ClearCode

リーダブルコードを読み解こう

3章 誤解されない名前

須藤功平

株式会社クリアコード

schoo

2014/12/9

質問（1）

プログラミングについて

- ✓ A：未経験
- ✓ B：学習中（schooや学校、独学など）
- ✓ C：趣味・仕事でたまに書く
  （趣味でWebサイトを作っている、職業がデザイナーなど）
- ✓ D：趣味・仕事でバリバリ書く
  （趣味でOSSを開発している、職業がエンジニアなど）

質問（2）

リーダブルコード（本）を…

- ✓ A：読んだ
- ✓ B：読んでいる
- ✓ C：まだ読んでいない

内容

- ✓ 自己紹介
- ✓ リーダブルコードとは
- ✓ 実例で考えよう
- ✓ 実際の改善にチャレンジ！
- ✓ まとめ
- ✓ 質疑応答

自己紹介（1）

- ✓ リーダブルコードの「解説」の著者
  http://www.clear-code.com/blog/2012/6/11.html

自己紹介（2）

- クリアコードの代表取締役
  - 「クリア」な（意図が明確な）
    「コード」を大事にする
    ソフトウェア開発会社

自己紹介（3）

Kouhei Sutou
kou

ClearCode Inc.
Tokyo, Japan
kou@clear-code.com
http://www.clear-code.com/
Joined on 3 Oct 2008

106	55	0
Followers	Starred	Following

Organizations

Contributions　Repositories　Public activity　Edit profile

Popular repositories

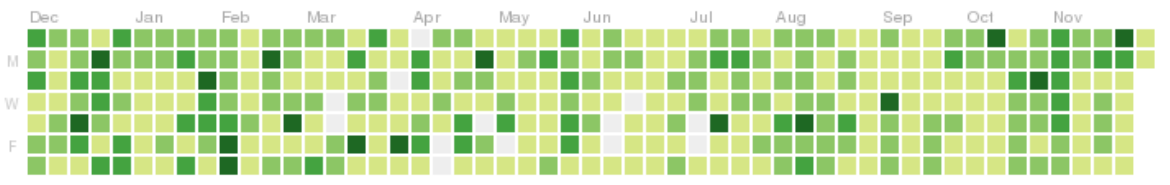
- **bundle-milkode** — Make all gems installed by Bundler milkable — 7 ★
- **mruby-pp** — pp for mruby — 4 ★
- **segv-handler-gdb** — Dump C level backtrace by GDB on SEGV — 4 ★
- **rubyinstaller** — RubyInstaller for Windows - Building recipes — 3 ★
- **misawa** — 3 ★

Repositories contributed to

- groonga/**groonga** — An embeddable fulltext search engine. Groon… — 224 ★
- droonga/**droonga-engine** — Droonga engine is a core component in Droo… — 5 ★
- ruby-gnome2/**ruby-gnome2** — A set of bindings for the GNOME-2.x libraries … — 111 ★
- mroonga/**mroonga** — A MySQL pluggable storage engine based o… — 81 ★
- project-hatohol/**hatohol** — 52 ★

毎日コードを書いている

Contributions



Summary of Pull Requests, issues opened, and commits. Learn more.　Less　More

Year of contributions	Longest streak	Current streak
7,271 total	143 days	143 days
Dec 1, 2013 – Dec 1, 2014	July 12 – December 1	July 12 – December 1

リーダブルコードとは（1）

> 本書の目的は、君のコードをよくすることだ
>
> ［「はじめに p. x」より引用］

リーダブルコードとは（2）

> その中心となるのは、コードは理解しやすくなければいけないという考えだ
>
> ［「はじめに p. x」より引用］

リーダブルコードとは（3）

> 「コードを理解する」というのは、変更を加えたりバグを見つけたりできるという意味
>
> ［「1.2 読みやすさの基本定理 p. 3」より引用］

リーダブルコード

- ✓ 変更できるコード
- ✓ バグを見つけられるコード

↓

読む人視点！

何をしているコード？

```
Node* node = list->head;
if (node == NULL) return;

while (node->next != NULL) {
    Print(node->data);
    node = node->next;
}
if (node != NULL) Print(node->data);
```

「優れた」コードって何？ p. 2より

何をしているコード？

```
for (Node* node = list->head;
     node != NULL;
     node = node->next)
    Print(node->data);
```

「優れた」コードって何？ p. 2より

どちらがリーダブル？

```
// どちらがリーダブルコード？どうして？
// リーダブルコード：変更できる・バグを見つけられるコード
// A.
Node* node = list->head;
if (node == NULL) return;
while (node->next != NULL) {
    Print(node->data);
    node = node->next;
}
if (node != NULL) Print(node->data);
// B.
for (Node* node = list->head; node != NULL; node = node->next)
    Print(node->data);
```

「優れた」コードって何？ p. 2より

実例で考えよう

3章「誤解されない名前」
より

3.1 例：filter() (1)

```
results =
  Database.all_objects.filter("year <= 2011")
```

resultsはなに？

- ✓ 2011年以前のレコード？
- ✓ 2012年以降のレコード？

3.1 例：filter() (2)

- filterはあいまいな言葉
  - 選択する→select()
  - 除外する→exclude()

```
select("year <= 2011")
exclude("year <= 2011")
```

番外1：sort() (1)

- どっちの名前がよい？
  - sort()
  - quick_sort()/merge_sort()

番外1：sort()（2）

- ソートすることが重要
  - sort()
- ソート方法が重要
  - quick_sort()/merge_sort()

ヒント：抽象化・カプセル化
(abstraction barrier, abstraction layer)

3.2 例：Clip()（1）

```
# textの最後を切り落として、
# 「...」をつける
def Clip(text, length):
```

- ✓ 最後からlength文字を削除する？
- ✓ 最大length文字まで切り詰める？

3.2 例：Clip() (2)

切り詰めるなら：Clip→Truncate

```
def Clip(text, length):
# ↓
def Truncate(text, length):
```

3.2 例：Clip() (3)

最大length文字なら：maxをつける

```
def Truncate(text, length):
# ↓
def Truncate(text, max_length):
```

3.2 例：Clip() (4)

最大length**文字**なら：
length→chars

```
def Truncate(text, max_length):
# ↓
def Truncate(text, max_chars):
```

番外2：add()（1）

- どの名前がよい？
  - add()
  - append()/prepend()

番外2：add()（2）

- ✓ **追加すること**が重要
  例：集合に追加するとき
  - ✓ add()
- ✓ 追加する**場所**が重要
  例：リストに追加するとき
  - ✓ append()/prepend()

ヒント：抽象化・カプセル化
(abstraction barrier, abstraction layer)

3.3 限界値（1）

ショッピングカートに入る商品が最大10点のケース

```
CART_TOO_BIG_LIMIT = 10

if cart.num_items() >= CART_TOO_BIG_LIMIT:
    Error("カートにある商品数が多すぎます。")
```

3.3 限界値（2）

バグあり！
9点までしか入らない

```
CART_TOO_BIG_LIMIT = 10

if cart.num_items() >= CART_TOO_BIG_LIMIT:
    Error("カートにある商品数が多すぎます。")
```

3.3 限界値（3）

- CART_TOO_BIG_LIMITはあいまい
  - 未満？以下？→境界値の情報がない
- MAX_ITEMS_IN_CARTは明確
  - この値が最大値（境界値を含む）
  - 最大値のときはmaxを使おう！
    （最小値のときはmin）

3.3 限界値（4）

```
# CART_TOO_BIG_LIMIT = 10
# if cart.num_items() >= CART_TOO_BIG_LIMIT:
#     Error("カートにある商品数が多すぎます。")
MAX_ITEMS_IN_CART = 10
if cart.num_items() > MAX_ITEMS_IN_CART:
    Error("カートにある商品数が多すぎます。")
```

他の例

- 範囲を指定するときは
  firstとlastを使う
- 包含／排他的範囲には
  beginとendを使う
- …
- （詳細は本を買ってください）

実際の改善にチャレンジ！

```
/* schooの学生名一覧を返す
   引数は返す学生名の最大値 */
function student($num) {
  /* 処理の具体的な内容 */
}
```

- ✓ 誤解されない名前に改善して投稿
- ✓ よい投稿に「いいね！」して応援

まとめ（1）

- ✓ リーダブルコードとは
  - ✓ 変更できるコード
  - ✓ バグを見つけられるコード
  - ✓ ↑は**読む人視点**

まとめ（2）

- 「誤解されない名前」を考えた
  - filterはあいまい
    - →select/exclude
  - Clipもあいまい
    - →Truncate
  - TOO_BIG_LIMITもあいまい
    - →MAX_ITEMS_IN_CART

まとめ（3）

- ✓ 実際の改善にチャレンジした
  - ✓ 「**読む人**が理解しやすいか？」をとことん考えたはず

> *名前が「他の意味と間違えられることはないだろうか？」と何度も自問自答する*
>
> *［「3章 誤解されない名前 p. 30」より引用］*

これから（1）

- これからも**読む人**のことを考えてコードを書こう
- **読む人**のことを考えるには？
  - 読む経験をたくさん積む
  - たくさんコードを読もう

これから（2）

- ✓ たくさんコードを読むコツ
  - ✓ コードから学ぶ気持ちで読む
  - ✓ ×悪いこと探し
  - ✓ ○いいこと探し
  - ✓ 本来、コードを読むことは楽しいことのはず！

悪いコード

- 見つけやすい
  - 異質
  - リーダブルじゃない
- 過剰に指摘したくなる
  - 「マサカリを投げてみたい」

マサカリ投げたい症候群

早めに卒業しましょう

- ✓ 必要十分な事実伝達以外の否定的な情報を過剰に含む指摘
  必要十分な事実伝達がない場合もある
  - ✓ 必要十分な事実伝達：○○な理由で××ではなく△△だ
    本に書いているから、は理由になっていない

↑は最近の「マサカリを投げる」の捉えられ方
参考：元々のモヒカン・手斧の使われ方：
http://www.otsune.com/diary/2005/06/14/4.html#200506144

よいコード

- 見つけにくい
  - リーダブルだから
  - すーっと理解できてひっかからない
- これからのチャレンジ
  - 意識して見つけよう！

これから（3）

「解説」を読む

http://www.clear-code.com/blog/2012/6/11.html

- ✓ 本文：**個人**で
  リーダブルコードを書く方法
- ✓ 解説：**チーム**で
  リーダブルコードを書く方法