第I部 表面上の改善

2章. 名前に情報を詰め込む

2.1 明確な単語を選ぶ

「**get**」はデータの取得元が不明確のため、データの取得元を示す単語を使用するようにする。 例えば、ローカルから取得する場合は、「**load**」を使用し、 インターネットから取得する場合は 「**fetch**」を使用するようにするなど、より目的に合っている明確な単語を選ぶ。

❌ NG例）

```
const getUser = (userId: string) => {
  const user = localStorage.getItem(userId);
  return user;
};
```

🟢 OK例）ローカルからユーザー情報を取得する場合

load は、ローカルからデータを取得する場合に使用する単語であるため、「**loadUser**」というメソッド名を使用する。

```
const loadUser = (userId: string) => {
  const user = localStorage.getItem(userId);
  return user;
};
```

🟢 OK例）インターネットからユーザー情報を取得する場合

fetch は、インターネットからデータを取得する場合に使用する単語であるため、「**fetchUser**」というメソッド名を使用する。

```
const fetchUser = (userId: string) => {
  const user = fetch(`https://api.example.com/users/${userId}`);
  return user;
};
```

命名時のポイントとしては、動作に合わせた明確な単語を選ぶことである。 単語を選ぶ際は、**類語辞典（シソーラス）** から目的により近い単語を選ぶようにする。

類語例）

単語	代替案

単語	代替案
send	deliver, dispatch, announce, distribute, route
find	search, extract, locate, recover
start	launch, create, begin, open
make	create, set up, build, generate, compose, add ,new

2.2 tmp や retVal などの汎用的な名前を避ける

tmp

「tmp」や「retVal」などの **汎用的な名前** を変数名に使用すると、変数の役割がわかりにくくなるため、バグの原因となる可能性がある。 汎用的な名前を使用する場合は、変数の **目的に合った名前** をつけること。

❌ NG例）ユーザー情報を保持する変数の場合

```
let tmp = user.name;
tmp += `${tmp} さんは ${user.age} 歳です`;
console.log(tmp);
```

この変数の目的はユーザー情報を保持することであり、一時的な保管を行うことが重要な目的ではないため、「tmp」という 一時保管を表す変数名は使用すべきでない。

🟢 OK）一時ファイル情報を保持する変数の場合

```
let tmpFile = file.load("tmp.txt");
file.save(tmpFile);
```

この場合は、ファイルを一時的に保管するための変数であるため、tmpFile という変数名を使用することは適切である。

ループイテレーター

ループイテレーターの変数名は、i, j, k などの名前よりも、明確な名前をつけるようにする。

❌ NG例）

ユーザーがどのクラブに属しているかを出力する場合、以下のように変数名を i, j, k としていると、どの変数がどの役割を持っているかわかりにくい。

```
for (var i = 0; i < clubs.length; i++) {
    for (var j = 0; j < clubs[i].members.length; j++) {
        for (var k = 0; k < clubs[i].members[j].users.length; k++) {
            if (clubs[i].members[j].users[k].id === userId) {
                console.log(`${clubs[i].members[j].users[k].name} が所属して
```

```
  いるクラブは ${clubs[i].name}`);
              }
          }
      }
  }
```

🟢 OK例)

以下のように、clubs（クラブリスト）には ci, members（メンバーリスト）には mi, users（ユーザーリスト）には ui という変数名をつけることで、変数の役割がわかりやすくなる。

```
// 検索対象のユーザーID
const userId = 1;

// クラブのメンバーを検索
for (int ci = 0; ci < clubs.length; ci++) {
  for (int mi = 0; mi < members.length; mi++) {
    for (int ui = 0; ui < users.length; ui++) {
      if (clubs[ci].members[mi].users[ui].id === userId) {
        console.log(`${users[ui].name} が所属しているクラブは
${clubs[ci].name}`);
      }
    }
  }
}
```

このように、変数名を説明的にすることで、変数の使用誤りも防ぐことができる。

2.3 抽象的な名前よりも具体的な名前を使う

抽象的な名前でつけられた変数名は、読み手に誤解を与える可能性があるため、名前から変数の役割がわかるようにする。

❌ NG例)

ローカルマシンでテストする場合に使用する環境変数に、以下のような変数名をつけると誤解を招く可能性がある。デバッグモードにすると、何が起こるかわからないため、具体的な名前をつけるようにすべき。

```
DEBUG_MODE
```

🟢 OK例)

ローカルマシンでテストする場合は、ローカルのデータベースに接続するという意味で、以下のように変数名をつけることで、ローカルのデータベースに接続することがわかる。

```
USE_LOCAL_DB_MODE
```

2.4 名前に情報を追加する

名前は短いコメントとして機能するため、変数の役割をわかりやすくするために、変数名に単語を追加する。

例えば、ユーザーIDを表す変数名に、userという単語を追加する。

❌ **NG例)**

```
const id = "user01";
```

🟢 **OK例)**

```
const userId = "user01";
```

値の単位を変数名に含める

変数名に値の単位を含めることで、変数の役割がわかりやすくなる。

❌ **NG例)**

単位が含まれていないため、秒で出力したいところが、ミリ秒で出力されてしまう。

```
const start = (new Date()).getTime();

・・・何らかの処理・・・

const elapsed = (new Date()).getTime() - start;
console.log(`読み込み時間は ${elapsed} 秒`); // 読み込み時間は 2000 秒
```

🟢 **OK例)**

ミリ秒で取得されるため、変数名に単位（ms）を含めることで、開発者が秒で出力するために、ミリ秒を秒に変換する必要があることがわかる。

```
const start_ms = (new Date()).getTime();

・・・何らかの処理・・・

const elapsed_ms = (new Date()).getTime() - start_ms;
console.log(`読み込み時間は ${elapsed_ms / 1000} 秒`); // 読み込み時間は 2 秒
```

2.5 名前の長さを決める

スコープが小さければ短い名前でもいい

スコープが小さい場合は、見える範囲が狭く、短い名前でも変数の役割がわかるため、短い名前を使用しても問題ない。

コード例）

m という変数名は、スコープが小さいため、変数の役割がわかる。

```
if (debug) {
  const m = new Map();
  m.set("name", "太郎");
  m.set("age", 20);
  setUser(m);
}
```

スコープが長い場合は見える範囲が広くなり、変数の役割がわかりにくくなるため、長い名前を使用することにより変数の役割がわかりやすくなる。

コード例）

user という具体的な名前を使用することで、スコープが長くなったとしても、変数の役割がわかる。

```
const user = new Map();
user.set("name", "太郎");
user.set("age", 20);

・・・長いコード・・・

setUser(user);
```

頭文字と省略形

変数名を短くするために、頭文字や省略形を使用することがあるが、以下のルールに従うこと。

- プロジェクト固有の省略形はダメ 例えば、プロジェクト内では BackEndManager というクラスを BEManager と省略している場合、 BE はプロジェクト固有の省略形であるため、新しくプロジェクトに参画した人は、BEManager が何を表しているかわからない。
- 一般的な省略形はOK プログラマの共通認識として、例えば、以下のような省略形は一般的な省略形であるため、使用しても問題ない。

  一般的な省略形の例）

単語	省略形
evaluation	eval
document	doc
string	str

このような、一般的に認知されている省略形であれば使用しても問題ない。

不要な単語を捨てる

例えば、ConvertToString を ToString に変更しても、必要な情報（ここでは文字列に変換する）が失われない場合は、省略してもよい。