第3章. 適切な名前をつけよう

読みやすいプログラムを書くには、変数やクラス、メソッド、その他において、適切な名前をつけることが大切です。
名は体を表します。 適切な名前がつけられていれば、その変数の役割や関数の使い方が予測できるため、プログラムが読みやすくなります。
この章では、変数やクラス、メソッド等に対する名前の付け方について説明します。

3－1. 変数の役割を明確に表す名前をつけよう

変数には、その変数が持つ役割を明確に表す名前をつけましょう。

適切な名前がつけられた変数を使用したソースコードからは、「こういう計算を行う」 とか 「この値を求めたい」 というようなプログラマの意図を読み取ることができます。
逆に、変数に適切な名前がつけられていない場合、一目見ただけではその処理の目的を理解することができません。 前後の処理内容から推察したりするなどの解析が必要になるため、とても難解に感じてしまいます。

考えるのが面倒くさいからといって、「a」や「b」のような意味のないアルファベット1文字の名前をつけたりしてはいけません。
もし、変数につける名前を考えたときに適切な名前が思いつかなかったら、 その変数の役割を十分に把握できていない可能性があります。 一度頭の中を整理し、実現したい処理とそのためにどのような役割の変数が必要であるかを考えて、 それぞれの変数に適切な名前をつけるようにしましょう。

また、変数と同様にメソッドの引数にも適切な名前をつけるようにしましょう。 引数は、その引数が呼び出し元からどのような値をもらうのかが明確に分かるような名前をつけてください。

×：変数、引数に適切な名前がつけられていないソースコード

```
/*
*指定された長さになるようにコピー元文字列の
*左側の文字埋めしてコピー先文字列に格納する。
*（例）
*     padCharLeft(dest, "abc", '*', 5);
*     →処理結果としてdestに"**abc"が格納される。
*/
String padCharLeft(char[] a, char[] b, char c, int l) {

    int k = b.length;

    // コピー元文字列が、指定された文字埋め後文字列長を
    // 超えている場合は、文字列のコピーだけして終了
    if (k > l) {
        for (int i = 0; i < b.length; i++) {
            a[i] = b[i];
        }
        return a;
    }

    // 指定された長さの文字列を格納する配列に文字列部分をコピー
    char[] aa = new char[l];
    for (int i = 0; i < a.length; i++) {
        aa[i] = a[i];
    }

    // 左側の文字埋めを実行
    int j = l - k;
    for (int i = 0; i < j; i++) {
        aa[i] = c;
    }

    // 文字埋め直後の位置から文字列部分をコピー
    for (int i = 0; i < b.length; i++) {
        aa[j++] = b[i];
    }
    return aa;
}
```

引数がどのような値を保持するのかを、引数名から推測することが難しい。

○：変数、引数に適切な名前がつけられているソースコード

```
/*
*指定された長さになるようにコピー元文字列の
*左側の文字埋めしてコピー先文字列に格納する。
*（例）
*     padCharLeft(dest, "abc", '*', 5);
*     →処理結果としてdestに"**abc"が格納される。
*/
String padCharLeft(char[] destStr, char[] srcStr, char padChar, int totalLen) {

    int srcLen = srcStr.length;

    // コピー元文字列が、指定された文字埋め後文字列長を
    // 超えている場合は、文字列のコピーだけして終了
    if (srcLen  > totalLen) {
        for (int i = 0; i < srcStr.length; i++) {
            destStr[i] = srcStr[i];
        }
        return destStr;
    }

    // 指定された長さの文字列を格納する配列に文字列部分をコピー
    char[] destStoreStr = new char[totalLen];
    for (int i = 0; i < destStr.length; i++) {
        destStoreStr[i] = destStr[i];
    }

    // 左側の文字埋めを実行
    int padLen = totalLen - srcLen;
    for (int i = 0; i < padLen; i++) {
        destStoreStr[i] =  padChar;
    }

    // 文字埋め直後の位置から文字列部分をコピー
    for (int i = 0; i < srcStr.length; i++) {
        destStoreStr[padLen++] = srcStr[i];
    }
    return destStoreStr;
}
```

引数がどのような値を保持するのかイメージできる。

・その変数がどのような値を保持するのか、何の役割をするのかをイメージできる。
・処理を見たときに何をしているのか理解しやすい。

3－2. クラスやメソッドの役割や処理内容を表す名前をつけよう

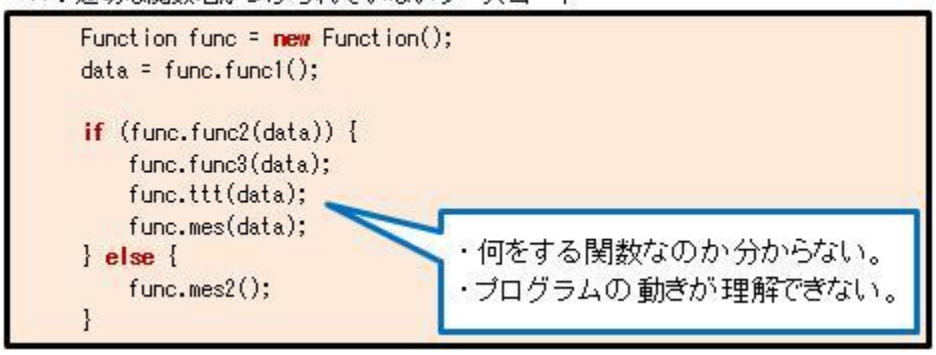
クラスには、そのクラスが行なう役割の内容を表す名前をつけましょう。
またメソッドには、そのメソッドが行う処理の内容を表す名前をつけましょう。

変数と同様、クラスやメソッドに適切な名前がつけられている場合は、そのクラスやメソッドの呼び出し箇所を見たときに「ここでこの処理を行う」とか「この値を取得する」というようなプログラマの意図を読み取ることができます。

逆に、クラスやメソッドに適切な名前がつけられていない場合、そのクラスやメソッドの呼び出しによって何が実行されるのかが分からなくなります。そのクラスやメソッドのソースコードを解析したりしないとプログラムの動きを把握することができないため、とても読みにくく感じてしまいます。

また、メソッドは、「～～処理を行う」「～～の値を取得する」というように、プログラムが目的の動作を実行するものなので、一般的に動詞から始まる名前をつけるとしっくりくることが多いようです。

×：適切な関数名がつけられていないソースコード



○：適切な関数名がつけられているソースコード

アクセスメソッド（アクセサ）は、get・set から始まる先頭を大文字に変更したインスタンスフィールド名を使用します。

アクセスメソッド

```
・ゲッタ
public String getMessage() {
    return message;

・セッタ
public void setMessage(String message) {
    this.message = message;
}
```

そして、クラスやメソッドの名前をつける際は、以下のような英語の対称性に気をつけましょう。

以下の英語は代表例です。

```
add/remove up/down          get/set          source/target
insert/delete      show/hide      start/stop      open/close
```

3－3. 慣習的な名前をつけよう － ループカウンタ

ループ処理で使用するループカウンタには、ｉ，ｊ，ｋ，…といった名前のint型変数がよく使用されます。

そのループ処理が配列要素を順番に参照する等、単純にカウンタを進めるだけのループであれば、このような慣習的な「i」などの名前で扱ったほうがよいです。そうすることで、他人が読んだ時にもその変数がループカウンタであることが暗黙的に伝わるため読みやすくなります。

慣用的なループカウンタ名の例（i，j）

```
for (int i = 0; i < array.length; i++) {
    System.out.printf("array[%d]:%d¥n", i, array[i]);
}

// 配列の要素を昇順に並べる（選択ソート）
for (int i = 0; i < array.length; i++) {
    int minIdx = i;

    // i番目以降で最小のデータを探す
    for (int j = i + 1; j < nums.length; j++) {
        if (nums[j] < nums[minIdx]) {
            minIdx = j;
        }
    }

    // 最小データをi番目に移動
    if (minIdx > i) {
        int tmp = nums[i];
        nums[i] = nums[minIdx];
        nums[minIdx] = tmp;
    }
}
```

単純なループの場合は、ループカウンタ名はiでOK。

・単純なループの場合は、ループカウンタ名はi,jでOK。
・多重ループは、iから始まり、j,k･･･と順番に使用する。

ただし、特に多重ループにおいてそのループカウンタが特別な意味を担っていたりする場合はｉやｊではなく、適切な変数名をつけるべきです。例えば画像処理を行う際、ピクセル情報をｘとｙの座標値で扱ったりしますが、その場合はｉ，ｊではなく、ｘ，ｙといったループカウンタ名のほうが意味が伝わりやすくなります。

ループカウンタが特別な意味を持っている場合の例

```
// 全ピクセルデータを参照
for (int x = 0; x < width; x++) {
    for (int y = 0; y < height; y++) {
        int pixel = getPixel(x, y);
        :
        :
    }
}


// 全セルデータを参照
for (int rowIdx = 0; rowIdx < rowMax; rowIdx++) {
    TableRow row = tableRow.getTableRow(rowIdx);

    for (int colIdx = 0; colIdx < colMax; colIdx++) {
        CellData cell = cellData.getCellData(row, colIdx);
        System.out.printf("セルの値・・・%d¥n", cell);
        :
        :
    }
}
```

ピクセルデータを参照する等、座標を扱う場合は座標値x,yをそのままループカウンタ名にすると分かりやすい。

表形式のようなデータを参照するときは、テーブルの行・列に合わせてrowIdx, colIdxのようなループカウンタ名にすると分かりやすい。

3－4. スコープが十分に狭い場合には短い変数名を使おう

変数にはその変数の役割にあった名前をつけるべきですが、ときにはより短い名前をつけるほうが望ましい場面があります。

数行程度のメソッドや短いループ処理など、処理内容が一目で見渡せるくらい十分に狭いスコープで使用される一時変数には、そこまで立派な名前は必要ありません。処理が短ければどのような処理が行われているかは容易に理解できる場合が多いので、役割をきちんと表すようにそれなりの長さの名前をつけるよりも、ある程度簡略化した短い文字の名前をつけたほうが視界の邪魔にならず読みやすくなります。

狭いスコープで使用される変数の名前の例

```
/*
 *引数に渡された文字列が数字列であるか否かをチェックします。
 *全ての文字が0～9の文字であれば1を返却します。
 *1文字でも0～9以外の文字が含まれていれば、数字列でないものとし、0を返却します。
 */
public int isNumeric(char[] str) {

    for (int i = 0; str[i] != '¥0'; i++) {
        char c = str[i];
        if (c < '0' || c > '9') {
            return 0;
        }
    }

    return 1;
}
```

文字列を1文字ずつ参照するループで、文字列から取り出した1文字を格納する一時変数。「c」や「ch」くらいの名前にしても十分理解できる。

また、説明量を落とさずに変数名を短くするテクニックとして、「単語の一部を省略する」という方法があります。

例えば、何かの回数を数える時、カウンタ変数として名前に「cnt」をつけた変数がよく使用されます。「cnt」は「count」を略したものです。

単語の一部を省略した形式の変数名として、他にも以下のようなものがよく使用されます。3文字程度に省略したり母音を削ったりして短くすることが多いようです。

単語	省略した形	意味	変数名の例
count	cnt	何かのカウンタ。	outputCnt++;
length	len	何かの長さ。	int maxLen = 10;
index	idx	配列の要素位置等。	int startIdx = (pageNo - 1) * ROW_PER_PAGE;
position	pos	ものの位置。	writePos += strlen(buf);
source	src	何かの元。コピー元等。	char srcStr[] = "コピー元文字列";
destination	dest、dst	行き先。コピー先等。	strcpy(destStr, "abcdef");
message	msg	伝言。メッセージ。	char errMsg[] = "xxx エラーが発生しました。";

3－5. 変数の使いまわしはやめよう

変数にはその変数の役割にあった名前をつけます。個々の変数はその名前の通りの役割をするように使用するべきです。

メモリ領域の節約のため、あるいは別の変数を用意するのが面倒だからといって、その変数の名前にそぐわない別の目的で使用したりしてはいけません。そのような「変数の使い回し」をすると、プログラムの流れが追いづらくなります。また、処理の終わりのほうで参照した値が実は途中で意図しない値に書き換えられていた、というようなバグも発生しやすくなってしまいます。

現在の大容量メモリのコンピュータ環境では、たかだか数バイトの領域を節約してもプログラムのパフォーマンスが向上したりするようなことはありません。よって、読みやすくバグの発生する可能性の低いプログラムにするため、変数の使い回しはやめましょう。それぞれの目的に合った変数を、適切な名前で用意するようにしてください。

×：変数の使い回しをしているソースコード

```
public static void main(String[] args) {

    int[] nums = {34, 25, 21, 90, 58};
    int arrLen = nums.length;

    // 合計値算出
    int value = 0;
    for (int i = 0; i < arrLen; i++) {
        value += nums[i];
    }
    System.out.printf("合計値：%d¥n", value);

    // 平均値算出
    double ave = (double)value / arrLen;
    System.out.printf("平均値：%d¥n", ave);

    // numsの要素を昇順に並べる（選択ソート）
    for (int i = 0; i < arrLen - 1; i++) {
        value = i;

        // i番目以降で最小のデータを探す
        for (int j = i + 1; j < arrLen; j++) {
            if (nums[j] < nums[value]) {
                value = j;
            }
        }

        // 最小データをi番目に移動
        if (value > i) {
            ave = nums[i];
            nums[i] = nums[value];
            nums[value] = (int)ave;
        }
    }
}
```

・合計値を格納したvalue変数を、ソート処理で使い回している。
・valueは「値」という意味なので、インデックス値(i,j)を格納するのは意味的にNG。
・そもそもvalueという名前が不明確で不適切。
・平均値を格納したave変数を、ソート処理で配列要素をスワップする時の一時変数として使い回している。
・aveは平均値を格納するための変数なので、それ以外の値を格納するのは不適切。

○：左記を、適切な変数を使用するように修正した例

```java
public static void main(String[] args) {

    int[] nums = {34, 25, 21, 90, 58};
    int arrLen = nums.length;

    // 合計値算出
    int sum = 0;
    for (int i = 0; i < arrLen; i++) {
        sum += nums[i];
    }
    System.out.printf("合計値：%d¥n", sum);

    // 平均値算出
    double ave = (double)sum / arrLen;
    System.out.printf("平均値：%d¥n", ave);

    // numsの要素を昇順に並べる（選択ソート）
    for (int i = 0; i < arrLen - 1; i++) {
        int minIdx = i;

        // i番目以降で最小のデータを探す
        for (int j = i + 1; j < arrLen; j++) {
            if (nums[j] < nums[minIdx]) {
                minIdx = j;
            }
        }

        // 最小データをi番目に移動
        if (minIdx > i) {
            int tmp = nums[i];
            nums[i] = nums[minIdx];
            nums[minIdx] = tmp;
        }
    }
}
```

・value変数の名前をsumに変更。より明確な名前になったため、役割がはっきり分かるようになった。
・ソート処理において、最小値の要素位置を記憶する変数としてminIdxを定義。より明確なん前になったため、各処理の意図が理解しやすくなった。

・ソート処理で配列要素をスワップする時の変数としてtmpを定義。ただの退避用変数で、これだけスコープが狭いなら、名前はtmpくらいでもOK。

3－6. マジックナンバーに名前をつけよう（定数を定義しよう）

ソースコードに直接記述された「10」や「5000」などの固定の数値は「マジックナンバー」と呼ばれます。

マジックナンバーは、プログラムを記述した直後であれば書いた本人にはおそらくその数字の意味が理解できます。しかし、他人がそのプログラムを読んだ時あるいは時間がたって自分がそのプログラムを読み返した時、マジックナンバーはその意味や意図が理解できず、プログラムの解読が難しくなります。

これを避けるため、マジックナンバーには定数を使って適切な名前をつけるようにしましょう。 適切な名前をつけることにより何の数字なのか、 どのような目的の処理なのかという意図が読む人に伝わるようになるため、読みやすくなります。

×：マジックナンバーが使われているソースコード

```
public void inputPetNames() {

    String[][] petNames = new String[10][21];
    int petCnt = 0;
    String[] inputStr = new String[256];

    // 'q'が入力されるまで名前の入力を繰り返す
    while(true) {

        // 入力ストリームの生成
        BufferedReader buffered = new BufferedReader(new InputStreamReader(System.in));

        // 名前を入力
        System.out.print("ペットの名前を入力してください ('q'で終了) : ");

        inputStr = buffered.readLine();
        int inputLen = inputStr.length;
        if (inputLen <= 0) {
            System.out.println("未入力です。");
            continue;
        }
        if (inputLen > 20) {
            System.out.println("名前が長すぎます。");
            continue;
        }
        if (inputStr.equals("q")) {
            break;
        }

        // ペット名リストに追加
        petNames[petCnt] = new String[inputStr.length];
        for (int i = 0; i < inputStr.length; i++) {
            petNames[petCnt][i] = inputStr[i];
        }
        petCnt++;

        // 入力最大値チェック
        if (petCnt >= 10) {
            System.out.println("入力最大数に達しました。");
            break;
        }
    }

    // 入力されたペットの名前を全て表示
    System.out.println("----------------------------");
    for (int i = 0; i < petCnt; i++) {
        System.out.printf("[%s]¥n", petNames[i]);
    }
}
```

・この「20」という数値が、どういう意味なのかすぐには分からない。
・何の判定なのか分かりづらい。
（比較的簡単な判定式であることと、判定式の結果が真だったときのメッセージ文字列の内容から推測することは可能）

○：定数を使用してマジックナンバーを排除したソースコード

```
    public void inputPetNames() {

        // 定数を定義
        /* 名前の最大文字列長*/
        public static final int NAME_MAX_LEN = 20;

        /* 入力最大数*/
        public static final int INPUT_MAX_COUNT = 10;


        String[][] petNames = new String[INPUT_MAX_COUNT][NAME_MAX_LEN + 1];
        int petCnt = 0;
        String[] inputStr = new String[256];

        // 'q'が入力されるまで名前の入力を繰り返す
        while(true) {

            // 入力ストリームの生成
            BufferedReader buffered = new BufferedReader(new InputStreamReader(System.in));

            // 名前を入力
            System.out.print("ペットの名前を入力してください（'q'で終了）：");

            inputStr = buffered.readLine();
            int inputLen = inputStr.length;
            if (inputLen <= 0) {
                System.out.println("未入力です。");
                continue;
            }
            if (inputLen > NAME_MAX_LEN) {
                System.out.println("名前が長すぎます。");
                continue;
            }
            if (inputStr.equals("q")) {
                break;
            }

            // ペット名リストに追加
            petNames[petCnt] = new String[inputStr.length];
            for (int i = 0; i < inputStr.length; i++) {
                petNames[petCnt][i] = inputStr[i];
            }
            petCnt++;

            // 入力最大値チェック
            if (petCnt >= INPUT_MAX_COUNT) {
                System.out.println("入力最大数に達しました。");
                break;
            }
        }

        // 入力されたペットの名前を全て表示
        System.out.println("--------------------------");
        for (int i = 0; i < petCnt; i++) {
            System.out.printf("[%s]¥n", petNames[i]);
        }
    }
```

定数名より、入力文字列長と名前の最大文字列長の比較を行なっていることが分かる。

定数名より、リスト登録数と入力最大数の比較を行っていることが分かる。

3－7. 一時変数やメソッドを使って式にも名前をつけよう

計算式や判定式にも名前をつけることができます。
例えば計算の結果を一時変数に格納するようにして、その変数に適切な名前をつけるのです。 この場合は計算式というよりは計算結果に対して名前をつけることになります。

また、計算式や判定式が長かったり複雑だったりする場合は、 その処理部分をメソッドとして切り出すと、ごちゃごちゃした記述が消えてより読みやすくなります。 この場合は計算式/判定式自体に名前をつけることになります。

名前をつけることにより、その計算や判定が何のための計算なのか、 何の判定をしているのかが明確になります。 読む人に意図が伝わりやすくなるため、読みやすいプログラムになります。

×：判定式が複雑になっているソースコード

```
if ((cursor.getX() >= btn.getX() && cursor.getX <= (btn.getX() + btn.getWidth()))
    && (cursor.getY() >= btn.getY() && cursor.getY <= (btn.getY() + btn.getHeight()))) {
    // マウスカーソルがボタンの上にあるときの処理
    :
    :
}
```

・if文の判定が長い。
・何の判定をしているのか分かりづらい。

○：上記のソースコードを、一時変数の使用、またはメソッド切り出しをして修正した例

■一時変数を使用して判定結果を格納した場合

```
boolean withinX = cursor.getX() >= btn.getX() && cursor.getX() <= (btn.getX() + btn.getWidth());
boolean withinY = cursor.getY() >= btn.getY() && cursor.getY() <= (btn.getY() + btn.getHeight());
if (withinX && withinY) {
    // マウスカーソルがボタンの上にあるときの処理
    :
    :
}
```

・変数名より、X座標の範囲内指定、Y座標の範囲内指定を行なっていることが分かる。
・if文の判定式が簡潔で理解しやすい。

■判定処理をメソッドに切り出した場合

```
if (btn.contains(cursor.getX(), cursor.getY())) {
    // マウスカーソルがボタンの上にあるときの処理
    :
    :
}
```

・メソッド名と引数より、ボタンがマウスカーソルの座標位置を含むか否かの判定処理を行なっていることが分かる。