データ指向プログラミング(仮)のススメ

…

CBcloud株式会社
新垣 雄志(あらかき ゆうじ)

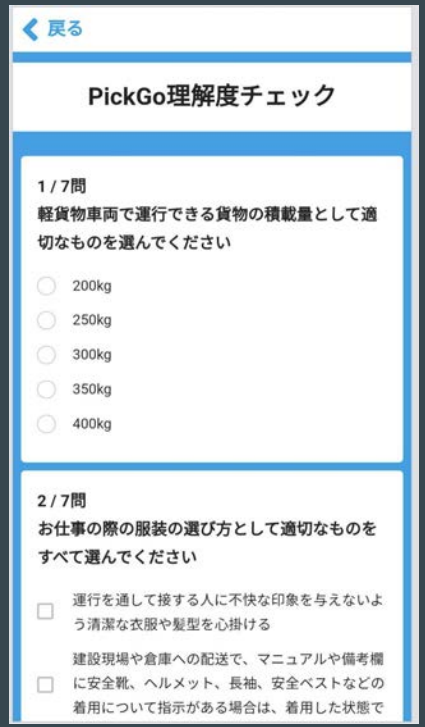
自己紹介

- 新垣 雄志(あらかき ゆうじ)
- Twitter: @arakaji
- 職歴
  - 琉球インタラクティブ株式会社
  - 株式会社Payke
  - CBcloud株式会社 ← NOW
- CBcloud株式会社の職務内容
  - バックエンドのリードエンジニア
  - Ruby on Rails
  - AWS
- 好きなプログラミング言語
  - Clojure

今日話したいこと

- ある仕様を実現したいときに、UIや振る舞いから実装を考えるのではなく、その仕様を適切に表現するデータ構造から考えて実装すると汎用的で保守しやすい機能として実装しやすくなるよ！
- これを示す言葉として「データ指向プログラミング(仮)」と僕が勝手に呼んでいます。
    - 別のことを示す言葉としてすでに存在するかもしれませんが、一旦この場はご容赦ください

例: PickGo理解度チェック



- 当社のサービスに登録していただいたドライバーに、初めての運行前にサービスのルールについて理解しているかどうかをチェックするテストをうけてもらう機能
- 配送品質向上のための施策として実装された
- 要件
  - 問題が表示される
  - 答えの選択肢が表示される
  - 回答をすべて選択したあと、次の画面でどれが正解でどれが不正解か表示される
  - 各問題の正解が一つのことも、2つ以上あることもある

例: PickGo理解度チェック

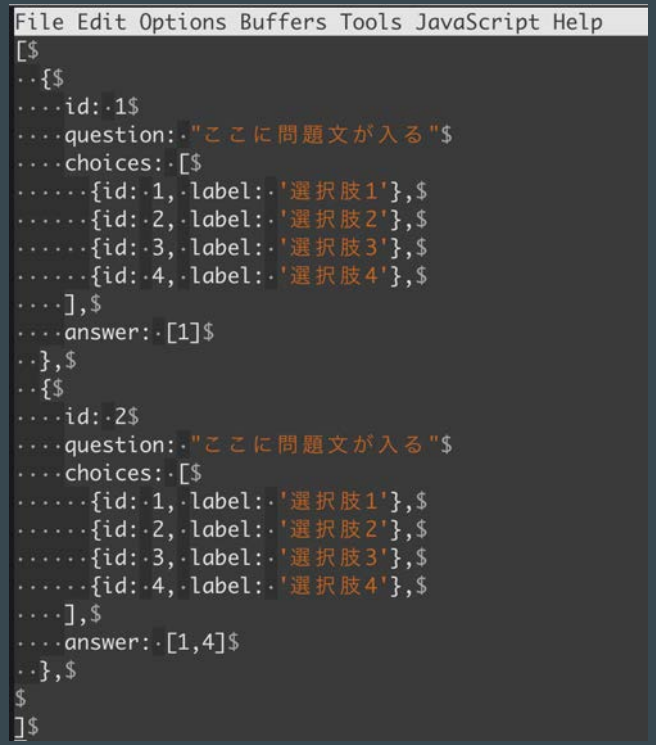
- 拡張性、保守性を考えず素直に実装すると
  - 表示される問題ページの UIをコーディング
  - 各問題毎に正解・不正解をチェックするコードを書く
    - それを各問題毎に実装
- 何が問題になるか？
  - 問題が増えるごとに実装を増やさないといけない
  - 答えが変わると実装を変更しないといけない
  - つらい・・・

例: PickGo理解度チェック

```
File Edit Options Buffers Tools JavaScript Help
[$
··{$
····id:·1$
····question:·"ここに問題文が入る"$
····choices:·[$
·····{id:·1,·label:·'選択肢1'},$
·····{id:·2,·label:·'選択肢2'},$
·····{id:·3,·label:·'選択肢3'},$
·····{id:·4,·label:·'選択肢4'},$
····],$
····answer:·[1]$
··},$
··{$
····id:·2$
····question:·"ここに問題文が入る"$
····choices:·[$
·····{id:·1,·label:·'選択肢1'},$
·····{id:·2,·label:·'選択肢2'},$
·····{id:·3,·label:·'選択肢3'},$
·····{id:·4,·label:·'選択肢4'},$
····],$
····answer:·[1,4]$
··},$
$
]$
```

- データ指向プログラミング的に実装する場合
  - まず理解度チェックという概念を表現するデータ構造を考える
    - 問題
      - 問題文がある
      - 選択肢がある
        - idとラベル(表示される文章)がある
      - 選択肢のうちどれが正解かがある
        - その答えは複数ある
    - 問題が複数ある

例: PickGo理解度チェック



- データ構造を元にして実装をすすめる
  - データ元にして画面(HTML)を表示するプログラムを作る
  - ユーザーが選択したものが正解かどうかをデータを元にしてチェックするプログラムを書く
- これによって以下のメリットが得られる
  - 問題、答えが変わってもデータだけ変更すればよく実装の変更はいらない
  - 問題が増えてもデータを増やすだけでよく、実装の変更はいらない
- 楽ちん。。。

え、、、当たり前じゃね。。。

意外と当たり前にやられていない

- DBのデータを表示する系の場合は当たり前に行われている
  - 商品一覧の表示
  - 商品詳細画面の表示
  - etc
- しかし、DBで値を管理しない機能の場合にも、同じ思考で考えられるかというとそうでない方も多い。
- DBに値があるなしとわず、システムの振る舞いや表示されるUIとそれらを表現するデータ構造を分けて考えるのは意識しないと意外とできてない。

応用編：データからプログラムを生成する

例: 都道府県を扱うクラス

- 値オブジェクトとして日本の都道府県を扱うクラスを作りたい
  - 沖縄県や東京都など都道府県として有効な文字列のみ正常にインスタンス化出来るオブジェクト
  - クラスメソッドとして指定した都道府県をインスタンス化したオブジェクトを返すメソッドがほしい

```
irb(main):123:0> Prefecture.new("東京都")
=> #<Prefecture:0x00007fe0bd06d060 @value="東京都">
irb(main):124:0> Prefecture.new("テスト")
Traceback (most recent call last):
        4: from /Users/y.arakaki/.rbenv/versions/2.5.1/bin/irb:11:in `<main>'
        3: from (irb):124
        2: from (irb):124:in `new'
        1: from (irb):73:in `initialize'
ArgumentError (日本の都道府県を表す文字列しか受け付けません)
irb(main):125:0> Prefecture.okinawa
=> #<Prefecture:0x00007fe0bd03b970 @value="沖縄県">
irb(main):126:0> Prefecture.itto_sanken
=> [#<Prefecture:0x00007fe0bc8bfa48 @value="東京都">, #<Prefecture:0x00007fe0bc8bf9a8 @value="神奈川県">,
0bc8bf868 @value="千葉県">]
```

例: 都道府県を扱うクラス

- 素直に実装すると
  - okinawaメソッドを定義して沖縄県というデータのインスタンスを返す
    - これを愚直に繰り返す
    - 都道府県は変更が少ないデータなのでありではあるが、、、
    - 面倒くさくない？

例: 都道府県を扱うクラス

```
#$
#·日本の都道府県を表現するモデル。$
#·DDDでいうところの値オブジェクトとして設計している。$
#$
class·Prefecture$
··attr_reader·:value$
$
··VALUES·=·{$
····hokkaido:·'北海道',$
····aomori:·'青森県',$
····iwate:·'岩手県',$
····miyagi:·'宮城県',$
····akita:·'秋田県',$
····yamagata:·'山形県',$
····fukushima:·'福島県',$
····ibaragi:·'茨城県',$
····tochigi:·'栃木県',$
····gunma:·'群馬県',$
····saitama:·'埼玉県',$
····chiba:·'千葉県',$
```

- 都道府県を表現するデータ構造を考える
  - プログラム上で扱う都道府県の名前 (key)と人間が見る都道府県の名前(ラベル)の一覧があればよさそう。
  - そのデータをもとに各メソッドの実装を生成すればよい

例: 都道府県を扱うクラス

```
··#$
··#·VALUESのkey名と同じクラスメソッドを定義$
··#·このメソッドの返り値はそのkeyに紐付いている値のPrefectureインスタン\\
ス$
··#·example:$
··#···Prefecture.tokyo·#=>·#<Prefecture:0x000056297e0c8098·@value="東京\\
都">$
··#···Prefecture.okinawa·#=>·#<Prefecture:0x000056297e0c8098·@value="沖\\
縄県">$
··#$
··VALUES.each·do·|key,·val|$
····define_singleton_method(key)·do$
·····new(val)$
····end$
··end$
$
··def·initialize(v)$
····unless·self.class::VALUES.values.include?(v)$
·····raise·ArgumentError,·'日本の都道府県を表す文字列しか受け付けません\
'$
····end$
····@value·=·v$
··end$
$
```

- 都道府県を表現するデータ構造を考える
  - プログラム上で扱う都道府県の名前 (key)と人間が見る都道府県の名前(ラベル)の一覧があればよさそう。
  - そのデータをもとに各メソッドの実装を生成すればよい
  - そのデータに含まれていない値で初期化しようとしても例外吐く
- 都道府県が増えても減ってもデータを変更すれば大丈夫！！！

まとめ

- ある仕様を実現したいときに、UIや振る舞いから実装を考えるのではなく、その仕様を適切に表現するデータ構造から考えて実装すると拡張しやすく、保守しやすい機能として実装しやすくなるよ！
- 考えるときのコツ
  - 仕様を表現するデータ構造はどんなものか？
  - 拡張するときにデータ変更のみで実現できるようにするにはどうすればいいか？

CM

ソフトウェアエンジニア採用してます！！！

- 荷主とドライバーをマッチングするサービスを提供(物流 x Tech)
- 創業者が沖縄出身
- 東京・大阪・沖縄に拠点がある
- 沖縄拠点のエンジニアを来年10人くらいにしたい(今二人)ので興味がある方はぜひ！
- Rails, Go, Flutter, Nuxt.js, AWS