Соглашение о кодировании

Содержание

[1 Вступление 3](#_Toc483823599)

[2 C# 3](#_Toc483823600)

[2.1 Общие правила 3](#_Toc483823601)

[2.2 Интерфейсы 4](#_Toc483823602)

[2.2.1 Наименование 4](#_Toc483823603)

[2.2.2 Общие рекомендации 4](#_Toc483823604)

[2.3 Классы 4](#_Toc483823605)

[2.3.1 Наименование 4](#_Toc483823606)

[2.3.2 Порядок размещения разделов в классе 4](#_Toc483823607)

[2.3.3 Поля 5](#_Toc483823608)

[2.3.4 Свойства 5](#_Toc483823609)

[2.3.5 Методы 5](#_Toc483823610)

[2.3.6 Общие рекомендации 6](#_Toc483823611)

[2.4 Перечисления 6](#_Toc483823612)

[2.5 Параметры обобщений 6](#_Toc483823613)

[2.6 Разбиение кода 6](#_Toc483823614)

[2.7 Оформление 6](#_Toc483823615)

[2.7.1 Комментарии 7](#_Toc483823616)

[2.8 Уточнения, разъяснения и комментарии к правилам 8](#_Toc483823617)

[2.8.1 Выбор идентификаторов 8](#_Toc483823618)

[2.8.2 Рекомендации по образованию имен 8](#_Toc483823619)

[2.8.3 Generic-типы 8](#_Toc483823620)

[2.8.4 Коллекции 9](#_Toc483823621)

[2.8.5 Перечисления 9](#_Toc483823622)

[2.8.6 Строки 9](#_Toc483823623)

[2.8.7 Исключения 9](#_Toc483823624)

[2.8.8 Основные принципы проектирования 10](#_Toc483823625)

[2.8.9 Стандартные приемы и рекомендации 10](#_Toc483823626)

[2.8.10 Опасные приемы 11](#_Toc483823627)

[2.9 Технологии 11](#_Toc483823628)

[2.9.1 ASP.NET MVC 11](#_Toc483823629)

[3 JavaScript 12](#_Toc483823630)

[3.1 Наименовае фалов 12](#_Toc483823631)

[4 HTML 12](#_Toc483823632)

# Вступление

Этот документ описывает правила и рекомендации по созданию кода на языке C#.

Помните! Код чаще читается, чем пишется, поэтому не экономьте на понятном, удобном и чистом коде в угоду скорости набора.

Соответствие создаваемого кода принципам и правилам, описанным в данном документе, является обязательным.

Соблюдение стандартов кодирования способствует созданию более качественного и унифицированного кода, лучшему восприятию кода, а также облегчает коллективную разработку.

*Написать код, понятный компьютеру, может каждый, но только хорошие программисты пишут код, понятный людям.* (с) М. Фаулер.

# C#

## Общие правила

1. Для всех public методов, классов и свойств необходимо указывать комментарии.
2. Ни интерфейсы, ни классы, ни объекты - ничего не может содержать название заказчика (МЕД, ЕЕК, АПРФ, и т.д.). Исключение – наименования проектов и папок.
3. Не используйте венгерскую нотацию.
4. Не используйте подчёркивание внутри идентификаторов.
5. Второе и все последующие слова в имени всегда пишутся с заглавной буквы (camel case).
6. Не используйте цифры в именах идентификаторов.
7. Не используйте сокращения и аббревиатуры, за исключением общепринятых. (Например: Xml, UI)
8. Если имеется идентификатор, являющийся аббревиатурой, длиной менее трёх букв, записывайте его всеми заглавными буквами [System.IO, System.Web.UI]. Имена длиннее двух букв записывайте в следующем виде Guid, Xml, XmlDocument.
9. Когда придумываете название для нового, открытого (public) класса, пространства имён (namespace) или интерфейса, старайтесь не использовать имена, потенциально или явно конфликтующие со стандартными идентификаторами.
10. Namespace каждого класса (интерфейса, перечисления и т.д.) должен полностью отражать структуру проекта.
11. Всегда явно указывайте вид доступа к члену класса и к классу (интерфейсу, перечислению и т.д.)
12. Добавляйте суффикс EventHandler для делегатов событий.
13. Используйте префикс On для обработчиков событий.
14. Не используйте префиксы Before и After для имен событий. Используйте для этого настоящее и прошедшее время. Например: WindowClosing, PropertyChanged.
15. Не используйте обобщенные имена, например - Element, Node, Log, Message и т.д. Всегда уточняйте: FormElement, XmlNode, EventLog, SoapMessage.
16. Не используйте префикс Ex чтобы показать, что метод или интерфейс является новой версией другого метода или интерфейса.
17. Наименование структур должно начинаться с заглавной буквы (пример: struct Point)
18. Наименование констант должно начинаться с заглавной буквы (пример: public const int Count)

## Интерфейсы

### Наименование

Наименование интерфейса должно начинаться с «I», далее с заглавной буквы (пример: IDataContext)

### Общие рекомендации

1. Избегайте объявления пустых интерфейсов. Предназначение интерфейса – определение некоего контракта для реализующих этот интерфейс классов.
2. Помните о разнице между явным и неявным определением реализации методов интерфейса в классе. Неявное не позволит реализовать несколько интерфейсов с методами с одинаковой сигнатурой.
3. При реализации стандартных интерфейсов, реализуйте их Generic версии, если они есть.

## Классы

### Наименование

1. Имена классом должны начинаться с заглавной буквы.
2. Не используйте префикс C для имен классов.
3. Имя класса не должно совпадать с именем пространства имен (namespace).

#### **Класс исключения**

С заглавной буквы, заканчивается на Exception (пример: EntityUsageException)

#### **Класс атрибута**

С заглавной буквы, заканчивается на Attribute (пример: ColumnAttribute)

### Порядок размещения разделов в классе

1. Константы
2. Статические поля и свойства
3. Поля
4. Закрытые свойства
5. Защищенные свойства
6. Открытые свойства
7. События
8. Конструкторы
9. Открытые методы
10. Защищенные методы
11. Закрытые методы

### Поля

1. Название приватного поля начинается с “\_”, далее со строчной буквы (пример: \_user)
2. Название публичного статического поля только для чтения (пример: public static readonly object Instance) должно начинаться с заглавной буквы.
3. Объявляйте все поля приватными. Не используйте открытые\защищенные поля.
4. Инициализируйте поля в конструкторе или отдельном методе. Это позволяет видеть начальное состояние класса в одном месте.
5. Не используйте public, internal, protected поля, вместо них следует использовать свойства.

### Свойства

1. Наименование свойства должно начинаться с заглавной буквы (пример: public string FirstName {get; set;})
2. Свойство не должно менять своего значения от вызова к вызову, если состояние объекта не изменяется. Если результат при новом вызове может быть другим при том же состоянии объекта, используйте метод. Например, функционал занимающийся работой с данными вне класса, в котором реализован, должен быть оформлен в виде метода, а не свойства.
3. Не используйте свойства «только для записи». Потребность в таком свойстве может быть признаком плохого проектирования.
4. Если свойство должно быть не неизменяемым или вычисляемым то используйте свойство только для чтения.

### Методы

1. Наименование метода должно начинаться с заглавной буквы (пример: public List<User> GetUserList())
2. Наименование локальных переменных и параметров метода должно начинаться со строчной буквы
3. Не используйте GetX() SetX() методы.
4. Не усложняйте прототип метода «зарезервированными» параметрами, которые, возможно, будут использоваться в будущих версиях реализации. Если в будущем понадобится новый параметр, используйте перегрузку методов.
5. Если метод возвращает или принимает тип, наследуемый от интерфейса или другого базового типа, то по возможности старайтесь возвращать или принимать интерфейс или базовый тип, если только вам действительно не нужно работать с конкретным типом. Это уменьшает связность кода и облегчает его повторное использование.
6. Заменяйте список повторяющихся параметров метода массивом params.
7. Не скрывайте методы базового класса в классах наследниках, перекрывайте только виртуальные методы.

### Общие рекомендации

1. Не заводите чрезвычайно общие классы. Разработать удачный класс с достаточно общей семантикой сложно даже для опытного разработчика. Общие классы обычно имеют сложную, плохо определенную семантику и крайне смутные правила использования.
2. Абстрактные классы не должны иметь публичных конструкторов, поскольку вы всё равно не можете создать экземпляр такого класса. Используйте защищённый конструктор.
3. Окончательные (sealed) классы не должны иметь ни защищённых, ни виртуальных методов. Такие методы используются в производных классах, а окончательный класс не может быть базовым.
4. Классы, определяющие только статические методы и свойства, должны быть объявлены как static.
5. Свойства и методы, возвращающие строку, список или коллекцию не должны возвращать null, возвращайте пустой список и пустую коллекцию.
6. Используйте опциональные параметры для уменьшения числа перегрузок метода

## Перечисления

1. Наименование перечисления начинаются с заглавной буквы (пример: public enum StorageType)
2. Именуются в единственном числе, за исключением перечислений являющихся битовыми полями (и помеченными атрибутом Flags).
3. Не используйте суффикс Enum для имен перечислений
4. Не используйте суффикс Flags для имен перечислений
5. Наименование значения перечисления начинаются с заглавной буквы (пример: Disk, Web)

## Параметры обобщений

1. Наименование перечисления начинаются с «T», далее заглавной буквы (пример: T, TKey, TValue)
2. Для параметров обобщений (generics) не используйте однобуквенные имена, за исключением случая одного параметра.

## Разбиение кода

1. Количество слов в идентификаторе не должно превышать 5.
2. Запрещается применять partial классы. Исключением является расширение автогенерированных классов.
3. Один файл должен содержать только один класс, интерфейс или перечисление. Исключения составляют обобщенные (шаблонные, Generic) классы и интерфейсы. Пример: IQuery и IQuery<T> должны быть объявлены в одном файле.
4. Файл должен называться так же как и класс который в нем находится. Пример: interface IQuery – файл должен называться IQuery.cs.

## Оформление

1. Текст выравнивается с помощью пробелов.
2. Во всех случаях без исключения после запятой ставится пробел, перед запятой – никогда.
3. После открывающей круглой скобки и перед закрывающей скобкой пробелы не ставятся.
4. После "[" и перед "]" пробелы не ставятся.
5. Бинарные операции с двух сторон окружаются пробелами.
6. Унарные операции пишутся слитно с операндом.
7. операторы " ? :" и “??” пишутся с пробелами вокруг "?", ":" или “??”
8. Открывающая и закрывающая фигурные скобки всегда располагаются на новой строке. Исключением являются свойства
9. Внутренние блоки if, else, for, foreach, using всегда помещаются в фигурные скобки. Не следует опускать скобки, т.к. это потенциальный источник сделать ошибку при последующем редактировании. Допускается исключение, для проверки входных параметров метода при условии бросания Exception’а.
10. Избегайте вложенности операций при ветвлении, данный код тяжело поддерживать и модифицировать. Используйте операторы return или continue, break.
11. Используйте одну пустую строку между методами и свойствами.
12. Не используйте 2 и более пустые строки.
13. Никогда не используйте пустые строки после открывающейся фигурной скобки и перед закрывающейся фигурной скобкой.
14. Содержимое класса, интерфейса, метода, перечисления должно быть сдвинуто на одну табуляцию относительно объявления.
15. Используйте алиасы для имен стандартных типов. int вместо Int32, long вместо Int64, и т.д.
16. Не используйте оператор goto
17. При использовании операторов сравнения, пишите константную часть выражения с правой стороны [например if (user == null)]
18. Не ставьте пробел между именем метода и открывающей скобкой
19. Всегда ставьте пробел после if, for, foreach, while и открывающей круглой скобкой
20. Не используйте лишние скобки (например: return (a + b);)
21. Отделяйте пустой строкой блок using-директив и объявление namespace
22. Размещайте using-директивы вне объявления пространства имен

### Комментарии

1. Комментарии пишутся непосредственно перед комментируемой частью на отдельной строке или строках.
2. Комментарии пишутся на русском языке.
3. Комментарии должны объяснять причины выбора конкретного решения и любые другие значительные моменты, не вытекающие из собственно кода.
4. Комментарии не должны повторять код или имена методов и классов
5. Для комментирования классов, интерфейсов, структур, перечислений и public методов должны использоваться комментарии в стиле Microsoft – “///”.
6. Все остальные комментарии должны начинаться с двух слешей – “//”.
7. Всегда начинайте текст комментария с пробела (например: // Комментарий).
8. Не используйте комментарии для выделения блоков кода.
9. Запрещается комментировать код который не нужен или устарел - он должен быть удален.

## Уточнения, разъяснения и комментарии к правилам

### Выбор идентификаторов

Основное требование к выбору идентификаторов - понятность. Следует избегать непонятных сокращений, бессмысленных или однобуквенных идентификаторов (кроме общепринятых переменных цикла i, j, k , для координат - x, y, z). Кроме того, на выбор идентификатора налагается ряд формальных и неформальных ограничений.

По идентификатору объекта или метода должно быть понятно, для чего объект или метод предназначен. Это означает, что идентификаторы должны быть осмысленными выражениями. При этом помните, что слишком длинные идентификаторы загромождают выражения. Обычно идентификатор должен быть не длиннее 15 символов. При этом более простым и часто используемым объектам и методам следует давать более короткие идентификаторы, более сложным и редко используемым объектам и методам – более длинные, но более понятные.

### Рекомендации по образованию имен

1. Bool

Для булевых идентификаторов имя, в большинстве случаев, должно начинаться с приставки "is" или “has”. Например: IsOpen, HasValue.

Имя метода получения бинарного атрибута должно быть предикатом (вопросом, на который можно ответить «Да» или «Нет»). В большинстве случаев имя метода должно начинаться с глаголов Is или Has.

1. Количество элементов

Если переменная служит для хранения количества элементов в каком-либо списке или массиве, то к названию такой переменной следует добавлять приставку ‘numberOf’ или суффикс ‘count’. Например, переменная, указывающая количество свободных блоков в менеджере памяти, может называться ‘numberOfFreeBlocks’ или ‘freeBlocksCount’.

1. Размер/длина

Слово "Size" применяется для переменных и методов, задающих размер (число байт).

Слово"Length" используется в имени переменных и методов, служащих для задания числа элементов.

Замечание: Слово "Size" - это размер, а "Length" - длина. Не следует размер блока памяти называть blockLength, а длину строки - stringSize.

1. Методы

Методам, изменяющим объект либо внешние по отношению к объекту данные, даются имена, начинающиеся с глагола, обозначающего это действие. Например, FilterDust(), BuildTree() и т. п. Старайтесь избегать очень общих глаголов - Do, Make, Process.

### Generic-типы

1. При работе со стандартной библиотекой всегда используйте generic версию типа, если она существует. Например, List<T> вместо ArrayList.
2. Используйте generic constraints везде где возможно.

### Коллекции

1. Если метод должен вернуть коллекцию, то старайтесь не возвращать конкретную реализацию (например, List<T>), возвращайте интерфейс IEnumerable<T>, IReadOnlyCollection<T>, IReadOnlyList<T>, IReadOnlyDictionary<TKey, TValue>, если коллекция предполагает изменения, то ICollection<T>, IList<T> >, IDictionary< TKey, TValue >. Внешний код, по возможности, не должен знать о конкретной реализации вашей коллекции, лишнее знание увеличивает связность и затрудняет дальнейшую поддержку кода.
2. Если код принимает коллекцию, то старайтесь принимать наиболее абстрактный тип коллекции. Если элементы коллекции возможно только читать, то используйте IEnumerable<T>, IReadOnlyCollection<T>, IReadOnlyList<T>, IReadOnlyDictionary<TKey, TValue>, если коллекция может быть модифицирована, то используйте ICollection<T>, IList<T>, IDictionary< TKey, TValue >.
3. Для обхода элементов коллекций, по возможности пользуйтесь итераторами (foreach), а не циклом for.

#### Общие рекомнедации

При выборе в качестве параметра или результата тип IEnumerable<T>, помните, что это однопроходная последовательность и человек который будет с ней работать должен использовать её единожды. При проектировании руководствуйтесь следующими правилами:

* 1. Если последовательность можно использовать единожды возвращайте\запрашивайте IEnumerable<T>;
  2. Если последовательность нужно использовать многократно возвращайте\запрашивайте более конкретную реализацию;

### Перечисления

1. Старайтесь использовать Int32 для типа enum. Не используйте не CRL-совместимых типов в качестве базовых для enum типов.
2. Enum-ы должны иметь значение для 0, так как без инициализации, как value тип, enum равен 0.

### Строки

1. Помните, что строка является immutable типом и любое изменение строки приводит к созданию новой строки.
2. Для активной модификации строк используйте класс StringBuilder, он эффективнее, так как рассчитан именно на такой сценарий.
3. Для выяснения пустая ли строка используйте статический метод string.IsNullOrEmpty.
4. Для работы с Интернет адресами и адресами в локальной сети, используйте специальный класс Uri, а не строки.

### Исключения

1. Старайтесь не перехватывать базового исключения, перехватывайте конкретный тип в иерархии исключений, на который точно знаете, как реагировать.
2. Если после обработки исключения необходимо выбросить это же исключение снова, используйте throw без параметров. Это позволит сохранить стек вызовов для других обработчиков.
3. Исключение, унаследованное напрямую от системных исключений, должно быть публичным. Если исключение является внутренним для модуля и, будучи выброшено в нем перехватится за пределами модуля, то будет доступно только как базовое исключение, при этом потеряется важная информация о том, что произошло.
4. Создавая свой класс исключения, переопределяйте все конструкторы класса Exception. В противном случае некоторые сценарии использования исключений будут недоступны.
5. Не наследуйте свой тип исключения от ApplicationException.
6. Передавайте дополнительную информацию в стандартных исключениях (ArgumentException, ArgumentNullException, ArgumentOutOfRangeException, DuplicateWaitObjectException)
7. Старайтесь обрабатывать только известные вам исключения. Если вы все же обрабатываете все исключения, то проведя необходимую обработку, генерируйте исключение повторно, чтобы его могли обработать последующие фильтры, или выводите пользователю максимально полную информацию об ошибке.
8. Если целью перехвата исключений является очистка ресурсов после сбоя, используйте секцию finally.
9. Пишите в логи Exception.ToString(), а не Exception.Message.
10. Классы исключений должны быть помечены атрибутом [Serializable]
11. Не ловите фатальные исключения, все равно приложение не может продолжать корректную работу.

### Основные принципы проектирования

1. KISS (Keep It Simple, Stupid). Все должно быть сделано настолько простым, насколько это возможно, но не проще (А. Эйнштейн)
2. DRY (Don’t Repeat Yourself). Не повторяйте свою и чужую функциональность.
3. YAGNI (You Aren’t Going Need It). Процесс и принцип проектирования, при котором в качестве основной цели и/или ценности декларируется отказ от добавления функциональности, в которой нет непосредственной нужды.
4. ООП принципы SOLID
5. **S**RP (Single Responsibility Principle). Не должно существовать более одного мотива для изменения данного класса.
6. **O**CP (Open/Close Principle). Классы и методы должны быть открыты для расширения, но закрыты для модификации.
7. **L**SP (Liskov Substitution Principle). Функции, которые используют ссылки на базовые классы, должны иметь возможность использовать объекты производных классов, не зная об этом
8. **I**SP (Interface Segregation Principle). Клиент не должен вынужденно зависеть от элементов интерфейса, которые он не использует.
9. **D**IP (Dependency Inversion Principle). Модули верхних уровней не должны зависеть от модулей нижних уровней. Оба типа модулей должны зависеть от абстракций. Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций.

### Стандартные приемы и рекомендации

1. Используйте блок using для типов, реализующих IDisposable.
2. Используйте оператор ?? вместо ?, : для проверки на null .
3. Используйте лямбда выражения для написания простых анонимных методов
4. Используйте var при инициализации переменных. Использование имени типа вместо var допускается там, где имя типа не присутствует в правой части выражения.
5. Используйте автоматические свойства (autoproperty) там, где не требуется дополнительная логика
6. Используйте switch…case вместо цепных условий. Цепные условия применяются только в сложных случаях, когда использование switch…case невозможно. В сложных случая исследуйте возможность реализацией логики условий паттерном, например Chain или State/Strategy.
7. Используйте стандартные константы: string.Empty вместо “”; Environment.NewLine вместо “\r\n” и т.д.
8. Старайтесь не делать методы с более чем 4мя параметрами.
9. Старайтесь не делать анонимные методы длиннее 2-3 строк.
10. Бросайте исключения вместо возвращения кода ошибки.
11. Текст исключения должен быть понятным и достаточным для идентификации проблемы.
12. Не передавайте null в качестве sender параметра событию.
13. Класс реализующий интерфейс IDisposable должен бросать исключение ObjectDisposedException при обращении к его public функционалу (методам и свойствам), после того как был вызван метод Dispose.
14. При возвращении из метода IEnumerable<T> пустого значения используйте Enumerable.Empty<T>().
15. При получении элемента по уникальному ключу из последовательности, необходимо использовать SingleOrDefault, а не FirstOrDefault, так как устранение логической ошибки очень трудозатратно.

### Опасные приемы

1. Не используйте магические числа, зашитые в код размеры буферов, времена ожидания и т.п. Лучше определите константу (если вы никогда не будете её менять), либо вынести значение в конфигурационный файл.
2. Не храните строковые константы, в частности пути, непосредственно в коде, получайте их из ресурсов, конфигурационных файлов или с помощью системных функций.
3. Если метод меняет состояние переданного в него объекта, то результатом работы этого метода должен быть измененный объект.
4. Использование оператора “as” для приведения типов возможно только в том случае, если после него идет обязательная проверка на null.
5. Не обращайтесь явно к методам сборщика мусора, за исключением специальных случаев
6. Для освобождения unmanaged ресурсов используйте паттерн Disposable
7. Старайтесь освобождать unmanaged ресурсы явно, с использование конструкции using, не полагаясь на сборщик мусора (когда у класса реализован паттерн Disposable).
8. Старайтесь не вызывать виртуальные методы из конструктора.
9. Не возвращайте из метода bool в знак успешности работы метода. Например, метод открытия файла должен в случае ошибки бросить исключение, а не возвращать false (именно потому, что его почти наверняка никто не проверит).

## Технологии

### ASP.NET MVC

1. Помечать атрибутами [HttpPost] и [HttpGet] открытые методы контроллеров (Actions).

# JavaScript

## Наименовае фалов

Если название файла должно состоять из 2х или более слов, то разделять слова следует через «-» и только из строчных символов. Например: general-info.js, order-transport.js

## Наименовае фалов

Минифицированные файлы руками не править. Если правится файл, для которого существует минифицированный файл, необходимо произвести минификацию заново.

# HTML

Все теги должны быть закрыты