**疑似開発コーディング規約**

**岡部　宜樹**

目次

1.コーディングの心得 五か条

　◇見やすさを重視せよ

　◇ネーミングはわかりやすく

　◇サンプルを鵜呑みにしない

　◇同じコードは二度書かない

　◇役割は一つに

2.ネーミング規約

　2.1　全般

　　◇英語を使う

　　◇大文字・小文字で名前を区別しない

　2.2　クラス名

　　◇クラス名はActivityかFragmentか分かる名前にする

　　　例）mainActivity.java activity\_main.xml resultFragment.java fragment\_result.xml

　　　　 takeQuizPackActivity.java activity\_takeQuizPack.xml

　2.3　メソッド

　　◇メソッド名は区切りのみ大文字にする

　　◇オブジェトを作るメソッド名は[“create”+オブジェクト名]にする

　　◇変換メソッド名は[“to”+オブジェクト名]にする

　　◇ゲッターメソッド名は[“get”+変数名]にする

　　◇セッターメソッド名は[set+変数名]にする

　2.4　変数全般

　　◇定数はすべて大文字で区切りは”\_”

　　◇変数名に役割の反映を

　2.5ローカル変数

　　◇スコープが狭い変数名は省略した名前でもよい

　　◇for文のループカウンタは、ネストごとに”i”,”j”,”k”・・・を使う

3コーディング規約

　3.1　全般

　　◇使われないコードは書かない

　3.2　フォーマット

　　◇空白文字を使わずに、タブを利用する

　　◇長すぎる行は程よい箇所で改行する

　　◇return文ではカッコを使わない

　3.3　コメント

　　◇メソッドやクラスの上に何をするものなのかがわかるコメントを入れる

　　◇コメントはDoxygenを参考に記入する　※6Pに詳しく書いてあります

　3.4　メソッド

　　◇メソッドの役割は一つにする

　3.5　変数全般

　　◇配列宣言は[型名[]]にする

　　◇できるだけローカル変数を利用する

　3.6　ローカル変数

　　◇ローカル変数を安易に再利用しない

　3.7　制御構造

　　◇制御文(if,else,while,for,do)の”{“は省略しない

　　◇for文を利用した繰り返し処理中でループ変数の値を変更しない

　　◇繰り返し処理中のオブジェクトの生成は最小限にする

4　ファイルの扱い

　　◇必要なファイル(パック情報やクイズなどの情報)は内部フォルダに入れる

◇見やすさを重視せよ

「良いコード」の基本は、「他の人が読んでもわかりやすいと感じられるコード」と言えます。コードの見やすさは、フォーマットはもちろん、ロジックの簡潔さや API の常識的な使い方などから生まれます。コーディングにあたっては、常に他の人の視点を意識しながら、見やすさに気を配って記述しましょう。 また、自分で記述したコードであっても、しばらくたってから読み返してみると理解に時間がかかった経験はないでしょうか。「3 日前に書いたコードは他人のコードと同じ」ということも言われます。見やすさを重視することは、他の人のためだけでなく、自分のためにもなります。

◇ネーミングはわかりやすく

コーディングでは、様々な変数やメソッドなどにネーミング(名前付け)しなければなりません。ネーミングとは、本来、その対象の本質を表すような名前を考える作業です。大変難易度の高い作業ですが、一方で適当に行ってもコードの動作は変わらないため、人によっては手を抜きがちです。しかし、ネーミングの良し悪しは、コードの可読性に非常に大きな影響を及ぼします。 例えば、「C0001」というクラス名があるとします。これでは、何を表すクラスなのかすぐにわかりません。また、「int p = 5000;」という記述があるとします。プログラマに聞くと、変数名pは価格(Price)の略だと言ったとします。であれば略さずに、「int price = 5000;」としたほうがわかりやすいと思いませんか？ 「ネーミングはわかりやすく」の背景には、読んで内容が理解できるという意味で、文章のようなプログラミングを行うという考え方があります。

◇サンプルを鵜呑みにしない

サンプルコードを活用すること自体は、著作権等を侵害しなければ問題ないでしょう。問題なのは、その内容や背景を理解しないまま、サンプルコードだけを鵜呑みにして、「おまじない」として表面的に適用してしまうことです。 コードを「おまじない」ととらえていては、サンプルコードの間違いを気づかないまま適用してしまうこともあります。 例えば、ストリームのクローズ処理を行っていないサンプルコードであっても、それに気づかずに自分のコードに適用してしまい、後で思わぬ障害を引き起こすということもありえます。サンプルコードは、そこで説明する内容に絞ったコードが多いため、このような例はよく見られます。 また、サンプルコードをそのまま適用した結果、自分が記述すべきコードには必要のないコードが含まれてしまうこともあります。その場合、コードの可読性を下げる原因になります。 自分のコードは、自分で深く理解して記述しましょう。

何よりもこの疑似開発は自分たちの技術を高めるものなので、「おまじない」と思って分からないままでいては成長できません

◇同じコードを二度書かない

コードをコピー・ペーストしていませんか？コピー・ペーストしてしまうと、何らかの修正をする際に、全ての個所に同じ修正をする羽目になります。同じコードが現れるようならまとめて一つにし、外に出してコールするような書き方をすべきです。 同じコードをまとめる作業は、どちらかといえば、コーディング時よりリファクタリング(ソフトウェアの外部的振る舞いを変更せずに内部構造を改善する作業)で行われることが多いでしょう。しかし、コーディング時からできるだけ気をつけておきたいことです。

役割は一つに

メソッドの役割が明確で、かつ1つであれば単体テストが行いやすくなります。つまり、コードの「試験性」が高まります。また、役割が一つであれば、後でコードを変更する際に修正箇所がわかりやすいため、障害修正に要する時間が短くなります。つまり、コードの「保守性」があがります。 例えば、「チェックをして実行する」機能を実現するために、checkAndDo()メソッドが存在したとします。このメソッドはcheck()メソッドとdo()メソッドに分割すべきです。なぜなら、checkAndDo()メソッドのチェックロジックに誤りがあった場合、本来は do()メソッドに書かれる内容まで把握する必要があるからです。分割してあれば、check()メソッドだけの変更で済みます。 このことはクラスの設計にも言えることです。

□Doxygen 書き方メモ

・基本的な書き方

/\*

\*ここにコメントを書く

\*/

・関数に対するコメント

　/\*\*

　\* @fn

　\* ここに関数の説明を書く

\* @brief 要約説明

　\* @param (引数名) 引数の説明

　\* @return 戻り値の説明

\* @sa 参照すべき関数を書けばリンクが貼れる

\* @detail 詳細な説明

　\*/

・変数に対するコメント

/\*

\*変数へのコメント

\*/