فهرست مطالب

مقدمه

استانداردهای کدنویسی بخش مهمی از پروژه‌های بزرگ هستند و برای بالا بردن خوانایی، کیفیت، عملکرد و قابل پیش‌بینی بودن کدها، تمام اعضای گروه موظف به رعایت این استانداردها هستند.

در تمام پروژه‌های زیرساخت و برنامه تدبیر، تمام این نکات تا حد امکان رعایت شده‌اند. علاوه بر نکات طرح شده در این مستند، استانداردهای رایج و اولیه‌ای هم که از ابتدای انتشار سکوی کاری دات نت توسط مایکروسافت ارائه شده، رعایت شده‌اند. مطالعه بخش‌های اصلی مستند مایکروسافت و رعایت نکات آن نیز برای همه اعضاء این پروژه ضروری است. نکاتی که به اشتباه یا به دلایل مختلف (مثل محدودیت زمانی برای تحویل به‌موقع) رعایت نشده باشند، در جلسات مرور کد به برنامه‌نویس گوشزد شده و هر کس موظف است ناهماهنگی‌ها و موارد رعایت‌نشده را با اولویت بالا و در اولین فرصت اصلاح کند. بدیهی است که رعایت نکات ریز طرح شده در همه استانداردهای کدنویسی مورد نیاز این پروژه، بخشی از معیارهای تکمیل شدن کار است و مادامی که اشکالات کدها برطرف نشده باشند، کار مورد نظر ناتمام به حساب می‌آید و امکان شروع کار جدید وجود نخواهد داشت.

لازم به یادآوری است که بعضی از موارد طرح شده در این مستند ممکن است لزوماً بهترین روش ممکن برای توسعه نرم‌افزاری نباشد. با این حال، برای حفظ یکدستی کار و بهره‌گیری از سایر مزایای استانداردسازی کدها، رعایت همه نکات ضروری است. در ضمن، این مستند در دست تکمیل است و نکات بیشتری که منجر به وضوح بیشتر یا عملکرد مطمئن‌تر یا سریع‌تر برنامه می‌شوند، به‌تدریج به موارد موجود اضافه می‌شوند. بنابراین برای کلیه اعضاء گروه، مطالعه و مرور دوره‌ای این مستند ضروری است.

برای مطالعه استانداردهای مایکروسافت برای برنامه‌نویسی در محیط دات نت، به آدرس وب زیر مراجعه کنید :

https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/

۱. قواعد کلی کدنویسی

۱,۱ قواعد اولیه

۱. برنامه موجود یک برنامه با واسط کاربری چندزبانه است. بنابراین تمام متن‌هایی که درون برنامه دیده می‌شوند، باید داخل ریسورس‌ها قرار بگیرند. از قرار دادن متن‌های هاردکد شده بین گیومه جداً خودداری می‌کنیم.

۲. از عبارات شرطی تودرتو که منجر به پیچیدگی و ناخوانایی کد می‌شوند، جداً پرهیز می‌کنیم. عبارات شرطی در حالت عادی باید در یک سطح یا در موارد خاص، در دو سطح منطق مورد نیاز را پیاده‌سازی کنند.
۳. حداکثر طول هر خط از کد را بین ۱۰۰ تا ۱۲۰ کاراکتر نگه می‌داریم. خطوط طولانی‌تر را با استفاده از فرورفتگی (indent) مناسب می‌شکنیم.
۴. تعداد سطرهای مورد نیاز برای پیاده‌سازی متدها را تا حد امکان محدود می‌کنیم و از ایجاد متدهای دارای ۶۰ خط یا بیشتر جلوگیری می‌کنیم. بخش‌های مختلف کدهای پیچیده را در توابع کمکی با نام‌های گویا و قابل پیش بینی قرار می‌دهیم و آنها را در تابع اصلی صدا می‌زنیم.
۵. در هر فایل سی شارپ بیشتر از یک کلاس قرار نمی‌دهیم و نام فایل را با نام کلاس مربوطه یکسان در نظر می‌گیریم.
۶. هنگام تعریف یک کلاس، بخش‌های عمومی (public) را در ابتدا قرار می‌دهیم، پس از آن بخش‌های محافظت‌شده (protected) و در انتها بخش‌های خصوصی (private) را قرار می‌دهیم.
۷. هنگام تعریف یک کلاس، متد یا متدهای سازنده (constructor) را ابتدای تعریف کلاس می‌گذاریم. سپس ویژگی‌ها (properties) و بعد از آن متدها را قرار می‌دهیم.
۸. هنگام استفاده از متدهای استاتیک در انواع داده‌ای پایه دات نت (int, long, double, string,...) از نام کامل کلاس‌ها (Int32, Int64, String,...) استفاده می‌کنیم، نه نام مستعار آنها در سی شارپ. (مثال)
۹. برای تعریف سطح دسترسی ها، تا حد امکان از سطح دسترسی داخلی (internal) استفاده نمی‌کنیم.

۱,۲ قواعد فاصله‌گذاری

۱. همان‌طور که فاصله نگذاشتن بین اجزای مختلف کد موجب ناخوانایی کد می‌شود، استفاده از فاصله‌های بی‌رویه هم به مرتب بودن کد لطمه می‌زند. در مواردی که فاصله لازم است، فقط یک فاصله می‌گذاریم. (مثال)
۲. قبل و بعد از هر عملگر دوتایی حتماً یک فاصله می‌گذاریم.
۳. قبل و بعد از عملگر لامدا (=>) حتماً یک فاصله می‌گذاریم.
۴. کاراکترهای خالی اضافه در ابتدای خطوط خالی و انتهای سایر خطوط کد یا توضیحات را حتماً حذف می‌کنیم.
۵. فرورفتگی کدها (indent) را با فواصل مضرب ۴ انجام می‌دهیم تا همراستایی (alignment) کدها به هم نخورد.

۶. برای فرورفتگی‌ها (indenting) در سورس به‌هیچ‌وجه از کاراکتر Tab استفاده نمی‌کنیم. محیط ویژوال استودیو به طور خودکار Tab را تبدیل به فاصله می‌کند. این تنظیم پیش‌فرض را تغییر نمی‌دهیم.
۷. فرورفتگی‌ها را از بالا به پایین همراستا می‌کنیم. رعایت نکردن این اصل منجر به نامرتب شدن کدها شده و به خوانایی کد آسیب می‌زند. (مثال)
۸. فرورفتگی‌ها را به طور منظم رعایت می‌کنیم و در سطوح جدید کد (داخل حلقه‌ها، دستورات شرطی و موارد مشابه) بیشتر از یک سطح به فرورفتگی کد اضافه نمی‌کنیم. (مثال)

۱,۳ قواعد نامگذاری

۱. برای نامگذاری فایل‌ها، کلاس‌ها، متدها، ویژگی‌ها و ... از نام‌های ساده یا ترکیبی انگلیسی استفاده می‌کنیم و در انتخاب نام، تأکید اصلی را روی خوانایی و قابل پیش بینی بودن نامها قرار می‌دهیم.
۲. بخش‌های جداگانه از نامها را با ابزار Spell Checker کنترل می‌کنیم تا نامها خطای تایپی یا املایی نداشته باشند.
۳. نام تمام فیلدهای داخلی (private یا protected) را با کاراکتر زیرخط (underscore یا _) شروع کرده و برای مشخص کردن اعضای کلاس از بکارگیری کلمه this پرهیز می‌کنیم.
۴. نام آرگومان‌های متد و متغیرهای محلی متدها را با کاراکتر زیرخط شروع نمی‌کنیم.
۵. تمام پروژه‌ها و namespace های مربوط به محصول تدبیر را با پیشوند SPPC.Tadbir و تمام پروژه‌ها و namespace های مربوط به زیرساخت تدبیر را با پیشوند SPPC.Framework نامگذاری می‌کنیم.
۶. آدرس های اینترنتی (URL) را همیشه با حروف کوچک (lower-case) تعریف و استفاده می‌کنیم.
۷. هنگام نامگذاری کلاس‌ها، متدها، متغیرها و غیره از کاراکتر زیرخط (_) برای جدا کردن کلمات استفاده نمی‌کنیم.
۸. قواعد متعارف دات نت را در استفاده از حروف کوچک و بزرگ رعایت می‌کنیم. (مثال)
۹. برای فیلدهای ثابت (const) از شکل مشابه کلاس‌ها و متدها استفاده می‌کنیم (Pascal-case) و برای سایر فیلدها از شکل متداول (camel-case) استفاده می‌کنیم.

۱,۴ قواعد مربوط به using

۱. تمام اعلانات مربوط به ماجول‌های مورد استفاده (دستورات using) خارج از namespace قرار می‌گیرند.
۲. دستورات using را همیشه به صورت مرتب‌شده نگهداری می‌کنیم. در مرتب‌سازی دستورات using، تمام مواردی که مربوط به پلتفرم اصلی دات نت هستند (و با کلمه System شروع می‌شوند) بدون درنظر گرفتن ترتیب الفبایی، همیشه در ابتدا قرار می‌گیرند.

۳. دستورات using که استفاده نشده‌اند را همیشه حذف می کنیم (دستورات using مربوط به System را به دلیل اینکه خیلی پایه‌ای و متداول است، می‌توانیم نگه داریم).
۴. بین دستورات using هیچ خط خالی قرار نمی‌دهیم. این دستورات نیازی به طبقه‌بندی ندارند.

۲. قواعد سبک کدنویسی

۲,۱ قواعد مدیریت و نگهداری پروژه ها در solution اصلی

۱. هنگام اضافه کردن پروژه به یک solution، نام پروژه، اسمبلی و فولدر اصلی پروژه (و در بیشتر مواقع، namespace پیش‌فرض) را یکسان درنظر می‌گیریم.
۲. اینترفیس‌ها و پیاده‌سازی پیش‌فرض آنها را - تا حد امکان - در یک پروژه و اسمبلی قرار نمی‌دهیم.
۳. اینترفیس‌ها را در یک پروژه و اسمبلی مرکزی تعریف کرده و با استفاده از فولدرها طبقه‌بندی می‌کنیم.
۴. تا حد امکان رفرنس‌های استفاده نشده و اضافی را از پروژه‌ها حذف می‌کنیم.

۲,۲ قواعد افزایش خوانایی کد

۱. برای حفظ یکدستی کدها و مستندات، پیش از شروع کار اصلی حتماً کدهای موجود را از نظر الگوهای پیاده‌سازی و مستندسازی به‌دقت مطالعه و بررسی کنید.
۲. در بلوک‌های کد (مورد استفاده در دستورات if/else، switch و امثال آن) حتی اگر یک خط دستور هم داریم، باز از آکولاد باز و بسته استفاده کنید. بلوک‌های یک‌خطی را به هیچ وجه در همان خط دستور اصلی قرار ندهید. (مثال)
۳. هنگام استفاده از عملگر شرطی سه تایی (a < b ? f(x) : g(x)) برای بالا بردن خوانایی کد، ترجیحاً شرط اصلی را در یک خط، عبارت اول را در خط بعد و عبارت دوم را در خط بعد از آن قرار دهید. (مثال)
۴. در دستورات زنجیره‌ای (مانند دستورات کمکی Linq و سایر توابعی که زنجیروار قابل فراخوانی هستند) در صورت استفاده از دو تابع یا بیشتر، هر تابع را در خط جدید و با رعایت فرورفتگی قرار دهید. رعایت این قاعده، به افزایش خوانایی کدها کمک می‌کند. (مثال)
۵. هنگام ایجاد یک نمونه از یک کلاس و مقداردهی همزمان به ویژگی‌های آن، در صورتی که دو ویژگی یا بیشتر را مقداردهی می کنیم، حتماً هر ویژگی را در خط جداگانه مقدار می دهیم. (مثال)
۶. برای تمام کلاس‌ها، متدها، ویژگی‌ها و غیره، نوع دسترسی را همیشه به صورت صریح قید می‌کنیم (کلمات public، private و ...) دسترسی پیش‌فرض بین نسخه‌های مختلف پلتفرم دات نت ممکن است تغییر کند. به‌علاوه، قرار ندادن این کلمات به وضوح کد لطمه می‌زند.

۲,۳ قواعد مدیریت استثنائات (Exception Management)

۱. مدیریت استثنائات (exceptions) را تا حد امکان داخل کلاس‌های مختلف پخش نکرده و به صورت مرکزی انجام می‌دهیم. در صورتی که مدیریت یک یا چند استثنا واقعا ضروری است، آن را داخل سورس انجام می‌دهیم.

۲. هنگام مدیریت استثنائات، در صورتی که مشکل اصلی برطرف نمی‌شود، حتما از دستور throw استفاده کرده و هرگز exception را به حال خود رها نمی‌کنیم (اصطلاحا: آن را نمی‌بلعیم).

۲,۴ قواعد استفاده از خطوط خالی

۱. تمام خط‌های خالی اضافه را از داخل تعریف کلاس‌ها و متدها حذف می‌کنیم. برای جدا کردن بخش‌های مختلف پیاده‌سازی یک متد به منظور بالا بردن خوانایی، فقط یک خط خالی بین هر دو بخش فاصله می‌گذاریم.

۲. بین تعریف متدهای یک کلاس (از پایان یک متد تا شروع متد بعدی) فقط یک خط خالی فاصله می‌گذاریم.

۳. هنگام پیاده‌سازی کلاس‌ها و متدها و داخل هر بلوک چندخطی کد، بعد از آکولاد باز یا قبل از آکولاد بسته خط خالی قرار نمی‌دهیم.

۴. برای بالابردن خوانایی کد از خطوط خالی پی در پی استفاده نمی‌کنیم. (مثال)

۳. سایر قواعد (سرعت عملکرد، جلوگیری از خطای زمان اجرا و غیره)

۱. هنگام ساختن عبارات متنی که از ترکیب بخش‌های ثابت و متغیر ساخته می‌شوند، ترجیحاً از عملگر جمع استفاده نمی‌کنیم و متن‌های پیچیده را با متد String.Format یا بهتر از آن با کلاس StringBuilder می‌سازیم.

۲. سطوح مختلف ارثبری (inheritance) را در کلاس‌ها در سطح حداقل نگه داشته و برای پیاده‌سازی کلاس‌های پیچیده، بین ارثبری و ترکیب (composition)، تا حد امکان مکانیزم ترکیب را ترجیح می‌دهیم.

۴. قواعد مستندسازی

۱. هنگام نوشتن مستندات فنی (مستندات XML، مستندات داخل کد یا مستندات فنی دیگر) متن نهایی را حتماً کنترل می‌کنیم و خطاهای تایپی، املایی و نگارشی را برطرف می‌کنیم. مستندات بخش مهمی از امکانات نگهداری نرم‌افزار هستند و باید تا حد امکان واضح، دقیق و قابل فهم باشند.

۲. هنگام استفاده از متن یا کلمات انگلیسی (برای نامگذاری اجزاء سورس، تهیه متن‌ها و ریسورس‌های چندزبانه، توضیح کامیت یا موارد مشابه) از ابزارهای موجود Spell Checker استفاده می‌کنیم تا اشتباهات تایپی و املایی ایجاد نشود.
۳. مستندسازی‌های خارج از کلاس‌ها، متدها و ویژگی‌ها (properties) را تا حد امکان به زبان فارسی انجام می‌دهیم. برای مواقعی که اصطلاحات فنی نرم افزار معادل مناسب فارسی ندارند، می توانیم از متن انگلیسی استفاده کنیم.
۴. برای مستندات XML تا حد امکان کلمات فارسی و انگلیسی را با هم در توضیحات نمی‌گذاریم. ویژوال استودیو برای متن‌های راست به چپ داخل سورس امکانات کامل و خوبی ندارد و راستای متن کاملاً به هم می‌خورد.
۵. در صورت استفاده از زبان انگلیسی در مستندات XML، حتماً اولین کلمه توضیحات در ابتدای هر جمله را با حرف بزرگ شروع می‌کنیم و توضیحات را هرگز به کاراکتر اسلش (/) نمی‌چسبانیم.
۶. فقط در مواردی که کد موجود پیچیدگی منطقی دارد، از توضیحات (comment) استفاده می‌کنیم. در این موارد، سعی می‌کنیم به صورت دوره‌ای این توضیحات را بازنگری کنیم تا صحیح و به‌روز باشند.

۵. قواعد تست اولیه کد

(در دست تکمیل...)

۶. قواعد تکمیل و ارسال کار

۱. پیش از ارسال تغییرات به گیت لب، تمام کدهای آزمایشی یا بی‌ارتباط به کار اصلی را از پروژه حذف می‌کنیم.
۲. کدهای کامنت‌شده را فقط در صورتی درون سورس‌ها نگه می‌داریم که بعداً بخواهیم از آنها استفاده کنیم. حتی در این صورت نیز کدهای کامنت‌شده نباید مدت زیادی داخل سورس‌ها باقی بمانند. در چنین شرایطی، حتما دلیل کامنت کردن کد را به صورت مختصر و گویا در ابتدای کد توضیح می‌دهیم.
۳. پیش از ارسال تغییرات به گیت لب، تمام خطاهای کامپایلر را برطرف می‌کنیم.
۴. پیش از ارسال تغییرات به گیت لب، یکبار تمام پروژه‌ها را rebuild می‌کنیم و همه هشدارهای کامپایلر را برطرف می‌کنیم.
۵. برای هر کامیت، توضیح مختصر و گویایی می‌گذاریم که کار انجام شده را به طور خلاصه توضیح دهد.

نمونه‌های کد

بازگشت به قواعد

قاعده ۸-۱-۱

```
var intAsString = “1234”;
var parsed = int.Parse(intAsString);                                    (Wrong!)
var formatted = string.Format(“parsed = {0}”, intAsString);             (Wrong!)

var intAsString = “1234”;
var parsed = Int32.Parse(intAsString);                                  (Right!)
var formatted = String.Format(“parsed = {0}”, intAsString);             (Right!)
```

بازگشت به قواعد

قاعده ۱-۲-۱

```
int number=25;               (Wrong!)
decimal total =25.0M;        (Wrong!)
double subtotal    = 12.5;   (Wrong!)


int number = 25;             (Right!)
decimal total = 25.0M;       (Right!)
double subtotal = 12.5;      (Right!)
```

بازگشت به قواعد

قاعده ۷-۲-۱

```
// Wrong! (jagged code is sloppy and unreadable)
foreach (var item in items)
{
      // Doing something with item...
    item.Quantity += 10;
  item.UnitPrice -= 25.0M;
     item.Discontinued = false;
   repository.Save(item);
}
// Right! (code is neatly aligned)
foreach (var item in items)
{
    // Doing something with item...
    item.Quantity += 10;
    item.UnitPrice -= 25.0M;
    item.Discontinued = false;
    repository.Save(item);
}
```

قاعده ۸-۲-۱

```
// Wrong! (2 levels of indent)
foreach (var item in items)
{
        // Doing something with item...
        item.Quantity += 10;
        item.UnitPrice -= 25.0M;
        item.Discontinued = false;
        repository.Save(item);
}

// Right! (1 level of indent)
foreach (var item in items)
{
    // Doing something with item...
    item.Quantity += 10;
    item.UnitPrice -= 25.0M;
    item.Discontinued = false;
    repository.Save(item);
}
```

بازگشت به قواعد

قواعد نامگذاری

```
public class Badly_Named_Class : ...        (Wrong!)
public class badlyNamedClass : ...          (Wrong!)

public class WellNamedClass : ...           (Right!)

public int Increment(int _badName)          (Wrong!)
{
    int _badCopy = _badName + 1;            (Wrong!)
    return _badCopy;
}

public int Increment(int number)            (Right!)
{
    int incremented = number + 1;           (Right!)
    return incremented;
}
```

قاعده ۲-۲-۲

```
// Wrong!
if (!items.Any())
    return;

// Wrong! (Never do this!)
if (!items.Any()) return;

// Right! (Some code may be added to the “if” block in the future)
if (!items.Any())
{
    return;
}
```

بازگشت به قواعد

قاعده ۳-۲-۲

```
// Wrong!
var result = someCondition ? DoSomething(argValue) : DoSomethingElse(argValue);

// Right!
var result = someCondition
    ? DoSomething(argValue)
    : DoSomethingElse(argValue);
```

قاعده ۴-۲-۲

```
// Wrong! (potentially long line + difficult to read)
var items = await query.Where(i => i == id).Select(item => Mapper(item)).ToListAsync();

// Right! (easier to read)
var items = await query
    .Where(i => i == id)
    .Select(item => Mapper(item))
    .ToListAsync();
```

بازگشت به قواعد

قاعده ۵-۲-۲

```
// Wrong!
var rectangle = new Rectangle { x = 100, y = 200 };

// Right!
var rectangle = new Rectangle
{
    x = 100,
    y = 200
};
```

قواعد مرتبط با خطوط خالی

```
// Wrong!
if (items.Any())
{
                                    (No empty line after opening brace)
    // Do something with items...
                                    (No empty line before closing brace)
}
items.Clear();                      (Empty line after closing brace)

// Right!
if (items.Any())
{
    // Do something with items...
}

items.Clear();

// Wrong!
public int SomeMethod() { ... }             (One empty line after method’s closing brace)
public int AnotherMethod() { ... }          (ONLY one empty line after method’s closing brace)


public int YetAnotherMethod() { ... }

// Right!
public int SomeMethod() { ... }

public int AnotherMethod() { ... }

public int YetAnotherMethod() { ... }

// Wrong!
public void SomeMethod()
{
    // First part of method logic...
    // Second part of method logic...



    // Third part of method logic...


    // Last part of method logic...
}

// Right!
public void SomeMethod()
{
    // First part of method logic...

    // Second part of method logic...

    // Third part of method logic...

    // Last part of method logic...
}
```

تاریخچه تغییرات

ردیف	نسخه	تهیه کننده	تغییرات
۱	۱,۰	بابک اسلامیه	نسخه اولیه (پس از چند مرحله ویرایش و تکمیل)
۲	۱,۱	بابک اسلامیه	افزودن کدهای نمونه برای روشنی بیشتر قواعد استاندارد