|  |
| --- |
| **ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**  **Факултет – „Компютърни системи и управление“** |

**Конвенции при писане на C# код**

**Дисциплина – „Валидация и верификация на**

**програмни системи“**

**Тема – „Изчисляване на множествена регресия”**

**Студент:** Иван Николаев Колев, фак. № 121213114, гр. 43

**Специалност:** „Компютърно и софтуерно инженерство“

**Дата:** 4.22.2016 г**. Заверил:**

**/ доц. А. Алексиева/**

**София  
2017**

Съдържание

[1. Въведение 2](#_Toc480666389)

[1.1. За какво служат конвенциите при писане на програмен код? 2](#_Toc480666390)

[2. Конвенции при именуване: 2](#_Toc480666391)

[2.1. “Using’s” 2](#_Toc480666392)

[3. Конвенции за изглед на кода 2](#_Toc480666393)

[4. Конвенции при коментиране на кода 3](#_Toc480666394)

[5. Насоки при писане на C# код 3](#_Toc480666395)

[5.2. Имплицитно типизирани променливи 4](#_Toc480666396)

[5.3. Масиви 4](#_Toc480666397)

[5.4. Прихващане на изключения 5](#_Toc480666398)

[5.5. Операторите „&&” и „||” 5](#_Toc480666399)

[5.6. LINQ заявки 6](#_Toc480666400)

# 1. Въведение

## 1.1. За какво служат конвенциите при писане на програмен код?

Придават консистентен вид на кода, за да може читателите, да се фокусират върху съдържанието и бизнес логиката, а не върху оформлението.

Позволява на читателите да разбират програмния код много по-бързо, правейки предположения на основа предишния си опит.

Улесняват копирането, промяната и поддръжката на даден фрагмент код.

Демонстрират най-добрите практики при писане на C# код.

# 2. Конвенции при именуване:

## 2.1. “Using’s”

В кратки примери, които не съдържат „using” директиви, трябва да използваме пълните „namespace” квалификации. Ако знаем, че даден “namespace” е внесен в проекта по подразбиране, няма нужда да използваме пълното му наименование. Квалифицираните имена, могат да бъдат прекъснати след точката и пренесени на нов ред, ако са прекалено дълги за един ред:

***Пълна квалификация:***

***Само основното име:***

2.2. Автоматично генериран код

Няма нужда да променяме имената на обектите, които са генерирани автоматично от IDE-то или дизайнера, който използваме (в случая Visual Studio), за да се вписват в нашите конвенции.

# 3. Конвенции за изглед на кода

Добрият изглед разчита на форматиране, което набляга върху структурата на кода и го прави по-лесен за четене. Конвенциите за форматиране, които са наложени от Майкрософт са следните:

* Използване на настройките по подразбиране от редактора или IDE-то, с което пишем. Умно идентиране, идентация с четири шпации (1 табулация = 4 шпации), табулациите се записват като определен брой шпации.
* Писане на само едно „изявление“ на ред.
* Писане на само една декларация на ред.
* Ако последващите редове не се идентират автоматично, ги идентираме с една табулация навътре (четири шпации).
* Добавяне на точно един празен ред между дефиниции на методи и пропъртита, с цел логическо разделение.
* Добавяне на точно един празен ред между смислово отделни части в кода.
* Използване на кръгли скоби при сложни изрази, с цел по-ясно определяне приоритета на операциите:  
    
  

# 4. Конвенции при коментиране на кода

* Поставяне на коментарите на отделен ред, а не на края на реда на който е написан кода. За предпочитане е коментарите да се слагат над реда, за който се отнасят.
* Всеки коментар, трябва да завършва с точка.
* Всеки текст трябва да започва с една шпация отстояние от разделителя за коментар, както е показано в следващия пример:
* Не трябва да имаме изкуствено създадени блокове от разделителни символи покрай коментарите.   
    
  
* Коментари трябва да се пишат в краен случай, когато се описва сложен алгоритъм. Ако описваме проста поредица от действия, тези действия трябва да бъдат изнесени в отделни методи с достатъчно описателни имена, които да заличат нуждата от коментари

# 5. Насоки при писане на C# код

5.1. Работа със символни низове

* Използване на оператора за събиране „+“ за контатенация на кратки символни низове:  
  
* За конкатенация на символни низове, при работа с големи обеми от текст или много итерации, се използва типа StringBuilder:  
  

## 5.2. Имплицитно типизирани променливи

* Използване на имплицитна типизация за локални променливи, когато типа данни с който работим е очевиден от дясната част на оператора за присвояване или когато типа на променливата не е от значение:  
  
* Не използвайте имплицитна типизация, когато типа не е ясен от дясната част на оператора за присвояване, например когато извикваме метод и от името му не става ясно какъв тип данни връща:  
  
* Не разчитайте на името на променливата да укаже типа й. Може да е е грешен:  
  
* Избягвайте употребата на ключовата дума “var” вместо „dynamic”, двете работят по тотално различен начин. “dynamic” позволява смяна на типа на променливата по време на изпълнение на програмата, а „var” – не.
* Използвайте имплицитна типизация за определяне типа на “loop” променливата във „for” и “foreach” цикли:  
  

## 5.3. Масиви

* Използвайте опростения синтаксис при инициализиране на масиви на реда в който е декларирана променливата:

## 5.4. Прихващане на изключения

* Използване на “try-catch” блок за прихващане на изключенията:  
  
* Можем да опростим използването на обекти, които имплементират IDisposable интерфейса, като ги викаме в „using” блок по следния начин:  
  

## 5.5. Операторите „&&” и „||”

* За да избегнем хвърляне на изключения, и да подобрим производителността на нашия код, като прескочим ненужни сравнения и операции, трябва да използваме оператора „&&” вместо “&” и „||“ вместо „|”:  
  

## 5.6. LINQ заявки

* Използване на смислени и значещи имена за променливите, които съдържат резултат от LINQ заявка:  
  