Стандарт кодирования

Язык С++, платформа .NET

1. ОБЩЕЕ

Данный стандарт определяет соглашения по разработке приложений на языке C++ для платформы .NET Framework. Следование данному стандарту является обязательным для всех проектов и всех видов учебной работы, выполняемых с использованием C++/.NET на кафедре ФН1-КФ.

1. СОГЛАШЕНИЯ ПО ИМЕНОВАНИЮ

При написании кода очень важно обеспечить ясность кода и легкость его чтения. В идеале код должен быть написан так, чтобы он читался как предложения на английском языке, и для того, чтобы понять фрагмент кода не приходилось бы заглядывать в определения участвующих в нем классов, методов и т. п.

Существенную роль в достижении этих целей является правильное именование программных элементов: типов, полей, методов, переменных.

* 1. **Общие правила именования**

1. В именах идентификаторов используйте американский английский. Строго не допускается использование транслита. При выборе имен не ленитесь заглядывать в словарь!

**+** optimizing, realize, behavior,…

**-** colour (британский вариант), spisokPolzovatelej (транслит)

1. При именовании идентификаторов используйте Upper Camel casing или Lower Camel casing.

Camel casing предполагает, что отдельные слова в идентификаторе, состоящем из нескольких слов, следуют подряд и разделяются путем написания каждого следующего слова с прописной буквы.

**+** property\_descriptor (используется знак подчеркивания), backcolor (слова не разделяются), HTML\_TAG (используются прописные буквы, разделитель — знак подчеркивания)

**-** PropertyDescriptor, BackColor, HtmlTag

В случае Upper Camel case первое слово в идентификаторе также начинается с прописной буквы. В случае Lower Camel case первое слово начинается со строчной буквы.

**+** *Upper Camel case*: PropertyDescriptor, HtmlTag, BackColor

**+** *Lower Camel case*: propertyDescriptor, htmlTag, backColor

Какой вариант начертания использовать (Upper Camel или Lower Camel) зависит от вида идентификатора. Конкретные правила указаны ниже в соответствующих разделах.

1. Акронимы (слова, образованные из начальных букв слов или словосочетаний) представляются следующим образом. Акронимы, состоящие из трех или больше символов подчиняются рекомендациям для обычных слов:

+ ProcessHtmlTag(string htmlTag)

- ProcessHTMLTag

Акроним, состоящий из двух символов, в Upper Camel Case набирается двумя прописными символами, а в Lower Camel Case — двумя строчными.

+ *Upper Camel case*: IOStream, StartIO, UI

+ *Lower Camel case*: ioStream, ui

**Исключения**: Id, Ok — часто используемые обозначения, набираются как здесь указано.

1. Составные слова, в отличие от словосочетаний, должны быть набраны как одно слово (**примеры:** Callback, Endpoint, Email, Gridline, Hashtable, Metadata, Multipanel, Multiview, Namespace, Placeholder; **ср.:** BitFlag, FileName, UserName).
2. Не следует вводить идентификаторы, отличающиеся только регистром. Идентификаторы следует давать так, чтобы они могли использоваться в языках, не являющихся чувствительными к регистру.

- Не допускается, например, наличие в одном классе методов Abs и abs.

1. В идентификаторах недопустимы символы подчеркивания, дефисы и другие неалфавитно-цифровые символы.

**Исключение**: код, автоматически сгенерированный средой (например, обработчики событий).

1. В идентификаторах недопустима венгерская нотация.

iCount, szUserName, hWnd

count, userName, windowHandle

1. Не следует использовать ключевые слова в качестве идентификаторов.
2. Важно, чтобы все идентификаторы были легко читаемыми (и произносимыми вслух) и понятными, а не максимально короткими. Поэтому недопустимым является использование сокращений, а использования акронимов следует избегать. При необходимости допускается использование только общепринятых акронимов (UI, XML, HTTP и т. п.) или акронимов, распространенных в предметной области, для которой разрабатывается приложение (например, BOE — Bank of England, VAT — Value Added Tax).

ScrollableX (непонятное название)

CanScrollHorizontally (смысл гораздо легче понять)

GetWin(IntPtr hWnd), Acnt, Rcpt, Log4Emulator, RcvTsk

GetWindow(IntPtr windowHandle), Account, Receipt, LogForEmulator, ReceivingTask



class DtaRcrd102

{

private DateTime \_genymdhms;

private DateTime \_modymdhms;

private readonly string \_pszqint = "102";

}



class Customer

{

private DateTime \_generationTimestamp;

private DateTime \_modificationTimestamp;

private readonly string \_recordId = "102";

}

**Исключения:**

Целочисленные счетчики цикла по традиции именуют i, j, k. При наборе коротких идентификаторов не следует использовать символы, которые можно принять за цифры (например: «оу большое» O, «ай большое» I, «эл маленькое» l).

Для параметров лямбда-выражений принято использовать короткие имена:

users.Where(u => u.LogonName == name).Select(u => u.Id)

1. При выборе имен не следует проявлять остроумие, использовать просторечия, сленг, элементы конкретной культуры (смысл таких идентификаторов будет понятен только людям, разделяющим чувство юмора автора, да и то, только если они помнят шутку).

HolyHandGrenade, Whack, EatMyShorts

DeleteItems, Kill, Abort

1. В идентификаторах, при необходимости, следует использовать названия алгоритмов, паттернов, термины математики и информатики и т. п. Также следует использовать префиксы и суффиксы, обозначающие роль и место типа в архитектуре или дизайне приложения.

QuickSort, BinarySearch (алгоритмы)

PasswordHash, PasswordSalt (термины)

AccountVisitor, ISessionFactory, MoveStockCommand (паттерны проектирования)

IStockInquiryPresenter, IFlightsDao, FundsService, EmployeesController (архитектурная роль)

1. **. Именованные константы (включая значения перечислений) должны быть записаны в верхнем регистре с нижним подчёркиванием в качестве разделителя.  
     
    MAX\_ITERATIONS, COLOR\_RED, PI**
2. **Общая практика в сообществе разработчиков C++. Использование таких констант должно быть сведено к минимуму. В большинстве случаев реализация значения в виде метода — лучшее решение:**

**int getMaxIterations() *// НЕЛЬЗЯ: MAX\_ITERATIONS = 25***

**{**

**return 25;**

**}**

Эта форма более читаемая и гарантирует единый интерфейс к значениям, хранящимся в классе.

* 1. **Именование пространств имен**

1. Идентификаторы пространств имен набираются в стиле Upper Camel case, а компоненты идентификатора отделяются точками.

System.Data.SqlClient, Spring.Context.Events, Hecsit.Caduceus.Services

1. Идентификатор пространства имен следует задавать в соответствии с одним из следующих шаблонов:

Company.Product.Feature.Subnamespace1.Subnamespace2.<...>

Company.Technology.Feature.Subnamespace1.Subnamespace2.<...>

В рамках учебных заданий и проектов кафедры рекомендуются следующие шаблоны:

для лабораторных работ — **Hecsit**.*Дисциплина*.**Lab***Номер*.<...>

для курсовых работ/проектов — **Hecsit**.*Дисциплина*.*ИмяПроекта*.<...> или **Hecsit**.*ИмяПроекта*.<...>

для дипломных и прочих проектов — **Hecsit**.*ИмяПроекта*.<...>

Т. о., в качестве префикса рекомендуется выбирать название компании, а во втором уровне — устойчивое, независящее от версии название продукта или проекта.

При необходимости следует использовать множественное число.

Имя «корневого» пространства имен для проекта в Visual Studio можно задать так: в контекстном меню проекта выбрать «Properties» и на открывшейся странице свойств проекта на вкладке «Application» в текстовом поле «Default Namespace» можно указать пространство имен.

*Hecsit.Caduceus.Services.CreditCard* Здесь *Hecsit* — префикс всех проектов кафедры, *Caduceus* — имя проекта, *Services.CreditCard* — иерархия пространств имен, отражающие логическую архитектуру приложения: классы слоя сервисов, относящиеся к использованию кредитных карт.

* 1. **Именование типов**

Имена классов являются наиболее важными, т. к. класс — это центральная концепция проектирования. Часто правильное имя является результатом последовательных упрощений и улучшений.

1. В идентификаторах классов и структур должен использоваться стиль Upper Camel case. Недопустимо прибавление к именам

классов/структур префиксов или суффиксов, указывающих, что это — имя класса/структуры (например, «C», «Class»).

CWindow, CInvoice, ContainerClass

Window, Invoice, Container

Название класса/структуры должно быть существительным или именной группой, т. к. они представляют объекты системы. Для именования важных классов лучше использовать одно слово. Старайтесь не использовать в именах классов такие «общие» слова как *Manager*, *Processor*, *Data* или *Info*.

Вообще говоря, если вы не можете придумать имя типа в соответствии с этими правилами, то следует заново продумать общий дизайн типа.

DeleteItems, EmployeeInfo

Payment, BusinessTransaction, TabOrder, LogOnScreen

1. В идентификаторах интерфейсов должен использоваться стиль Upper Camel case, при этом имя должно снабжаться префиксом «***I***». В качестве имен интерфейсов следует выбирать существительные и именные группы, а, если интерфейс представляет возможность, — прилагательные и адъективные фразы:

IComparable, IDisposable, IEnumerable (прилагательные, обозначают возможность)

ICollection, IList (описательные существительные)

ITransactionAuthorizationService (именная группа)

1. В именах абстрактных базовых классов можно использовать слова «*Abstract*» (как префикс) и «*Base*» (чаще как суффикс).

AbstractStock, ReportBase, BaseDao

1. Идентификатор производного класса должен указывать, на что класс похож и чем отличается. Рекомендуется имена производных классов оканчивать именами базовых классов.

DeferredCharge, OverdraftCharge, StorageCharge и т. п. (наследники класса Charge, представляют различные виды начислений)

**Исключение:** важно разумно применять это правило. Например, подклассы, находящиеся во главе собственной иерархии, могут иметь более короткое имя (т. е. без имени базового класса в качестве суффикса)

Handle — наследник класса Figure и корень собственной иерархии StretchyHandle, TransparencyHandle и проч.

Button — наследник класса Control (в данном случае имя ButtonControl не будет более ясным).

1. Существуют следующие правила именования классов по их роли в дизайне приложения (см. раздел 2.1, п. 11):

при именовании классов-исключений (*System.Exception* и его наследники) должен использоваться суффикс «*Exception*»;

при именовании классов, представляющих атрибуты (*System.Attribute* и его наследники), должен использоваться суффикс «*Attribute*»;

при именовании делегатов, представляющих методы обратного вызова, должен использоваться суффикс «*Callback*»;

при именовании делегатов, представляющих событие, должен использоваться суффикс «*EventHandler*»;

при именовании типа, представляющего параметры события (наследники *System.EventArgs*), должен использоваться суффикс «*EventArgs*».

EntityNotFoundException, HttpGetAttribute, ServiceRequestFailedCallback, ClickedEventHandler, MouseEventArgs.

1. В паре «интерфейс — класс», в которой класс является стандартной реализацией интерфейса, идентификаторы должны совпадать с точностью до префикса «***I***» у интерфейса. Для имени класса не допускается использование дополнительных префиксов и суффиксов типа «*Impl*», «*Default*».

 IMovementService — DefaultMovementService, MovementServiceImpl

 IMovementService — MovementService

1. Настоятельно не рекомендуется использовать одно и то же название для пространства имен и типа в этом пространстве имен.
2. В именах перечислений должен использоваться Upper Camel case. Идентификатор должен быть существительным или именной группой. Если значениями перечисления являются битовые флаги, то идентификатор должен быть во множественном числе, в остальных случаях — в единственном числе. Не допускается использование суффиксов «*Enum*», «*Flags*» и проч. В идентификаторах элементов перечисления недопустимы префиксы, обозначающие перечисление.

[Flags] enum ConsoleModifierFlags { ... }

[Flags] enum ConsoleModifiers

enum ConsoleColors { ... }

enum ConsoleColorEnum { ... }

enum ConsoleColor { ... }



enum ImageMode

{

ImageModeBitmap,

ImageModeGrayscale,

ImageModeIndexed,

ImageModeRgb

}



enum ImageMode

{

Bitmap,

Grayscale,

Indexed,

Rgb

}

1. **. Локальные типы, используемые в одном файле, должны быть объявлены только в нём.**  
     
   Улучшает сокрытие информации.
2. **Разделы класса**public**,** protected**и**private**должны быть отсортированы. Все разделы должны быть явно указаны.**  
     
   Сперва должен идти раздел *public*, что избавит желающих ознакомиться с классом от чтения разделов *protected/private*.
3. **Приведение типов должно быть явным. Никогда не полагайтесь на неявное приведение типов.**

floatValue = static\_cast<float>(intValue); *// НЕЛЬЗЯ: floatValue = intValue;*

Этим программист показывает, что ему известно о различии типов, что смешение сделано намеренно.

-------------------------------------------------------------------------------------

* 1. **Именование членов типа** 
     1. **Методы**

1. В идентификаторах методов должен использоваться стиль Upper Camel case. Т. к. методы представляют собой некоторые действия, идентификаторы методов должны быть глаголами или глагольными фразами. Имена должны прояснять обязанности метода и при этом быть максимально короткими.

StockMovement (именная группа)

MoveStock, GetRowsCount (глагольные фразы)

ComputeReportTotalsAndOpenOutputFile (слишком длинное имя и наличие союза and — признаки того, что методу поручено слишком много обязанностей, и он должен быть разбит на несколько методов)

DoDelete (лишнее слово «Do»)

Следует избегать глаголов, которые могут обозначать практически любое действие.

HandleCalculations, PerformServices, OutputUser, ProcessInput, DealWithOutput

1. Если из контекста понятно, над каким объектом выполняется действие, то название объекта не следует включать в идентификатор метода. Под контекстом здесь понимается объект, для которого определяется метод, типы и имена его параметров.

UsersRepository.GetUser(long id) (из контекста понятно, что возвращается именно User)

UsersRepository.Get(long id)

AdminService.DeleteUser(User user) (из контекста понятно, что удаляться будет именно User)

AdminService.Delete(User user)

AdminService.Delete(long id) (а здесь уже не понятно, что будет удалено)

AdminService.DeleteUser(long id)

1. При именовании методов следует выработать единый согласованный лексикон и следовать ему. Например, существование эквивалентных методов с именами *Get*, *Find*, *Fetch*, *Retrieve* неизбежно создаст путаницу.

TasksFinder.FindByNumber(string number), TasksFinder.FetchActive()

TasksFinder.FindByNumber(string number), TasksFinder.FindActive()

Для именования «обратных» методов следует использовать общепринятые пары антонимов.

OpenFile, \_lclose (идентификаторы несимметричны, вызывают замешательство)

OpenFile, CloseFile

Вот некоторые примеры популярных пар антонимов:

Add/Remove Increment/Decrement Open/Close

Begin/End Insert/Delete Show/Hide

Create/Destroy Lock/Unlock Source/Target

First/Last Min/Max Start/Stop

Get/Put Next/Previous Up/Down

Get/Set Old/New

1. В идентификаторах методов не следует использовать имена типов, а следует использовать слова, проясняющие семантику.

GetInt

GetLength

В тех редких случаях, когда у идентификатора нет семантического значения, нужно использовать имена типов CLR, а не имена языка.

ToLong

ToInt64

1. Методы-обработчики событий следует именовать, используя префикс «*On*».

OnInstallmentCreated

1. При именовании параметров методов должен использоваться стиль Lower Camel case. Идентификаторы параметров в большинстве случаев должны представлять собой существительные или именные группы. Параметром следует давать описательные имена.

Customer.CopyFrom(Customer c)

Customer.CopyFrom(Customer prototype)

Для параметров следует использовать имена, основанные на семантике, а не на типе. В тех случаях, когда у идентификатора нет никакого семантического значения, а тип не важен, следует использовать общие названия, такие как «*value*», «*item*», вместо того, чтобы повторять название типа. Последнее правило применимо к параметрам примитивных типов.

Console.Write(double value), List<T>.Add(T item)

EmployeesRepository.Add(Employee employee)

* + 1. **Свойства**

1. В идентификаторах свойств должен использоваться стиль Upper Camel case. Свойства должны иметь имена, являющиеся существительными, именными группами или прилагательными.

GetTextWriter (в основе имени — глагол «Get»)

TextWriter

Свойствам, являющимся коллекциями, необходимо давать имена во множественном числе. Не следует использовать единственное число с суффиксами «*List*», «*Collection*» и т. п.

IEnumerable<Refund> RefundList { get; set; }

IEnumerable<Refund> Refunds { get; set; }

1. Если сложно подобрать семантически специфичное название, то рекомендуется давать свойству имя, совпадающее с типом.

Invoice Invoice { get; set; }

1. Булевы свойства рекомендуется называть в утвердительно форме. При необходимости можно также добавлять и префиксы «*Is*», «*Can*» или «*Has*».

При выборе имени нужно учесть частоту использования свойства в условном операторе. Нужно стремиться к тому, чтобы полученные фразы имели смысл как английские формы (рассматриваются такие грамматические варианты как число, залог). Как следствие, например, действительный залог следует предпочитать страдательному.

CantReceive, Receivable, IsCreated

CanReceive, Created

* + 1. **События**

В идентификаторах событий должен использоваться стиль Upper Camel case. Т. к. события связаны с некоторым действием, которое происходит сейчас или уже произошло, при именовании событий следует использовать глаголы или глагольные фразы. Время выполнения события следует указывать временем глагола, а не префиксами или суффиксами «*Before*», «*After*» и т. п. Так, события происходящие перед действием, именуются в настоящем времени (форма -ing), а события, происходящие после завершения действия — в прошлом (форма -ed).

OrderShipment, AfterOrderShipment

OrderShipping, OrderShipped

* + 1. **Поля, константы**

1. Для именования общедоступных (public) статических полей и констант (с любой областью видимости) должны быть использованы правила именования свойств (см. раздел 2.4.2), т. к. эти элементы с точки зрения дизайна очень похожи на свойства.

String.Empty, Uint32.Min

1. Для именования закрытых (private) и защищенных (protected) полей должен использоваться стиль Lower Camel case и знак подчеркивания «\_» в качестве префикса. Имя поля должно быть существительным, именной группой или прилагательным.

\_receiptType, \_customer, \_quantity, \_available

**---------------------------------------------------------------------------**

* 1. **Именование локальных переменных**

Идентификаторы локальных переменных должны подчиняться правилам для параметров методов — см. раздел 2.4.1, п. 6.

**3. СОГЛАШЕНИЯ ПО ФОРМАТИРОВАНИЮ**

**3.1. Использование фигурных скобок**

**1.** Размещайте открывающую и закрывающую фигурные скобки с новой строки. Открывающую скобку выравнивайте на начало предыдущей строки. Закрывающую скобку выравнивайте по соответствующей открывающей.



if(someExpression) {

DoSomething();

}



if (someExpression)

{

DoSomething();

}

**2.** Блоки из одного оператора можно начинать и заканчивать в одной строке. Чаще всего это правило используется при написании аксессоров свойств.



public int Foo

{

get { return \_foo; }

set { \_foo = value; }

}

Это правило действует и в случаях объявления свойства в интерфейсе, абстрактного и автоматического свойств.

int Foo { get; set; }

**3.** Не опускайте скобки, даже если язык это разрешает. Следование этому правилу облегчает внесение изменений в код.



if (someExpression)

DoSomething();



if (someExpression)

{

DoSomething();

}

**Исключение:** скобки можно опускать в блоках оператора *switch*.



switch(someValue)

{

case 0:

Foo();

break;

case 1:

Bar();

...

}

**3.2. Горизонтальное форматирование**

**1.** Строки кода должны быть максимально короткими. Не допустима ситуация, когда строка не помещается на экране.

**2.** Для отображения иерархии в файле с исходным кодом используйте отступы. Определение пространства имен отступа не имеет. Определения классов сдвигаются на один уровень по отношению к пространству имен. Определения членов класса сдвигаются еще на один уровень вправо. Реализация методов — еще на один уровень и т. д.



if (someExpression)

{

foreach (var item in collection)

{

Foo(item);

Bar(item);

}

}

**3.** Если открывающая и закрывающая фигурные скобки следуют в одной строке, то после открывающей и перед закрывающей скобками ставится один пробел.

int Foo { get; set; }

**4.** Не следует ставить пробелы после открывающей, перед закрывающей круглыми и квадратными скобками.

Foo( a, b, c );

Foo(a, b, c);

**5.** Не следует ставить пробел между именем вызываемого метода и скобкой, открывающей список параметров.

Foo (a, b, c);

Foo(a, b, c);

**6.** При объявлении и вызове метода ставьте пробел после запятой, разделяющей его параметры.

void Foo(int x,int y,int z)

void Foo(int x, int y, int z)

Foo(a,b,c)

Foo(a, b, c)

**7.** Не рекомендуется ставить пробелы между унарным оператором и его операндом. Стандарт кодирования (C#/.NET) 17

if (! someExpression);

if (!someExpression)

**8.** Для группировки взаимосвязанных элементов и разделения разнородных используйте пробелы.

if (x==y);

if (x == y)

d=b\*b-4\*a\*c;

d = b\*b – 4\*a\*c;

9. **Логические блоки в коде следует отделять пустой строкой.**

Matrix4x4 matrix = new Matrix4x4();

double cosAngle = Math.cos(angle);

double sinAngle = Math.sin(angle);

matrix.setElement(1, 1, cosAngle);

matrix.setElement(1, 2, sinAngle);

matrix.setElement(2, 1, -sinAngle);

matrix.setElement(2, 2, cosAngle);

multiply(matrix);

Улучшает читаемость.  
10. **Конструкцию**try-catch**следует оформлять следующим образом:**

try {

statements;

}

catch (Exception& exception) {

statements;

}

**11. Цикл**for**следует оформлять следующим образом:**

for (initialization; condition; update) {

statements;

}

Следствие из правила, указанного выше.  
  
**12. Цикл**for**с пустым телом следует оформлять следующим образом:**

for (initialization; condition; update)

;

Делает акцент для читающего на том, что тело пусто. Однако циклов, не имеющих тела, следует избегать.   
  
**13 Цикл**while**следует оформлять следующим образом:**

while (condition) {

statements;

}

Следствие из правила, указанного выше.  
  
**14. Цикл**do-while**следует оформлять следующим образом:**

do {

statements;

} while (condition);

**3.3. Вертикальное форматирование**

**1.** Не следует допускать наличие больших файлов с исходным кодом (объем порядка тысячи строк — это уже перебор).

**2.** Каждая группа строк исходного кода представляет собой законченную мысль. Для облегчения восприятия такие группы следует разделять пустыми строками. И наоборот, отсутствие пропусков подчеркивает тесную связь строк. Так:

объявление пространства имен (директива *namespace*) отделяется пустой строкой от импортов пространств имен (директива *using*);

перед каждым методом должна быть пустая строка;

перед каждым свойством и событием, определение которого занимает более одной строки, следует помещать пустую строку;

между свойствами (событиями), определение каждого из которых занимает одну строку, и между объявлениями полей пустая строка не требуется;

пустая строка требуется между логическими группами операций, даже если они находятся в реализации одного метода.

**3.** Методы и аксессоры свойств и событий нужно делать максимально короткими. Лучше несколько простых коротких методов с ясными именами, чем один здоровый метод, в котором даже автор разбирается с трудом. 18 Стандарт кодирования (C#/.NET)

**4. СОГЛАШЕНИЯ ПО СТРУКТУРИРОВАНИЮ**

**1.** В одном файле размещайте один класс. Называйте файл по имени класса.

**2.** Размещайте директивы *using* перед объявлением пространства имен (директива *namespace*).

**3.** Члены типа должны быть сгруппированы в следующие разделы в порядке перечисления:

внутренние типы;

все константы и статические неизменяемые поля;

экземплярные поля;

конструкторы;

свойства;

открытые методы (следует разбить на логические подразделы, если методов много);

события;

явные реализации интерфейсов;

внутренние, защищенные и закрытые методы.

Для группировки используйте регионы.

**4.** Элементы, связанные друг с другом размещайте рядом друг с другом (по вертикали). Например, помещайте объявление переменной в непосредственно близости от места ее первого использования. Если один метод вызывает другой, то эти методы должны находиться рядом, причем вызывающий метод должен находиться над вызываемым.

**5.** Иерархия каталогов проекта должна отражать иерархию пространств имен.

Пусть для проекта корневым является пространство имен *Hecsit.Pos*. Тогда файл с классом Стандарт кодирования (C#/.NET) 19

*Hecsit.Pos.Domain.Discounts.FixedDiscount* должен находиться в каталоге «Domain\Discounts».

**5. СОГЛАШЕНИЯ ПО КОММЕНТИРОВАНИЮ**

«На самом деле комментарии в лучшем случае являются неизбежным злом»

*Р. Мартин*

«Не комментируйте плохой код —

перепишите его»

*Б. У. Керниган, П. Дж. Плауэр*

**1.** Пишите **само**документированный код, т. е. такой код, который понятен благодаря именованиям, своему форматированию и структурированию (в том числе и декомпозиции), а не комментариям. Не допустимы: избыточные и очевидные комментарии; комментарии, не соответствующие текущему состоянию кода; комментарии, связь которых с кодом неочевидна; комментарии, называющие автора или историю изменений. Такие комментарии нужно удалять или, если они все-таки требуются, исправлять.

**2.** Комментарии можно применять в следующих случаях.

В начале файла с исходным кодом. Могут содержать информацию об авторском праве, ссылку на лицензию и т. п.

Пояснение к коду. Но чаще всего таких комментариев удается избежать путем переименования или выделения отдельного класса/метода и т. п.



// Возвращает тестируемый экземпляр IDiscountsService

protected abstract IDiscountsService GetDiscountsService();

Комментарий не сообщает почти ничего нового. Его можно избежать, переименовав метод в GetDiscountsServiceForTest.

20 Стандарт кодирования (C#/.NET)

// Поиск по формату kk:mm:ss EEE. MMM dd. yyyy

protected static readonly string TimePattern

= "\\d\*:\\d\*:\\d\* \\w\*. \\w\* \\d\*. \\d\*";

Однако, код был бы лучше, если бы его перенесли в специальный класс, отвечающий за преобразование даты/времени.

Комментарии «TODO». Напоминают о том, что должно быть сделано, но не может быть сделано в момент написания. Плагин ReSharper содержит в себе средства по поиску и управлению такими комментариями. Однако, код не должен загромождаться такими комментариями: регулярно просматривайте их и удаляйте те, которые неактуальны.

XML-документация. Эти комментарии в обязательном порядке должны составляться для открытого API разрабатываемых библиотек и фреймворков. Кроме того, желательно документировать и код в других проектах.

**3.** Удаляйте закомментированный код. Не превращайте код в свалку.

**4.** Для комментирования используйте синтаксис однострочного комментария «//» даже для многострочных комментариев. Отделяйте текст комментария от «//» пробелом. Помещайте комментарий перед поясняемым фрагментом кода, но допускается использование комментариев в конце кода при объявлении переменных и в операторах «*case*», «*default*». Соблюдайте в комментариях правила грамматики, орфографии и пунктуации. Не шутите в комментариях.

**6. ПРОЧЕЕ**

**1.** При объявлении переменных используйте ключевые слова языка, а не имена типов FCL.

void Foo(Int32 x, String y, Object z)

void Foo(int x, string y, object z) Стандарт кодирования (C#/.NET) 21

**2.** При объявлении локальной переменной используйте ключевое слово «*var*» в тех случаях, когда тип переменной очевиден.

var source = GetSource();

var names = new List<string>();

var tokens = source.Split(' ');

var d = b\*b - 4\*a\*c;

**3.** Не используйте полностью квалифицированные имена типов, импортируйте нужные пространства имен с помощью операторов «*using*».

var areEqual = System.String.Equals(left, right);

var areEqual = string.Equals(left, right);

**4.** Для определения анонимных методов, состоящих из одного оператора, используйте синтаксис лямбда-выражений.

**7. BEST PRACTICES**

**7.1. Обработка ошибок**

**1.** Не следует использовать код ошибки как значение, возвращаемое методом. Используйте механизм исключений.

**2.** Перехватывайте только те исключения, о возможности возникновения которые знаете и которые можете обработать.

**3.** Никогда не «проглатывайте» исключения (т. е. не ловите, ничего не делая).

**4.** Если ошибка не может быть полностью устранена, то из блока «*catch*» пробросьте исключение «выше». Сгенерируйте то же исключение (оператор «*throw*» без параметров) или сгенерируйте новое, передав старое как «*InnerException*». Это нужно для того, чтобы не потерять информацию о месте и причине изначальной ошибки. 22 Стандарт кодирования (C#/.NET)

**5.** При генерации ошибки, по возможности, используйте стандартные исключения FCL (*ArgumentException* и его потомков, *InvalidOperationException* и проч.). Никогда не генерируйте «системные» исключения *NullReferenceException*, *OutOfMemoryException*, *StackOverflowException* и подобные им. Определяйте собственное исключение только если стандартные не подходят. При этом наследуйте от *System*.*Exception*.

**6.** Сообщение об ошибке должно отвечать на следующие вопросы: что послужило причиной ошибки, и как ошибку исправить? Собщение должно быть написано с учетом правил орфографии, грамматики и пунктуации.

**7.2. Работа со строками**

**1.** Используйте *string.Empty* вместо "".

**2.** Используйте *System.Environment.NewLine* вместо "\n".

**3.** Для создания сложных строк используйте не оператор конкатенации, а метод *string.Format* или класс *System.Text.StringBuilder*.

**4.** При сравнении и упорядочении строк используйте методы, позволяющие явно задать правила учета региональных настроек и регистра символов. При этом при сравнении строк для «внутреннего пользования» (пути, имена файлов, ключи реестра, теги и атрибуты XML, переменные окружения и т. п.) следует использовать опцию *StringComparison.Ordinal* или *StringComparison.OrdinalIgnoreCase*. Для сравнения строк с учетом языковых особенностей — опцию *StringComparison.CurrentCulture* или *StringComparison. CurrentCultureIgnoreCase*.

**5.** Никогда не «хардкодьте» строки, которые будут отображаться конечному пользователю. Используйте ресурсы. Стандарт кодирования (C#/.NET) 23

**6.** Никогда не «хардкодьте» строки, которые могут меняться во время развертывания приложения (например: пути, адреса, строки подключения). Используйте конфигурационные файлы.

**7.3. Работа с делегатами и событиями**

**1.** Перед вызовом делегата, проверьте его на *null*.

**7.4. Общий дизайн. Разное**

**1.** Старайтесь, чтобы у классов и у отдельно взятых методов была ровно одна обязанность, которая явно определена идентификатором класса/метода. В случае класса все его открытые элементы должны быть согласованы с этой обязанностью.

**2.** Все экземплярные поля определяйте как закрытые. Если же к данным нужен доступ с другим уровнем, определите свойство.

**3.** Не используйте в коде «магические числа» и «магические строки», вместо этого определите и используйте константу (исключение — числа 0 и 1).

if (code == "VIP") ...

var start = 385170;



private static readonly string VipCode = "VIP";

private static readonly int StartIndex = 385170;

if (code == VipCode) …

var start = StartIndex;

**4.** Определяйте константы (директива «*const*») только для величин, которые являются константами по сути. Например, количество дней в неделе, электрическая постоянная и т. п. Для представления остальных величин используйте статические readonly-поля.

**5.** Вместо числовых констант по возможности используйте перечисления. 24 Стандарт кодирования (C#/.NET)

**6.** Для перечислений, не являющихся флагами, не следует явно определять значения. Не следует явно указывать базовый тип перечисления.

**7.** По возможности используйте тернарный условный оператор «?:».