コーディング基準

株式会社三重県農協情報センター

改廃履歴

Ｒｅｖ	改 廃 内 容	実施日
1.0	新規作成	H23.01.26
2.0	全体の見直し	H27.01.01
3.0	全体の見直し	2022.10.01
4.0	セキュリティに配慮したコーディング原則の追加	2024.11.01

コーディング基準

規程番号　４００１－０００－０４－基
制 定 日　２０１１年　1月２６日
改 正 日　２０２４年１１月　１日

１．はじめに

本コーディング基準では、ソフトウェア開発プロジェクトにおいて一定の品質・保守性を担保する目的で、コーディング時のルール、指針を提供する。

２．原則

コーディングスタイルは、原則として主に使用するフレームワークのスタイルに従います。また、VisualStudio 等のコードエディタが提供する既定のフォーマットルールに従います。

既存システムを改修する場合は、既存システムで使用されているスタイルに従います。

３．命名規則

・パスカルケースを使用します。
・プライベートフィールド、ローカル変数、メソッドの引数にのみキャメルケースを使用します。
・各単語の最初の文字を大文字にします。
・コレクション型である場合や、戻り値がコレクションとなる場合は、複数形を使用します。
・単語は一般化しているものを除き、省略しません。
　○：AddCount　×：AddCnt　　○：UserID　○：UserIdentifier

・クラス（Class)、構造体（Structure）には名詞または名詞句で名前を付けます。
・インタフェース（Interface）には形容詞句または名詞・名詞句で名前を付けます。
　また、プレフィックスに“I”を付け、インタフェースであることを明示します。
・型パラメータは“T”または“T”で始まる名前を付けます。
・クラスメソッドには動詞または動詞句で名前を付けます。
・クラスプロパティには名詞または名詞句・形容詞で名前を付けます。
・プライベートフィールドには名詞または名詞句・形容詞で名前を付けます。
　また、プレフィックスに“_”を付加します。
・イベントには動詞または動詞句で名前を付け、時制を考慮して進行形や過去形を使用します。
・派生クラスは基底クラス名で終わることを検討してください。
・パラメータ名は説明的であり、意味に基づく名前の使用を検討してください。

４．レイアウト規則

・コードエディタの既定の設定を使用します。
　例えば、インデントにタブ文字は使用しません。
・１つの行には１つのステートメントのみを記述します。
・１つの行には１つの宣言のみ記述します。

５．コメント規則

・コメントはコード行の末尾ではなく、別の行に記載します。
・すべてのパブリックメンバーに、インテリセンスに必要なコメントを記載します。

C#

```
/// <summary>
/// 指定されたユーザ ID のユーザ氏名を取得します。
/// </summary>
/// <param name="userId">ユーザ ID</param>
/// <returns>ユーザ氏名</returns>
public string GetUserName(string userId)
{
    ...
}
```

・コメントは、コードに表せない意図や理由を残すことを基本とします。
　以下の例では、保存する前に意図して画像を変形（Mutate）していますが、コードから変形していることはわかっても、その理由がわかりません。

```
public async Task SaveUploadedImageAsync(Stream uploadedFileStream)
{
    using (var image = await Image.LoadAsync(uploadedFileStream))
    {
        // IE11 でアップロードされた画像を表示すると、EXIF 回転情報を
        // 無視して表示するため、保存前に画像をあらかじめ回転する。
        image.Mutate(x => x.AutoOrient());
        await image.SaveAsync("...");
```

・不要に見える可能性があるコードや必要があって冗長となっているコードにコメントを記載します。
　改修やリファクタの際に削除されないよう、コメントを残します。
・コードの意図を伝えるために必要であれば、根拠となるドキュメントやサイトを記述します。

・コード行（ステップ）を説明するコメントは記述しません。

```
//フォルダを作成する
System.IO.Directory.CreateDirectory("...");
```

・変数を説明するコメントは記述しません。コメントが必要ない変数名に変更します。

```
var c = 0;    //アクセスカウント
var accessCount = 0;
```

・変更履歴コメントは記述しません。

・不要なコード行のコメントアウトは残しません。その行を削除してください。

6．その他

・金額計算に浮動小数型を使用したことにより、計算誤差が発生し障害となったことがあります。

計算誤差が許容されない場合は、データ型に注意してください。

言語によって仕様が異なりますが、小数を含む数値リテラルは浮動小数型です。

C#の場合、サフィックスを付加して数値リテラルの型を明示できます。

端数が出る可能性がある計算処理は、以下の方針でコーディングします。

①乗算を優先して記述する。（演算子の優先順位に注意すること）

※以下の例では、1÷3＊3 の結果は 1 でない可能性がありますが、1＊3÷3 は常に 1 です。

②端数が出る計算と同時に、端数処理（切り上げ、切り捨て等）する。

③小数点以下の数値を格納する場合は、Decimal 型（真数型）を使用する。

C#

```
var d = 1.1;     //double 型
var d = 1.1d;    //double 型（d または D サフィックス）
                 //VisualBasic で D サフィックスは decimal なので注意
var f = 1.1f;    //float 型（f または F サフィックス）
var m = 1.1m;    //decimal 型（m または M サフィックス）
var s = "a";     //string 型
var c = "a"c;    //char 型（c または C サフィックス）

Console.Write($"1 ÷ 3 * 3 = {(1d / 3d) * 3d}");    // 1 ÷ 3 * 3 = 1
Console.Write($"1 ÷ 3 * 3 = {(1m / 3m) * 3m}");     // 1 ÷ 3 * 3 =
0.999999999999....
```

7．言語ガイドライン（VisualBasic、C#）

7．1．文字列

・短い文字列を連結するときは、文字列補間を使用します。

C#	VB
var name = "Taro"; var s = $"My Name is {name}";	Dim name = "Taro" Dim s = $"My Name is {name}"

・ループ内で文字列を大量に追加（結合）する場合は StringBuilder クラスを使用します。

7．2．暗黙の型指定（型推論）

・右辺によって変数の型が明らかである場合、厳密な型指定が必要でない場合は暗黙の型指定を使用します。

```
C#
var value = "Taro";                  //○　右辺は明確に String です
var value = 24;                      //○　右辺は明確に Integer です
var value = new ExampleClass();      //○　右辺は明確に ExampleClass です
var user = ExampleClass.GetUser();   //×　右辺からデータ型が明確ではありません
```

7．3．unsigned データ型

・どうしても必要な場合を除いて使用しません。

7．4．配列

・配列を初期化するときは、短い構文を使用します。

C#	VB
string[] values = {"a", "b", "c"};	Dim values As String() = {"a", "b", "c"}

7．5．例外処理（VisualBasic のみ）

・On Error Goto ステートメントは使用しません。

7．6．New キーワード

・オブジェクトをインスタンス化するときは、簡潔な形式を使用します。
また、オブジェクト初期化子を使用してオブジェクトの作成を簡素化します。

C#	VB
var x = new ExampleClass() { ID = 1 };	Dim x As New ExampleClass() With { ID = 1 }

7．7． ３項演算子（C#のみ）

・単純な if 文は３項演算子に置換することを検討します。

C#

```
bool result;
if(condition)
{
    result = true;
}
else
{
    result = false;
}
```

```
bool result = condition ? true : false;
```

7．8． NULL チェック

・NULL であることを確認するために if 文を使用せず、null 条件演算子や null 合体演算子（C#のみ）を使用します。

C#

```
if(obj != null)
{
    return obj.value;
}
else
{
    return string.empty;
}
```

```
return obj?.value ?? string.empty;
```

VB

```
If obj Is Nothing Then
    Return obj.value
Else
    Return Nothing
End If
```

```
Return obj?.value
```

7．9． Using の使用

・IDisposable インタフェースを実装したオブジェクトを扱うときは、必ず Using 句を使用します。

８．セキュア・プログラミング

システムやアプリケーションの脆弱性を事前に排除し、情報漏洩や乗っ取り、予期しないシステムダウンなどを防ぐため手法です。

脆弱性を作りこまないためのセキュア・プログラミングの原則を、設計時と実装時に分けて説明します。

なお、具体的なコーディングについては、ＩＰＡ、ＪＰＣＥＲＴ等から公表されているコーディング規約等を参照してください。

８．１．設計原則

(1)効率的なメカニズム

単純で小さな設計にします。

(2)フェイルセーフなデフォルト

必要ないものを排除するのではなく、必要なものを許すという判断を基本とします。

(3)完全な仲介

用意したセキュリティ機能がバイパスされないようにし、考え得るアクセス方法はすべてチェックします。

(4)オープンな設計

設計内容を秘密にしていることに頼らないようにします。

(5)権限の分離

アクセスは１つの条件より複数の条件で保護するほうが、強固で柔軟になります。

(6)最小限の権限

システムのすべてのプログラムおよびすべてのユーザは、業務を遂行するために最小の権限の組み合わせを使って操作することで、攻撃を受けたときの被害を最小限に抑えることができます。

(7)共通メカニズムの最小化

複数のユーザが共有し依存する仕組みの規模を最小限に押えます。

(8)心理学的受容性

ユーザおよび開発者が受け入れられる、簡単に使えるように設計します。

８．２．実装の原則

(1)入力を検証する。

すべての信頼されていないデータソースからの入力を検証します。

(2)他のシステムに送信するデータは無害化します。

外部に渡すデータは渡した先で問題を起こさないように加工します。

(3)コンパイラの警告を注視します。

(4)セキュリティポリシーの実現のためのプログラム構成／設計をおこないます。

守るべきものを特定してそれを守るためにおこなうものであり、すべてを同様に守るようにするものではありません。

(5)シンプルなプログラムを心がけます。

同じ結果を得られる実装は、ひとつとは限らず、複数の選択候補があるならば、できるだけシンプルなものを選択します。

(6)アクセス拒否をデフォルトにします。

許可ベースではなく、拒否ベースでアクセスを決定します。

(7)最小特権の原則を順守します。

どのプロセスも、実行するために必要な最低限の特権で実行します。

権限が昇格されている時間を最小限にし、攻撃者が昇格した権限で任意のコードを実行する機会を減らすことができます。

(8)多層防御を実践します。

根本的対策だけでなく、保険的対策も含めた異なるタイプの防御策をおこないます。

(9)品質保証技術を使用します。

優れた品質保証技術は、脆弱性を特定し、排除するのにも有効です。

Fuzzテスト、侵入テスト、およびソースコードの監査等の効果的な品質保証プログラムを必要に応じ使用します。

(10)セキュアコーディング標準を採用します。

ターゲット開発言語やプラットフォームのためのセキュアコーディング標準を適用し開発します。

９．参考

[Microsoft Docs - 名前付けのガイドライン]
https://docs.microsoft.com/ja-jp/dotnet/standard/design-guidelines/naming-guidelines

[Microsoft Docs - C#のコーディング規約]
https://docs.microsoft.com/ja-jp/dotnet/csharp/fundamentals/coding-style/coding-conventions

[Microsoft Docs - プログラム構造とコード規則 (Visual Basic)]
https://docs.microsoft.com/ja-jp/dotnet/visual-basic/programming-guide/program-structure/program-structure-and-code-conventions

[Stack Overflow Blog - Best practices for writing code comments]
https://stackoverflow.blog/2021/12/23/best-practices-for-writing-code-comments/

[ＩＰＡ セキュア・プログラミング講座]
https://www/ipa.go.jp/archive/security/vuln/programming/index.html

[ＩＰＡ　安全なウェブサイトの作り方]
https://www.ipa.go.jp/security/vuln/websecurity.html

[ＪＰＣＥＲＴ／ＣＣ　セキュアコーディング (JPCERT/CC)]
https://www.jpcert.or.jp/securecoding/