Tabellen den Namen im Singular aus. Verwenden Sie z.B. Employee anstelle von Employees
- Wiederholen Sie beim Benennen von Spalten nicht den Tabellennamen; vermeiden Sie z.B. in der Tabelle Employee die Feldbenennung EmployeeLastName.
- Binden Sie den Datentyp nicht in den Namen einer Spalte ein. Dadurch sinkt der Arbeitsaufwand, wenn Sie den Datentyp später ändern müssen
- Verwenden Sie möglichst wenige Abkürzungen, und verwenden Sie die erstellten Abkürzungen konsistent. Eine Abkürzung darf nur eine Bedeutung haben, und für jedes abgekürzte Wort darf nur eine Abkürzung vorhanden sein. Wenn Sie z.B. Minimum mit min abkürzen, müssen Sie durchgängig so vorgehen und dürfen nicht auch Minute mit min abkürzen

Quelle: Microsoft

Programmiertechniken – Namen

- Namen von Ordnern und Dateien, z.B. Prozedurnamen, sollten genau ihren Zweck wiedergeben.
- Schließen Sie beim Benennen von Funktionen eine Beschreibung des zurückgegebenen Wertes ein, z.B. GetCurrentWindowName()
- Vermeiden Sie es, einen Namen für verschiedene Elemente wiederzuverwenden, z.B. eine Routine mit dem Namen ProcessSales() und eine Variable mit dem Namen iProcessSales.
- Vermeiden Sie beim Benennen von Elementen Homonyme (Teekesselchen), um Verwirrung während der Codeüberprüfungen auszuschließen, z.B. wahr und war.
- Vermeiden Sie beim Benennen von Elementen auch häufig falsch geschriebene Wörter. Beachten Sie außerdem, dass für verschiedene Regionen Unterschiede in der Schreibweise auftreten können, z.B. Januar/Jänner und Scheck/Cheque.
- Verwenden Sie beim Bezeichnen von Datentypen keine typografischen Zeichen, z.B. $ für Zeichenfolgen oder % für ganze Zahlen.
- Entscheiden Sie sich für eine Sprache, mixen Sie nicht Englisch mit Deutsch, nur bei Zusammenspiel mit Code-Inhalten wie z.B. einer Get-Methode GetUmsatz()

Quelle: Microsoft

Programmiertechniken - Kommentare

- Verwenden Sie bei der Entwicklung in C# das XML-Dokumentationsfeature (///)
- Halten Sie beim Ändern von Code die entsprechenden Kommentare stets auf dem aktuellen Stand
- Zu Beginn jeder Routine ist es hilfreich, in standardisierten Kommentaren den Zweck, die Voraussetzungen sowie die Einschränkungen der Routine anzugeben. Der standardisierte Kommentar sollte eine kurze Einführung zum Zweck und zur Leistungsfähigkeit der Routine enthalten.
- Fügen Sie keine Kommentare am Ende einer Codezeile hinzu. Kommentare am Ende der Zeile verschlechtern die Lesbarkeit des Codes. Kommentare am Ende von Zeilen eignen sich jedoch für Anmerkungen zu Variablendeklarationen. Richten Sie in diesem Fall alle Zeilenendkommentare an einem gemeinsamen Tabstopp aus.
- Vermeiden Sie überflüssige Kommentare, z.B. eine ganze Zeile mit Sternchen. Trennen Sie stattdessen Kommentare und Code durch Leerraum.
- Umgeben Sie einen Blockkommentar nicht mit einem typografischen Rahmen. Dieser mag attraktiv aussehen, er erschwert jedoch die Verwaltung.
- Entfernen Sie vor der Weitergabe alle temporären oder überzähligen Kommentare, um Verwirrung bei der zukünftigen Verwaltung zu vermeiden.

Quelle: Microsoft

Programmiertechniken - Kommentare

- Wenn Sie zur Erläuterung eines komplexen Codeabschnitts Kommentare benötigen, überprüfen Sie den Code und stellen Sie fest, ob dieser in einer veränderten Fassung geschrieben werden kann. Dokumentieren Sie nach Möglichkeit keinen minderwertigen Code. Schreiben Sie ihn stattdessen neu. Obwohl die leichte Lesbarkeit normalerweise nicht auf Kosten der Leistung des Codes erreicht werden sollte, muss ein Gleichgewicht zwischen Leistung und Verwaltbarkeit gefunden werden.
- Schreiben Sie Kommentare in Form von vollständigen Sätzen. Kommentare sollen den Code erläutern und nicht seine Verständlichkeit erschweren.
- Erstellen Sie Kommentare während des Codierens, da Sie später meist keine Zeit dafür haben. Außerdem werden Sie bei einer erneuten Überprüfung des geschriebenen Codes möglicherweise feststellen, dass der zum Zeitpunkt der Erstellung leicht verständliche Code sechs Wochen später nicht mehr verständlich ist.
- Vermeiden Sie überflüssige oder unangebrachte Kommentare wie humoristische Randbemerkungen.
- Beschreiben Sie in den Kommentaren den Zweck des Codes. Kommentare sollten keine direkten Übersetzungen des Codes darstellen.
- Kommentieren Sie alles, was nicht sofort aus dem Code ersichtlich ist.

Quelle: Microsoft

Programmiertechniken - Kommentare

- Verwenden Sie für Bugfixes und für Code zur Problemumgehung stets Kommentare, insbesondere in einer Teamumgebung. So vermeiden Sie eine Wiederholung von Problemen.
- Kommentieren Sie Code, der aus Schleifen und logischen Verzweigungen besteht. Dies sind Schlüsselbereiche, in denen Leser des Quellcodes unterstützt werden sollten.
- Erstellen Sie innerhalb der gesamten Anwendung Kommentare in einheitlichem Format und mit einheitlicher Interpunktion und Struktur.
- Trennen Sie Kommentare und Kommentartrennzeichen durch Leerraum. Dadurch werden Kommentare leicht erkennbar und auffindbar, wenn keine Ansicht mit farblicher Markierung verfügbar ist.

Quelle: Microsoft

Programmiertechniken - Formate

- Legen Sie eine Standardgröße für einen Einzug fest, z.B. vier Leerzeichen, und verwenden Sie diese durchgängig. Richten Sie Codeabschnitte mit Hilfe des vorgeschriebenen Einzugs aus.
- Verwenden Sie eine Schriftart mit fester Breite, wenn Sie ausgedruckte Versionen des Quellcodes veröffentlichen.
- Richten Sie bei Paaren geschweifter Klammern die zusammengehörigen öffnenden und schließenden Klammern vertikal aus oder verwenden Sie ein versetztes Format, aber verwenden Sie dies einheitlich

```
for (i = 0; i < 100; i++)
{
    ...
}
```

```
for (i = 0; i < 100; i++){
    ...
}
```

- Legen Sie eine maximale Länge für Kommentare und Code fest, damit Sie keine Bildläufe im Quellcodeeditor ausführen müssen und eine problemlose Darstellung in der Druckfassung möglich ist.
- Fügen Sie vor und nach den meisten Operatoren Leerzeichen ein, wenn dadurch der Zweck des Codes nicht geändert wird. Studio unterstützt dabei.

Quelle: Microsoft

Programmiertechniken - Formate

- Verdeutlichen Sie mit Hilfe von Leerraum die Gliederung des Quellcodes. Dadurch werden "Absätze" im Code geschaffen, die dem Leser das Verständnis der logischen Segmentierung der Software erleichtern. (In Studio mit Ctrl-A den gesamten Quelltext markieren und dann mit Ctrl-E-D Absätze neu formatieren)
- Vermeiden Sie nach Möglichkeit mehrere Anweisungen pro Zeile. Eine Ausnahme ist eine Schleife z.B. for (i = 0; i < 100; i++).
- Legen Sie beim Schreiben von HTML jeweils ein Standardformat für Tags und Attribute fest, z.B. nur Großbuchstaben für Tags und nur Kleinbuchstaben für Attribute. Sie können aber auch die XHTML-Spezifikation anwenden und so die Gültigkeit aller HTML-Dokumente sicherstellen. Obwohl beim Erstellen von Webseiten Erwägungen hinsichtlich der Dateigröße berücksichtigt werden müssen, sollten Sie Attributwerte in Anführungszeichen und schließende Tags verwenden, um die Verwaltbarkeit zu verbessern.
- Verwenden Sie beim Schreiben von SQL-Anweisungen für die Schlüsselwörter ausschließlich Großschreibung und für Datenbankelemente wie Tabellen, Spalten und Ansichten Groß- und Kleinschreibung.
- Verteilen Sie den Quellcode logisch auf physische Dateien – je Klasse eine Datei
- Unterteilen Sie große, komplexe Codeabschnitte in kleinere, leichter erfassbare Module.

Quelle: Microsoft

Weitere Quellen

C# - Ressourcen

- Eine wirklich gute Quelle mit vielen Beispielen ist das C# 2005 Codebuch (auf CD) von Jürgen Bayer, erschienen im Pearson-Verlag bzw. bei Addison-Wesley
- http://www.codezone.de
  Seite mit umfangreichen Infos zu Microsoft-Technologien wie C# aber auch .NET und anderes
- http://www.c-sharpcorner.com
  Quellcode und Artikel zu C# und .NET Themen
- http://www.planetsourcecode.com
  Viele Downloads zu allen möglichen Programmiersprachen, C# im Bereich .NET
- http://www.codeproject.com
  sehr gute Seite mit vielen Beispielen und Artikeln zu unterschiedlichsten Sprachen, aber auch zu Themen wie Windows-API, Datenbanken, DirectX-Grafik
- http://www.csharphelp.com
  Artikel, Beispiele und Tutorials zum Thema C#
- http://www.icsharpcode.net/OpenSource/SD
  Open Source-IDE – als Alternative zu MS Windows