品質の高いコードとは？

~リーダブルコード~

講師：石井

アジェンダ

・自己紹介

・リーダブルコードについて

・今回伝えたいこと

・QAタイム

・リーダブルコードの中から紹介

・開発体験

・まとめ

自己紹介

石井　治樹（いしい　はるき）

エンジニア歴：3年目

好きな言語：PHP, JS, Swift

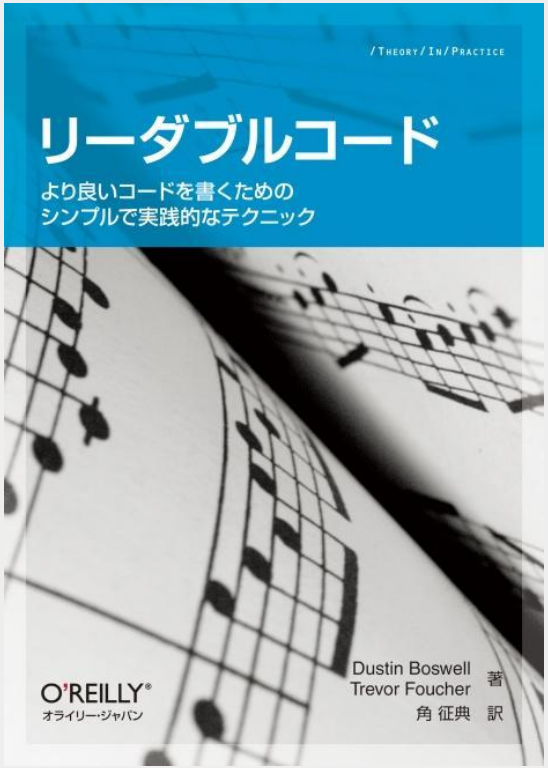
趣味：お酒、温泉巡り

リーダブルコードとは

・より良いコードを書くための
　　シンプルで実践的なテクニック

・ITエンジニアに読んで欲しい技書2014 大賞受賞

・名著、良書として広く有名



今回伝えたいこと

・「リーダブルコード」から以下の観点から項目をピックアップ

- 可読性や保守性を上げるのに必要な項目
- 説明しやすい項目

・今回の勉強会を通して、「リーダブルコード」を読むきっかけにしてほしい

※あくまで一つの手段として

QAタイム（5分間）

下記のリンクからリーダブルコードに関する問題を解いてください

https://docs.google.com/forms/d/e/1FAIpQLSdPxanzcqIPVL4tHPLY2FjDadmSr54IrnW8Obr0BEATecKxUw/viewform

1章　理解しやすいコード

・コードは理解しやすくなければならない

→コードを書く上で最も大事な規則

・コードは他の人が最短時間で理解できるように書かなければならない

→読みやすさの基本定理を抑える

・「理解するまでにかかる時間」は他の目標と競合しない

→優れた設計やテストのしやすさに繋がる

1章　理解しやすいコード

・読みやすさの基本定理

コードを短くする ≠ 理解するまでにかかる時間を短くする

コードを短くするポイント

・不要なコードの削除　→　不必要な関数や使用していないコードの削除

・過剰な機能を持たせない　→　メソッドやクラスで分ける

2章　名前に情報を詰め込む

・名前のフォーマットで情報を伝える

→ アンダースコア・ダッシュ・大文字を使って、名前に情報を詰め込むことができる。

・明確で正確な単語を選ぶ

・汎用的な名前を避ける

・名前の長さを決める

明確で正確な単語を選ぶ 1/2

例:

GetPage()

疑問点:

・このページはどこから取ってくる？

（ローカルキャッシュ？データベース？インターネット？）

インターネットからなら　FetchPage()やDownloadPage()の方が適当

明確で正確な単語を選ぶ 2/2

単語を選択する時のヒント

・シソーラス（類語辞典）を使用する

例：

- send：deliver, dispatch, announce, distribute, route
- find：search, extract, locate, recover
- start：launch, create, open

・人に質問する（選択した単語で意図が伝わるか）

汎用的な名前を避ける

・retval / fooのような汎用的な名前を避ける

→エンティティの値や目的を表した名前を選ぼう

・tmpを使う場合

→tmpという名前は、生存期間が短くて、

一時的な補完が最も大切な変数にだけ使う

名前の長さを決める

・不要な単語を切り捨てる

例:

doServerLoop() -> serverLoop()

convertToString() -> toString()

・スコープの大きな変数には長い名前をつける

・スコープが小さければ短い名前でもいい

命名規則に困ったら、、、

用途に応じて名前を考えてくれるツールもあります。

3章 誤解されない名前

・名前が「他の意味と間違えられることはないだろうか？」と何度も自問自答

・限界値を示す時はminとmaxを使用する

・範囲を示すときはfirstとlastを使う

・包括/排他的範囲にはbeginとendを使う

→ 対義語に注意:startの対義語はendではなく、stop

参考：http://webspace.jugem.jp/?eid=947

他の意味と間違えられないように

例:

result = example.filter(“year <= 2012”)

疑問点:

・「year <= 2012」のオブジェクトか「year <= 2012」ではないオブジェクト

どちらなのか分からない

→ 選択するなら「select」

除外するなら「exclude()」にした方がよい

ブール値の名前

例:

bool read_password = true;

疑問点:

・パスワードをこれから読み取る必要がある？パスワードをすでに読み取っている？

どちらかわからない。

→ need_passwordやuser_is_authenticateを使った方がよい

ポイント:

ブール値の変数名は、頭に is/has/can/shouldなどをつけてわかりやすくする

4章 美しさ

3つの原則

・読み手が慣れているパターンと一貫性のあるレイアウトを使う

→ 1行辺りの文字数・改行・インデントを意識する

・似ているコードは似ているように見せる

→ 複数のコードブロックで同じようなことをしていたら、シルエットも似せる

・関連するコードをまとめてブロックする

→ メソッドを使った整列や単位を使用して可読性を上げる

4章 美しさ

どちらが読みやすい？？

```
def suggest_new_friends (user, email_password):
    friends = user.friends()
    friend_emals = set(f.email for f in friends)
    contacts = import_contacts(user.email, email_password);
    contact_emals = set(c.email for c in contacts)
    non_friend_emails = contact_emals - friend_emails
    suggested_friends = USer.objects.select(email__in=non_friend_emails)
    display['user'] = user
    display['friends'] = friends
    display['suggested_friends'] = suggested_friends
    return render("suggested_friends.html", display)
```

```
def suggest_new_friends (user, email_password):
    # ユーザの友達のメールアドレスを取得する。
    friends = user.friends()
    friend_emals = set(f.email for f in friends)

    # ユーザのメールアカウントからすべてのメールアドレスをインポートする。
    contacts = import_contacts(user.email, email_password);
    contact_emals = set(c.email for c in contacts)

    # まだ友達になっていないユーザを探す。
    non_friend_emails = contact_emals - friend_emails
    suggested_friends = USer.objects.select(email__in=non_friend_emails)

    # それをページに表示する
    display['user'] = user
    display['friends'] = friends
    display['suggested_friends'] = suggested_friends
    return render("suggested_friends.html", display)
```

5章 コメントすべきことを知る

・コメントするべきことでは「ない」こと

・自分の考えを記録する

コメントするべきでは「ない」こと

・コードからすぐに分かることをコメントに書かない

例：

```
public void main(String[] args) throws Exception {
    // 名前
    String name;

    // 年齢
    String age = 24;

    // 名前と年齢を連結して出力する
    System.out.println("名前:" + name + "年齢:" + age);
}
```

・ひどい名前はコメントをつけずに名前を変える

→「優れたコード」 ＞ 「ひどいコード ＋ 優れたコメント」

7章　制御フローを読みやすくする

・条件の引数の並び順・条件式の優劣を気にかける

・関数から早く返す

・三項演算子を効果的に使う

条件の引数の並び順・条件式の優劣をつける

・条件式の引数の並び順

→ 左側：「調査対象」の式　変化する

→ 右側：「比較対象」の式　あまり変化しない

例：

○

```
if (length > 10) {}
```

×

```
if (10 < length) {}
```

関数から早く返す

・行数を短くするよりも他の人が理解するのにかかる時間を短くする

・ネストを浅くする

・関数の出口は複数あっても良い

三項演算子を効果的に使おう

・十分に短く書くことができて、
理解しやすい場合に限り三項演算子を使うとよい

NG例:

```
return exponent >= 0 ?
    mantissa * ( 1 << exponent) : mantissa / (1 << -exponent);
```

どちらが読みやすい？？

```
if(!url.hasParameter("expand_all")) {

} else {

}

if(url.hasParameter("expand_all")) {

} else {

}
```

- 否定形よりは肯定形を使う
  if(debug) がわかりやすい
- 単純な条件を先に書くようにする
- 関心を引く条件や目立つ条件を先に書く

9章 変数と読みやすさ

・変数のスコープを縮める

→　例えばグローバル変数は、どこでどのように使われるか追跡する必要があり、

　　把握するのに時間がかかってしまう

・定義の位置を下げる

→　変数の宣言、定義は使う直前に

開発体験（HTML,CSS,JSを使用したTODOアプリ）

準備1

・以下のURLからgit cloneする

```
$ git clone https://github.com/haru507/study_meet_app
```

※開発体験の注意点

個人で、Firebaseの登録をして認証情報を配ります。
そのため、リクエストできる回数について規定値を超えればリクエストができなくなるのでご了承ください。

また、開発体験終了後に認証情報は使えなくするため自分自身で登録してください。

まとめ

・「理解しやすいコード」は保守性・可読性を上げることができる

・リーダブルコードのやり方に完全に従うのではなく

プロジェクトやチームのやり方の中に取り入れていく

・今回をきっかけにリーダブルコードを読んでみよう！！