Рязанский станкостроительный колледж РГРТУ

Стандарт оформления кода C#

Рязань 2020

Оглавление

Основные положения

Стандарт является соглашением по оформлению и написанию кода на языке C#. В документе приведены основные правила оформления кода и приемы, используемые при написании программ.

Стандарт описывает принципы построения программного кода в части:

- Структурирования программных кодов в соответствии с правилами, изложенными в настоящем документе.
- Написания содержательных пояснений (комментариев) к программным кодам в соответствии с правилами, изложенными в настоящем документе.

Стандарт является обязательным для любых разработок программных систем, а также разработки учебных программ в рамках учебного процесса РССК РГРТУ.

Каждое требование и рекомендация Стандарта действительны только для тех языков программирования, где допускается их использование и они технически выполнимы.

Непосредственные руководители разработчиков программного кода, преподаватели, курирующие разработки учебных программ, обязаны осуществлять контроль соблюдения настоящего Стандарта.

Стили наименования

PascalCase – первая буква каждого слова в имени идентификатора начинается с верхнего регистра.

Пример: TheCategory.

CamelCase – первая буква первого слова в идентификаторе в нижнем регистре, все первые буквы последующих слов – в верхнем.

Пример: theCategory.

UpperCase – стиль в ряде случаев используется только для сокращений, все буквы в имени идентификатора в верхнем регистре.

Пример: ID.

Hungarian notation – перед именем идентификатора пишется его тип в сокращенной форме.

Пример: strFirstName, iCurrentYear.

Правила именования идентификаторов

Общие правила именования идентификаторов

Для названий любых элементов программного кода (модулей, классов и их методов, функций, переменных, констант, последовательностей, представлений), следует использовать содержательные имена, позволяющие сделать вывод о назначении данных элементов. При именовании идентификаторов не используются аббревиатуры или сокращения, если только они не являются общепринятыми.

Пример: GetWindow(), а не GetWin();

Если имя идентификатора включает в себя сокращение – сокращение пишется в PascalCase. Исключение - когда имя идентификатора должно быть указано в CamelCase и

сокращение стоит в начале имени идентификатора. В этом случае сокращение пишется в нижнем регистре.

Пример:
SqlAccount для PascalCase, sqlAccount для CamelCase.

Использование верхнего и нижнего регистра в именах

Запрещается создавать два различных имени, функции или свойства с одинаковыми именами, отличающиеся только регистром. Запрещается создавать функции с именами параметров, отличающимися только регистром. Ниже приведены примеры НЕправильных названий.

Пример:
KeywordManager и Keywordmanager;
int id {get, set} и int ID {get, set};
findByID(int id) и FindByID(int id);
void MyFunction(string s, string S).

Правила именования переменных

Для именования переменных используется стиль CamelCase;

Пример: int userId = 20;

В циклах foreach имя переменной назначается как имя массива в единственном числе.

Пример: foreach(Campaign newCampaign in NewCampaigns).

Правила именования констант

Для именования констант используется стиль PascalCase.

Правила именования функций

Для именования функций используется стиль PascalCase;

Функции объявляются согласно следующему шаблону:

<Модификатор доступа> [Другие модификаторы] <Тип> <Название функции>();

Пример: protected abstract void HelloWorld();

Имена функций должны давать четкое представление о том, какое действие эта функция выполняет. Имя функции начинается с глагола, указывающего на то, какое действие она выполняет.

Пример: public void CreateItem();

Имена методов класса или функций, которые возвращают значения, должны иметь свой унифицированный префикс «Get», например, “GetStatus”.

Имена методов класса или функций, которые устанавливают значения, должны иметь свой унифицированный префикс «Set», например, “SetStatus”.

Правила именования параметров функций

Для именования параметров используется стиль CamelCase;

Имена параметров должны давать четкое представление о том для чего используется параметр, и какое значение следует передать при вызове функции.

Пример:

public void EncodeString(string sourceString, ref string encodedString),

а не public void EncodeString(string string1, ref string string2).

В том случае, когда это не препятствует понимаю кода, в качестве имени параметра функции используется имя соответствующего параметру класса. Для коллекций и массивов используется имя объектов, содержащихся в коллекции или массиве.

Пример:

```
UserFactory.Create(Company company);
CheckUsers(UserCollection users);
```

Имена параметров не должны совпадать с именами членов класса, если этого не удается избежать, то для разрешения конфликтов используется ключевое слово this.

Пример:

```
public void CreateUser(string firstName, string lastName)
{
        this.firstName = firstName;
        this.lastName = lastName;
}
```

Правила именования классов

Следует избегать имен классов, совпадающих с именами классов .NET Framework;

Для классов используется стиль именования PascalCase;

Для классов, унаследованных от CollectionBase используется суффикс Collection, перед которым указывается тип объектов, для которых используется коллекция.

Пример: UserCollection, CompanyCollection;

В качестве имен классов используются имена существительные;

Имя класса не должно совпадать с именем namespace’а.

Пример: namespace Debugging, класс Debug;

Если класс представляет собой сущность, хранимую в базе данных – имя класса соответствует имени таблицы. В этом случае имя класса – это название сущности в единственном числе, имя таблицы – во множественном числе.

Пример: таблица Users, класс User;

При создании производных классов их имена могут состоять из имени базового класса и префикса производного класса.

Пример:

базовый класс Figure, производный класс CircleFigure;

Имена файлов, в которых находятся классы, совпадают с именами классов. Для именования файлов используется стиль PascalCase.

Правила именования интерфейсов

Имена интерфейсов начинаются с буквы I, после которой следует название интерфейса в PascalCase.

Пример: IDisposable.

Правила именования свойств

Для именования свойств используется стиль PascalCase;

Свойства объявляются согласно следующему шаблону:

<Модификатор доступа> [Другие модификаторы] <Тип> <Название свойства>;

Пример: public static User CurrentUser { get; }

В том случае, когда это не препятствует понимаю кода, в качестве имени свойства используется имя соответствующего свойству класса. Для коллекций и массивов используется имя объектов, содержащихся в коллекции или массиве.

Пример:
public User User { get; set; }
public UserCollection Users { get; set; }

Название свойства типа bool должно представлять из себя вопрос, требующий ответа да или нет.

Примеры названий: “CanDownload”, “HasKeywords”, “IsChecked”, “NeedsUpdate”.

Правила именования полей

Для именования полей, доступных вне класса, используется стиль PascalCase, для private полей – CamelCase ;

Поля объявляются согласно следующему шаблону:

<Модификатор доступа> [Другие модификаторы] <Тип> _<Название поля>;

Пример: private static User _currentUser = null;

В том случае, когда это не препятствует понимаю кода, в качестве имени поля используется имя соответствующего полю класса. Для коллекций и массивов используется имя объектов, содержащихся в коллекции или массиве.

Пример:
public User User1 = new User();
public UserCollection Users = new UserCollection();

Название поля типа bool должно представлять из себя вопрос, требующий ответа да или нет.

Примеры названий: “CanDownload”, “HasKeywords”, “IsChecked”, “NeedsUpdate”.

Правила именования enum’ов

Для именования enum’ов и их значений используется стиль PascalCase.

Имена enum’ов указываются в единственном числе.

Имена, как правило, состоят из имени сущности, к которой относится enum и названия содержимого enum’а (status, type, state).

Пример: KeywordStatus, ConnectionState, TaskType.

Правила именования exception’ов

Для exception’ов используется стиль PascalCase;

Имена классов для создаваемых custom exception’ов заканчиваются суффиком Exception;

В качестве имени объекта исключения внтури catch, для исключений типа Exception, используется имя “ex”.

Пример: catch(Exception ex).

Форматирование кода программы

Используются стандартные настройки форматирования Visual Studio.

Для улучшения наглядности структуры программного кода следует использовать пробелы и отступы:

- Пробелы – для лучшего обозначения отдельных элементов кода в строке.
- Горизонтальные отступы – для лучшего обозначения вложенности функциональных блоков многострочного кода. При формировании горизонтальных отступов рекомендуется использовать tab - 4 пробела.
- Вертикальные отступы – для отделения логических блоков программного кода между собой.

В одном файле не объявляется больше одного namespace’а и одного класса (исключение – небольшие вспомогательные private классы).

Фигурные скобки размещаются всегда на отдельной строке.

В условии if-else всегда используются фигурные скобки.

Использование строк длиннее 100 символов не желательно. При необходимости инструкция переносится на другую строку. При переносе части кода на другую строку вторая и последующая строки сдвигаются вправо на один символ табуляции.

Каждая переменная объявляется на отдельной строке.

Все подключения namespace’ов (using) размещаются в начале файла, системные namespace’ы объявляются над custom namespace’ами.

Если для свойства существует соответствующее поле (например, при загрузке по требованию), то поле объявляется над свойством.

Пример:

```
private User user;
public User User { get; set; }
```

Если set или get свойства состоит из одной операции – весь set или get размещается на одной строке.

Пример:

```
Public User
{
        get {return user; }
}
```

Функции, поля и свойства группируются внутри класса по своему назначению. Такие группы объединяются в регионы.

Комментирование кода программы

Все комментарии должны быть на русском языке.

В комментариях не допускается использование слов, не относящихся к техническому или деловому лексикону.

В комментариях не допускается наличие информации, не несущей смысловую нагрузку к пояснению сути алгоритма программного кода. Например, недопустимы комментарии вида: «это гениальный код», «таблица1», «!№;%:?*», «Это комментарий», «Эту фигню все равно никто не читает» и подобные им.

Если для языка программирования применяется система контроля версий, то комментарии вида: дата создания; дата модификации; автор создания; автор модификации - допускается вести в ней.

Комментарии к программному модулю, представленному в виде отдельного файла необходимо указывать всегда, при этом обязательны следующие комментарии:

- ***Описание назначения.***
- ***Дата создания.***
- ***Автор создания.***
- Дата модификации (при необходимости).
- Автор модификации (при необходимости).
- Цель модификации (при необходимости).

Комментарии к переменным, заголовкам свойств, полей, интерфейсов и прочего необходимо указывать всегда, при этом обязательны следующие комментарии:

- ***Описание назначения.***

Комментарии к заголовкам функций необходимо указывать всегда, при этом обязательны следующие комментарии:

- ***Описание назначения.***
- ***Описание входных и выходных параметров (при их наличии).***

Для функций, выполняющих сложные алгоритмы, не очевидные для восприятия, необходимо указывать подробные комментарии не только к заголовку функции, но и самому алгоритму с пояснением каждого шага выполнения алгоритма.

Для удобства работы с большой глубиной вложенности программных структур рекомендуется добавлять следующие комментарии:

- Метка окончания каждого ветвления алгоритма с указанием типа ветвления и краткого текста с условием.
- Метка окончания каждого цикла с указанием типа цикла и названия переменной-счетчика.
- Метка окончания модуля, класса, метода класса, события класса, функции, процедуры, триггера, пакета, с указанием названия закончившегося элемента программного кода, если указание названия не предусмотрено синтаксисом используемого языка.

Комментарии к тексту программного кода должны быть лаконичными и пояснять суть и назначение программного кода, а не его техническую составляющую. Например, вместо комментария «увеличиваем счетчик на 1» нужно писать «переходим к обработке следующего факультета». Текст комментариев не должен иметь сокращений, понимание которых может быть неоднозначным или затруднительным.

Пример программного кода с соблюдением правил оформления кода (язык C++):

```cpp
// 14.01.2020  Сидоров С.С. ИСП-21
// Практическая работа №12 Сортировка
// Сортировка строк двумерного массива методом пузырька
#include <iostream>
#include <iomanip>
#include <stdlib.h>
#include <time.h>
using namespace std; //Установка стандартного пространства имен

//печать массива
void printMatrix
(int[][10], // Сортируемый массив
    const int // Размерность массива
);

int main()
{
    const int Size = 10; // Размерность массива
    int matrix[Size][Size]; // Определяем массив
    int temp; // Временная переменная
    srand(time(NULL)); // Определяем исходную точку для генератора случайных чисел

    // Заполняем массив случайным образом
    for (int i = 0; i < Size; i++)
        for (int j = 0; j < Size; j++)
            matrix[i][j] = 1 + rand() % 100;

    // Сортируем пузырьком
    for (int n = 1; n < Size * Size; n++)
    {
        for (int i = 0; i < Size; i++)
        {
            for (int j = 0; j < Size - 1; j++)
            {
                // Если нужно скорректировать порядок сортировки
                if (matrix[i][j + 1] < matrix[i][j])
                {
                    // Делаем обмен элементов массива местами
                    temp = matrix[i][j + 1];
                    matrix[i][j + 1] = matrix[i][j];
                    matrix[i][j] = temp;
                }// for (int j = 0; j < Size - 1; j++)
            }// for (int i = 0; i < Size; i++)
        }// for (int n = 1; n < Size * Size; n++)

        //Выводим массив на печать
        printMatrix(matrix, Size);

        return 0;
    }
}// end of main
```

```
// Печать массива
// Входные знвчения:
// varMX[][10] - сортируемый массив
// Size - размер массива
void printMatrix(int varMX[][10], const int size)
{
    for (int i = 0; i < size; i++)
    {
        cout << endl;
        for (int j = 0; j < size; j++) // Печать j-массива на одной строке
            cout << setw(4) << varMX[i][j];
        cout << endl; // Каждый j-массив на отдельной строке
    }
}// end of printMatrix
```

Переменные и типы

Свойства необходимо использовать только тогда, когда это имеет смысл. Если при получении и сохранении значений никакая дополнительная логика не участвует – вместо свойства необходимо использовать поле.

Необходимо использовать максимально простые типы данных. Так, например, необходимо использовать int, а не long, если известно, что для хранимых значений будет достаточно типа int.

Константы необходимо использовать только для простых типов данных.

Для сложных типов вместо констант необходимо использовать readonly поля.

При задании значений нецелых типов, значения должны содержать как минимум одну цифру до точки и одну после.

Поля и переменные инициализируются при их объявлении, когда это возможно.

Пример:
private int userId = -1;
private string firstName = “”;
private string lastName = “”;

Модификаторы доступа необходимо указывать всегда. Не смотря на то, что по умолчанию назначается модификатор доступа private, поле модификатора не остается пустым –модификатор private необходимо указывать явным образом.

Пример: private User CreateUser(string firstName, string lastName);

Вместо использования “magic numbers” для идентификаторов статусов, состояний и т.п. необходимо указывать константы или enum’ы. Идентификаторы состояний в виде чисел использовать нельзя.

Пример не правильного использования: public GetUserByStatus(int statusId);
Пример правильного использования: public GetUserByStatus(UserStatus userStatus);

Тип object необходимо использовать только когда это действительно необходимо, в большинстве случаев вместо него используются generic’и. Вместо Hashtable необходимо использовать Dictionary<>, вместо ArrayList используется List<>.

В том случае если в get’е или set’е какого-либо свойства выполняются сложные вычисления, если операция, выполняемая в get или set является преобразованием, имеет побочный эффект или долго выполняется – свойство должно быть заменено функциями.

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, вместо свойства необходимо использовать функции.

Внутри get'а и set'а не должно быть обращений к коду, не связанному напрямую с получением или сохранением значения свойства, т.к. такие действия могут быть не очевидны для клиентов, использующих свойство.

Приложение А Сводная таблица правил именования

Идентификатор	Регистр	Пример
Класс	Pascal	User
Локальная переменная	Camel	user
Интерфейс	Pascal	IDisposable
Generic	Pascal	T, TKey, TValue
Public функция	Pascal	Authenticate
Private функция	Pascal	Authenticate
Параметр функции	Camel	userId
Public свойство	Pascal	FirstName
Private свойство	Pascal	FirstName
Public поле	Pascal	FirstName
Private поле	Camel	firstName
Enum	Pascal	UserStatus
Значение enum’а	Pascal	Active
Exception	Pascal	UserAuthenticationException
Event	Pascal	StatusChanged
Namespace	Pascal	UserManager