

فهرست مطالب

مقدمه	۲
۱. قواعد کلی کدنویسی	۲
۱,۱ قواعد اولیه	۲
۱,۲ قواعد فاصله گذاری	۳
۱,۳ قواعد نامگذاری	۳
۱,۴ قواعد مربوط به using	۴
۲. قواعد سبک کدنویسی	۴
۲,۱ قواعد مدیریت و نگهداری پروژه ها در solution اصلی	۴
۲,۲ قواعد افزایش خوانایی کد	۴
۲,۳ قواعد مدیریت استثنائات (Exception Management)	۵
۲,۴ قواعد استفاده از خطوط خالی	۵
۳. سایر قواعد (سرعت عملکرد، جلوگیری از خطای زمان اجرا و غیره)	۶
۴. قواعد مستندسازی	۶
۵. قواعد تست اولیه کد	۶
۶. قواعد تکمیل و ارسال کار	۷

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 اعضای این پروژه ضروری است. نکاتی که به اشتباه یا به دلایل مختلف (مثل محدودیت زمانی برای تحویل به موقع) رعایت نشده باشند، در جلسات مرور کد به برنامه‌نویس گوشزد شده و هر کس موظف است ناهماهنگی‌ها و موارد رعایت نشده 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، به تدریج به موارد موجود اضافه می‌شوند. بنابراین برای کلیه اعضا 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. برای فرورفتگی‌ها (indenting) در سورس به هیچ‌وجه از Tab استفاده نمی‌کنیم.
۶. در هر فایل سی شارپ بیشتر از یک کلاس قرار نمی‌دهیم و نام فایل را با نام کلاس مربوطه یکسان در نظر می‌گیریم.
۷. 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.
۸. برای تعریف سطح دسترسی‌ها، تا حد امکان از سطح دسترسی داخلی (internal) استفاده نمی‌کنیم.

۱,۲ قواعد فاصله‌گذاری

۱. قبل و بعد از هر عملگر دوتایی حتماً یک فاصله می‌گذاریم.
۲. کاراکترهای خالی اضافه در ابتدای خطوط خالی و انتهای سایر خطوط کد یا توضیحات را حتماً حذف می‌کنیم.
۳. فرورفتگی کدها را با فواصل مضرب ۴ انجام می‌دهیم تا همراستایی (alignment) کدها به هم نخورد.

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 املائی 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. تمام پروژه‌ها و namespace های مربوط به محصول تدبیر را با پیشوند SPPC.Tadbir و تمام پروژه ها و namespace های مربوط به زیرساخت تدبیر را با پیشوند SPPC.Framework نامگذاری می کنیم.
۵. آدرس های اینترنتی (URL) را همیشه با حروف کوچک (lower-case) تعریف و استفاده می کنیم.
۶. هنگام نامگذاری کلاس‌ها، متدها، متغیرها و غیره از کاراکتر زیرخط (_) برای جدا کردن کلمات استفاده نمی کنیم.

۱,۴ قواعد مربوط به using

۱. تمام اعلانات مربوط به مارجول‌های مورد استفاده (دستورات using) خارج از namespace قرار می گیرند.
۲. دستورات using را همیشه به صورت مرتب‌شده نگهداری می کنیم. در مرتب‌سازی دستورات using، تمام مواردی که مربوط به پلتفرم اصلی دات نت هستند (و با کلمه System شروع می شوند) بدون درنظر گرفتن ترتیب الفبایی، همیشه در ابتدا قرار می گیرند.
۳. دستورات using که استفاده نشده‌اند را همیشه حذف می کنیم (دستورات using مربوط به System و System.Collections.Generic را به دلیل اینکه خیلی پایه‌ای و متداول هستند، می توانیم نگه داریم).
۴. بین دستورات using هیچ خط خالی قرار نمی دهیم. 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 پیش فرض) را یکسان درنظر می گیریم.
۲. اینترفیس‌ها و پیاده‌سازی پیش فرض آنها را - تا حد امکان - در یک پروژه و اسمبلی قرار نمی دهیم.
۳. اینترفیس‌ها را در یک پروژه و اسمبلی مرکزی تعریف کرده و با استفاده از فولدرها طبقه‌بندی می کنیم.
۴. تا حد امکان رفرنس‌های استفاده نشده و اضافی را از پروژه‌ها حذف می کنیم.

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 به دقت مطالعه و بررسی کنید.

۲. در بلوک‌های کد (مورد استفاده در دستورات `switch, if/else` 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.
۳. هنگام استفاده از عملگر شرطی سه تایی $(g(x) : f(x) < a)$ 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.
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.
۵. هنگام ایجاد یک نمونه از یک کلاس و مقداردهی همزمان به ویژگی‌های آن، در صورتی که دو ویژگی یا بیشتر را مقداردهی می‌کنیم، حتماً هر ویژگی را در خط جداگانه مقدار می‌دهیم.
۶. برای تمام کلاس‌ها، متدها، ویژگی‌ها و غیره، نوع دسترسی را همیشه به صورت صریح قید می‌کنیم (کلمات `private, public` و ...) دسترسی پیش‌فرض بین نسخه‌های مختلف پلتفرم دات نت ممکن است تغییر کند. به علاوه، 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 واقعاً ضروری است، آن را داخل سورس انجام می‌دهیم.
۲. هنگام مدیریت استثنائات، در صورتی که مشکل اصلی برطرف نمی‌شود، حتماً از دستور `throw` استفاده کرده و هرگز `exception` را به حال خود رها نمی‌کنیم (اصطلاحاً: 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 متدها، بعد از آکولاد باز یا قبل از آکولاد بسته خط خالی قرار نمی‌دهیم.
۴. برای بالا بردن خوانایی کد از خطوط خالی پی در پی استفاده نمی‌کنیم.

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، املائی 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 املائی 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 نرم‌افزار معادل مناسب فارسی ندارند، می‌توانیم 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.
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، حتماً 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، توضیحات مختصر و گویایی می‌گذاریم که کار انجام شده را به طور خلاصه توضیح دهد.