第2章. コメントを書こう

読みやすいプログラムを書くには、コメントを書くことが大切です。
もちろん、コメントがなくても十分読みやすいプログラムを書くべきではありますが、 それでもソースコードの要約やその処理の意図などを明確に緑色の日本語の文字で説明しておくべきです。 そうすることで、ロジックを解析する手間が省けたり、ソースコードを読んでもすぐには理解できないような情報を読む人に伝えたりすることができます。
この章では、ソースコードのどこに、どのようなコメントを書くべきかを説明します。

2－1. ソース看板コメントを記述しよう

各モジュールのソースコードの先頭部分の、package と import の間には、ファイル名や作成日等を記述するコメントを書きましょう。
作成日や変更日、変更内容を記録することにより、ソースコードを読まなくても修正箇所や変更箇所を把握することができます。 通常は、開発現場ごとに、記述する内容や形式が決まっていることが多いです。

ソース看板コメント記述例

```
package sample.pkg;
/*============================================================================*
* システム：顧客情報管理
* ファイル：SampleClass.java
*
* Copyright (C) JobSupport Co. Ltd. All Rights Reserved.
* ----------------------------------------------------------------------------
* 〔変更履歴〕
* yyyy/mm/dd T.Yamada 新規作成
* yyyy/mm/dd T.Yamada xxの条件の時の更新ボタンの挙動を修正
* yyyy/mm/dd T.Yamada 更新処理時の更新対象カラムを変更
*============================================================================*/
import java.util.…;
                :

public class sample {
                :
```

・変更履歴の新規作成の行は、ファイル作成時に記述する。
・変更履歴の修正の行(2行目以降)は、コードレビュー(インストラクタチェック)・UT(単体テスト)・IT(結合テスト)完了後の修正履歴を列挙していく。
(※製造作業時の修正履歴は記入しないこと)

2－2. javadoc コメントを記述しよう

javadoc とは、クラス・フィールド・メソッドの説明時に使用するドキュメンテーションコメントのことを指します。

自作したクラスの先頭部分には、そのクラスの概要や作成者等を説明するコメントを書きましょう。
また、自作したメソッドの先頭部分には、その処理の概要や戻り値の内容等を説明するコメントを書きましょう。
説明を記述することにより、ソースコードを読まなくてもそのクラスやメソッドの役割・機能や使い方を把握

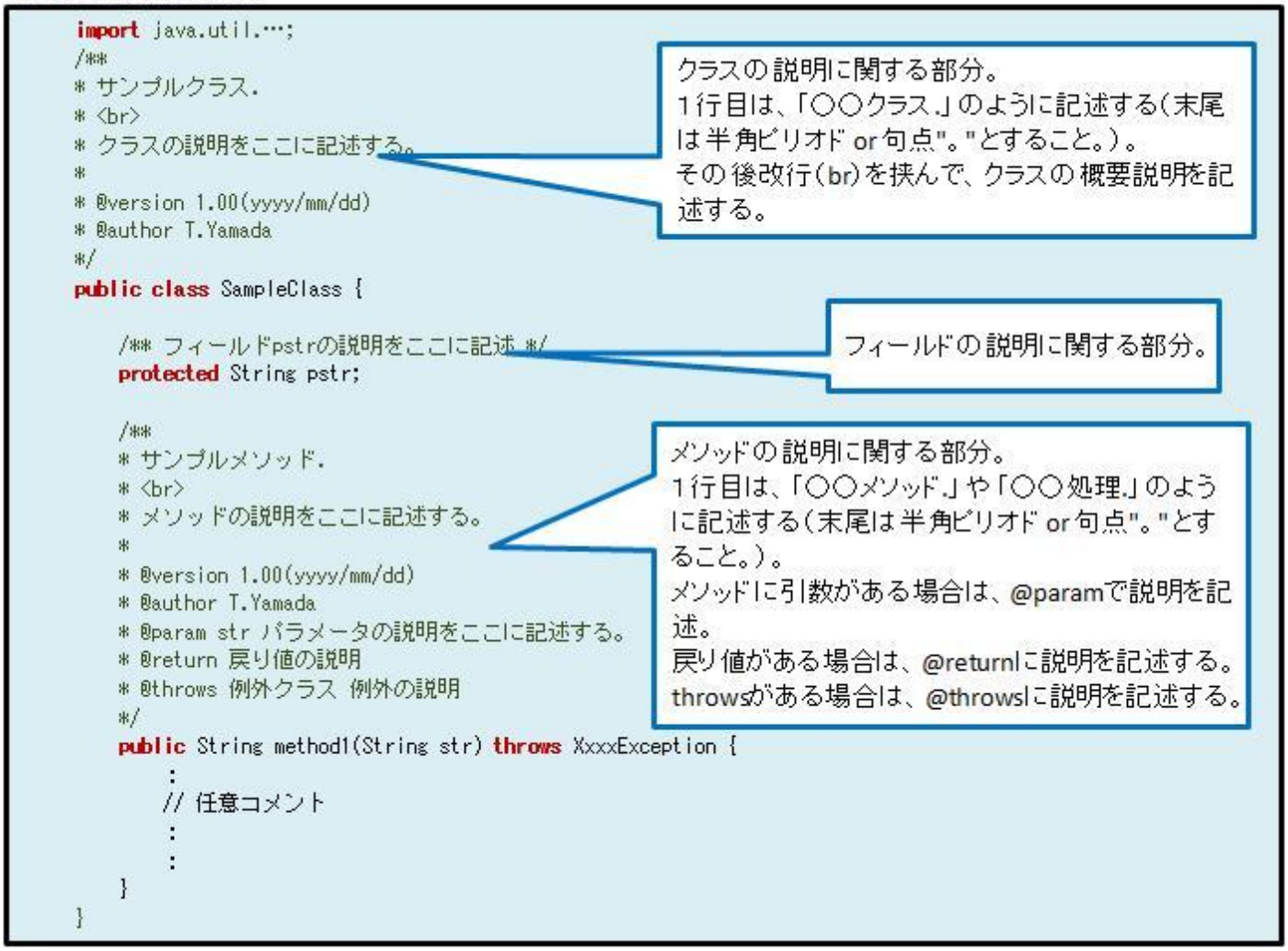
することができます　通常は、開発現場ごとに、関数の説明として記述する内容や形式が決まっていることが多いです。

javadoc 内では、コメントを記入する際にクラスやメソッドの役割を示す為のタグを使用します。

以下のタグは代表例です。

@version ： バージョンを記述します。
@author　： クラスやメソッドの作成者名を記述します。
@param　： メソッドの引数の説明を記述します。
@return　： メソッドの戻り値の説明を記述します。
@throws　： throws がある場合、例外クラスの説明を記述します。

javadocコメント記述例



2－3. 特殊な変数は、用途を説明しよう

変数のスコープが狭い場合は、適切な変数名さえつけてあれば変数に対するコメントなどは特に必要としません。しかし、例えばフィールドなど、変数のスコープが広い場合や、保持する値が特殊な変数の場合は、その用途や保持する値の内容をコメントで説明しましょう。

(1) スコープが広い変数

フィールドなど、複数のクラスやメソッドから参照される変数は、どのような使われ方をするのか、どのような値を保持するのかを説明しましょう。

ローカル変数はそのメソッド内のみで使用されるので、そのメソッドのソースコードを読めばどのように使っているのかが分かります。しかし、フィールドは、まずどのクラスで使っているのかを調べて、該当したそれぞれのクラスのソースコードを読まないとどのように使っているのかが分かりません。場合によってはそれぞれのクラスがどのタイミングでどこから呼ばれるクラスなのか、ということまで調べないと役割が理解できないこともあります。

このような手間を少しでも軽減するため、ソースコードを読む人の理解の助けになるよう、スコープの広い変数には説明コメントを書きましょう。

(2) データ構造に関わるような重要な変数

プログラムでデータをどのように扱うか、その方法に大きく影響する変数には、詳細な説明を記述しましょう。

プログラムは、データを処理するためにあります。

データの集まりを保持する形式のことをデータ構造と呼び、配列やリスト、ハッシュテーブル、スタック、キューなど、さまざまな種類があります。どのデータ構造を使用するかによって、プログラムの処理の仕方に大きく影響します。

よって、データ構造が把握できれば、そのプログラムでどのような処理が行われているかを自ずと予想できたりすることがあります。そのため、データ構造そのもの、もしくはそのデータ構造に関わる変数に対しては、データの保持の仕方や保持するデータの内容などの詳細な説明を記述しましょう。

(3) フラグのような特殊な使われ方をする変数

通常の正常でストレートなロジックを乱してしまうような、例えばフラグのような変数には、どのように扱われる変数なのかを説明しましょう。

フラグのような変数は、プログラムのどこでどんな値を設定しているのか、どこで参照しているのかが分かりづらくなることがあるため、プログラムが難解になってしまいます。そのような変数はできるだけ使用しないようにするべきですが、やむを得ず使用せざるを得ない時もあります。その時は、どのような役割を持った変数なのか、必要であれば保持する値によって プログラムの挙動がどう変化するのかといったことまで説明を記述しましょう。

2－4. ブロックの先頭に概要説明を記述しよう

if や for/while、switch 等のブロックの先頭には、そのブロックで行おうとしている処理の概要を説明するコメントを記述しましょう。

ブロックの先頭に説明を記述することにより、そのブロック内のソースコードを読まなくてもそこで行おうとしている処理の概要が把握でき、ソースコードをかいつまんで読む際にとても役に立ちます。
処理内容を要約するという役割において、コメントの効果は絶大です。

・if文のコメント例

```
// xxxxxをチェックする
if (checkData(data) != CHECK_OK) {
    :
    return -1
}


// xxxの場合のみ、yyy処理を行う
if (ret == 0) {
    :
    :
}
```

・for、while文のコメント例

```
// 表の全行分のデータを出力する
for (row = 0; row < ROW_MAX; row++) {
    :
    :
}


// 等差数列の値が100になるまで表示を繰り返す
while (num <= 100) {
    :
    System.out.println(num);
    :
    num += add;
}
```

・switch文のコメント例

```
// 入力されたメニュー番号により、各画面を表示
switch (menuNo) {
case MENU_REGISTRY:
    :
    :
    break;
case MENU_LIST:
    :
    :
    break;
case MENU_DELETE:
    :
    :
    break;
}
```

2－5. 特殊な処理、際どい処理には、理由や意図を書こう

プログラムは読みやすさを考慮して、できるだけ明快かつ自然な流れで、安全で安定した処理になるよう記述するべきですが、ときにはどうしても複雑で際どい処理を書かなければならない場面があります。
そのときは、後々他人が、または未来の自分がそのプログラムを読んだ時に、「なぜこんな複雑な処理を行っているのだろうか？」と困惑してしまわないよう、きちんと説明を記述しておく必要があります。

具体的には、記述したプログラムを読み返したときに、以下のようなイレギュラーな処理内容が見つかった場合、その理由や意図をコメントとして記述しましょう。

- 変則的な処理。（例） 通常行わなくてもいいような処理を、わざわざ行っている。
- 不順な処理。　（例） 通常の処理手順からするとそこで行うべきではない処理を、あえてそこで行っている。
- 特殊な処理。　（例） データの単純な参照を行わず、文字列の部分切り出し等の特別な加工をしている。
- 複雑な制御。　（例） 処理を行うかどうかの制御や条件が複雑、または多い。
- 複雑な処理。　（例） 複雑なアルゴリズムを採用している。

もし、コメントを記述しようとして説明内容を考えたときに明確でもっともらしい理由や意図が思いつかないようであれば、そのプログラムのロジックに問題があるのかもしれません。一度プログラムの処理内容を見直し、適切なロジックになるようコーディングし直してください。

コメントを記述する作業は、自分の頭の中を整理し直す作業でもあります。 この処理は何か？ 何のための処理か？ なぜこの位置で行っているのか？ コメントは要るか？ というようなことを考えることにより、時にはそのプログラムの不必要に複雑になっている部分や無駄な記述に気付いたりすることがあります。それを修正することで、より読みやすくて無駄のないプログラムに近づくことができます。

このように、コメントを書くことはプログラムの保守性を向上させ、 更に自身のコーディングレベルの向上にもつながりますので、面倒くさがらず、しっかり記述するようにしましょう。

２－６. 処理のひとまとまりに、目印になる要約コメントを書こう

プログラムは読みやすさを考慮して、メソッドの処理が長くなりすぎないようにメソッドを分割したり する必要がありますが、ときにはどうしても長い処理を書かなければならない場面があります。

そのときは、長い処理の中でも関連する処理ごとに空行で記述エリアを分け、 そのひとまとまりに目印になる要約コメントを記述しておくと、長い処理でもだいぶ流れが追いやすくなります。

以下に、要約コメントの記述例を示します。

要約コメントの記述例

```
void func (int aaaaa, int bbbbb) {
    int zzzzz;

    //---------------------------------
    // XXXXX情報を取得する
    //---------------------------------
    XXXXXXXXXX;
    XXXXXXXXXXXXX;

    // xxxxxxxxxxxxx
    if (XXXXXXXXXXX > XXXXX) {
        XXXXXXXXX;
        XXXXXXXXXX;
    }

    //---------------------------------
    // YYYYY情報を取得する
    //---------------------------------
    YYYYYYYYYY;
    YYYYYYYYYYYYYYY;

    // yyyyyyyyyyyyyy
    if (YYYYYYYYYY > YYYYY) {
        YYYYYYYYY;
        YYYYYYYYYY;
    }

    //---------------------------------
    // XXXXXXX情報とYYYYY情報から、ZZZZZを算出する
    //---------------------------------
    for (ZZZZZ; ZZZZZ; ZZZZZ) {
        ZZZZZZZZZZ;

        // zzzzzzzzzzzz
        if (ZZZZZZZZZ == ZZZ) {
            ZZZZZZ;
        } else {
            ZZZZZZZZZZZZ;
            ZZZZZZZ;
        }
    }

    //---------------------------------
    // ZZZZZをFFFFFFFファイルに出力する
    //---------------------------------
    FFFFFFFFFFFFF;
    if (FFFFFFFF == FFFFF) {
        FFFFFFFF;
    }

    // fffffffffff
    for (FFFFF; FFFFF; FFFFF) {
        FFFFFFFF;
        FFFFFFFFFF;
    }
}
```

・要約コメントを拾い読みするだけで、処理の流れが把握できる。
・その関数のどこで何を行っているかが分かる。

目立つコメントは、書きすぎに注意

上述のように目印になるような目立つコメントは、要所要所に記述すると長いプログラムの処理が追いやすくなってとても役に立ちますが、記述する際は書きすぎないように注意してください。
本当はそれほど長くない処理だったり目立たせる必要がなかったりする場面で、何でもかんでも目立つコメントを書いてしまうと、処理の記述が埋もれてしまい逆に読みづらくなってしまうことがあります。
また、コメントに頼りすぎず、「うまくメソッド分割ができないか？」等、簡潔に記述する方法もきちんと検討しましょう。

2－7. 無駄なことを書かず、プログラムの理解を助けるためのコメントを書こう

コメントには、当たり前すぎることは書かず、プログラムを読む人の理解を助けるための説明を記述しましょう。

日本語での説明のほうが分かりやすいからと言って、全ての処理一行一行に対してコメントを書く必要はありません。何でもかんでもコメントを記述すると、処理の部分が埋もれてしまい、逆に読みづらくなります。また、本当に注目すべき説明文が目立たなくなってしまい、せっかく書いた注釈としてのコメントの効果が薄くなってしまうこともあります。

コメントを記述する際は、そのソースコードを他人が読んだ時にすんなり理解できるか？　そのままでは誤解や混乱を招いてしまうようなロジックになっていないか？　というようなことを考え、コメントの要／不要を判断するようにしてください。
当たり前すぎて説明する必要のない処理にまでコメントを書く必要はありません。「これは説明しておかないとコードを見てもすぐには理解できないだろう」と思われる箇所にコメントを書きましょう。

×：無駄なコメントが多いソースコードの例

```
/**
* 等差数列.
* <br>
* 等差数列を表示するメソッド。
*
* @version 1.00(yyyy/mm/dd)
* @author T.Yamada
* @param startVal 開始値。
* @param endVal 終了値。
* @param zobun 増分値。
*/
public void printSequenceOfNum(int startVal, int endVal, int zobun) {
    int outputCnt = 0; // 出力回数を数える変数

    // 等差数列を表示
    for (int i = startVal; i <= endVal; i += zobun) {
        // 3桁分の幅を指定し、各数字をスペースを区切って表示する
        System.out.printf("%3d ", i);

        // 出力回数を数える変数をインクリメントする
        outputCnt++;

        // 出力回数を10で割った時に割り切れた場合
        if ((outputCnt % 10) == 0) {
            // 改行を出力する
            System.out.println();
        }
    }
}
```

出力回数のカウンタであることは、名前から明らかなので、それ以上の役割がなければコメント不要。

ソースコードを見れば明らかなことをコメントで説明する必要はない。

コメントが多い割に10個出力するごとに改行するという意図が伝わりづらい。

○：上記ソースコードの冗長なコメントを排除した例

```
/**
* 等差数列.
* <br>
* 等差数列を表示するメソッド。
*
* @version 1.00(yyyy/mm/dd)
* @author T.Yamada
* @param startVal 開始値。
* @param endVal 終了値。
* @param zobun 増分値。
*/
public void printSequenceOfNum(int startVal, int endVal, int zobun) {
    int outputCnt = 0;

    // 等差数列を表示
    for (int i = startVal; i <= endVal; i += zobun) {
        System.out.printf("%3d ", i);

        // 10個出力するごとに改行する
        outputCnt++;
        if ((outputCnt % 10) == 0) {
            System.out.println();
        }
    }
}
```

・10個出力するごとに改行するという意図が伝わる。
・outputCnt変数のインクリメントと10で割って判定する記述がまとまっているので、何のための変数か、何のための処理かが分かりやすい。(参照範囲の局所化)