**Требования к оформлению кода на C# для студентов**

Этот документ описывает стандарты написания кода на C#, которые должны соблюдаться студентами для обеспечения читаемости, единообразия и удобства сопровождения программ.

Оглавление

[**Требования к оформлению кода на C# для студентов** 1](#_Toc200103429)

[**1. Общие требования к оформлению кода** 2](#_Toc200103430)

[**1.1. Именование переменных, методов, классов** 2](#_Toc200103431)

[**1.2. Отступы и пробелы** 2](#_Toc200103432)

[**1.3. Максимальная длина строки** 3](#_Toc200103433)

[**2. Комментирование кода** 4](#_Toc200103434)

[**2.1. XML-документация для публичных методов и классов** 4](#_Toc200103435)

[**2.2. Однострочные комментарии** 4](#_Toc200103436)

[**2.3. Запрещены** 4](#_Toc200103437)

[**3. Структура программы** 5](#_Toc200103438)

[**3.1. Порядок объявлений в классе** 5](#_Toc200103439)

[**3.2. Разделение логических блоков** 5](#_Toc200103440)

[**4. Обработка ошибок и валидация** 6](#_Toc200103441)

[**5. Запрещённые практики** 6](#_Toc200103442)

[**6. Пример хорошо оформленного кода** 7](#_Toc200103443)

[**Заключение** 8](#_Toc200103444)

**1. Общие требования к оформлению кода**

**1.1. Именование переменных, методов, классов**

* **Классы, методы, свойства, пространства имён** — PascalCase.

class StudentManager

{

public void CalculateAverageGrade()

{

*// ...*

}

}

* **Локальные переменные и параметры методов** — camelCase.

int studentCount;

void PrintStudent(string studentName)

{

*// ...*

}

* **Константы и статические поля** — UPPER\_CASE с разделителем \_.

public const int MAX\_STUDENTS = 100;

private static readonly string DEFAULT\_NAME = "Unknown";

* **Поля класса (не статические)** — \_camelCase с префиксом \_.

private int \_studentId;

* **Интерфейсы** — начинаются с I.

interface IPrintable

{

*// ...*

}

**1.2. Отступы и пробелы**

* **Отступы** — 4 пробела (не табы). Ставятся автоматически в современных IDE.
* **Пробелы вокруг операторов** (=, +, -, && и т. д.).

int sum = a + b;

if (x > 0 && y < 10)

{

*// ...*

}

* **Пробел после запятой** в списке параметров.

void Print(string name, int age)

{

*// ...*

}

* **Фигурные скобки** { } — открывающая скобка **на новой строке**, закрывающая — на уровне начала блока.

if (condition)

{

*// ...*

}

**1.3. Максимальная длина строки**

* **Не более 120 символов** в строке. Если строка слишком длинная, её следует разбить.

**2. Комментирование кода**

**2.1. XML-документация для публичных методов и классов**

*/// <summary>*

*/// Вычисляет средний балл студента.*

*/// </summary>*

*/// <param name="grades">Массив оценок.</param>*

*/// <returns>Средний балл.</returns>*

public double CalculateAverageGrade(double[] grades)

{

*// ...*

}

**2.2. Однострочные комментарии**

* Используются только для пояснения сложных мест.
* Размещаются **над кодом** (предпочтительно) или в конце строки (если кратко).

*// Проверка на отрицательное значение*

if (value < 0)

{

throw new ArgumentException("Value cannot be negative.");

}

int x = 10; *// Коэффициент масштабирования*

**2.3. Запрещены**

* Закомментированный код (удаляйте ненужные фрагменты).
* Избыточные комментарии (очевидный код не комментируется).

**3. Структура программы**

**3.1. Порядок объявлений в классе**

1. Поля класса (private, затем public).
2. Конструкторы.
3. Свойства (public, затем private).
4. Методы (сначала public, потом private).

Пример:

class Student

{

private int \_id;

public string Name { get; set; }

public Student(int id)

{

\_id = id;

}

public void PrintInfo()

{

*// ...*

}

private void Validate()

{

*// ...*

}

}

**3.2. Разделение логических блоков**

* Между методами — **1 пустая строка**.
* Внутри метода — пустые строки только между логическими блоками (не более одной подряд).

**4. Обработка ошибок и валидация**

* **Проверка входных параметров** в публичных методах.

public void SetAge(int age)

{

if (age < 0)

{

throw new ArgumentException("Age cannot be negative.");

}

\_age = age;

}

* Использование try-catch только там, где ошибка может быть обработана.

**5. Запрещённые практики**

* **Магические числа** (используйте константы).

*// Плохо:*

if (age > 120) { ... }

*// Хорошо:*

const int MAX\_HUMAN\_AGE = 120;

if (age > MAX\_HUMAN\_AGE) { ... }

* **Слишком длинные методы** (более 30 строк — повод для разбивки).
* **Повторяющийся код** (выносите в отдельные методы).
* **Неиспользуемые переменные, методы, using-директивы**.

**6. Пример хорошо оформленного кода**

using System;

namespace StudentManagement

{

*/// <summary>*

*/// Представляет студента с основными данными.*

*/// </summary>*

public class Student

{

private int \_id;

private string \_name;

public Student(int id, string name)

{

if (id <= 0)

{

throw new ArgumentException("ID must be positive.");

}

if (string.IsNullOrEmpty(name))

{

throw new ArgumentException("Name cannot be empty.");

}

\_id = id;

\_name = name;

}

public string Name

{

get => \_name;

set => \_name = value ?? throw new ArgumentNullException(nameof(value));

}

*/// <summary>*

*/// Выводит информацию о студенте.*

*/// </summary>*

public void PrintInfo()

{

Console.WriteLine($"ID: {\_id}, Name: {\_name}");

}

}

}

**Заключение**

Код должен быть:

* **Аккуратным** (без лишних пустых строк, ненужных комментариев, неиспользуемого кода).
* **Читаемым** (правильные отступы, осмысленные имена переменных).
* **Структурированным** (логичное разделение на методы и классы).

Несоблюдение требований приведёт к снижению оценки.